

Probabilistic Search for Tracking Targets

Probabilistic Search for Tracking Targets

Theory and Modern Applications

Eugene Kagan

*Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Israel*

Irad Ben-Gal

Department of Industrial Engineering, Tel-Aviv University, Israel



A John Wiley & Sons, Ltd., Publication

This edition first published 2013
© 2013 John Wiley & Sons, Ltd

Registered office

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

The right of the author to be identified as the author of this work has been asserted in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book. This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Library of Congress Cataloging-in-Publication Data

Kagan, Eugene.

Probabilistic search for tracking targets : theory and modern application / Eugene Kagan, Irad Ben-Gal.
pages cm

Includes bibliographical references and index.

ISBN 978-0-470-97393-6 (cloth)

1. Search theory. 2. Target acquisition—Mathematics. 3. Motion—Mathematical models. I. Ben-Gal, Irad.
II. Title.

T57.97.K34 2013
658.4'034—dc23

2012051596

A catalogue record for this book is available from the British Library.

ISBN: 978-0-470-97393-6

Typeset in 10/12pt Times by Laserwords Private Limited, Chennai, India

To our beloved families

Contents

List of figures	xi
Preface	xv
Notation and terms	xvii
1 Introduction	1
1.1 Motivation and applications	4
1.2 General description of the search problem	5
1.3 Solution approaches in the literature	7
1.4 Methods of local search	11
1.5 Objectives and structure of the book	14
References	15
2 Problem of search for static and moving targets	19
2.1 Methods of search and screening	20
2.1.1 General definitions and notation	20
2.1.2 Target location density for a Markovian search	24
2.1.3 The search-planning problem	30
2.2 Group-testing search	55
2.2.1 General definitions and notation	56
2.2.2 Combinatorial group-testing search for static targets	63
2.2.3 Search with unknown number of targets and erroneous observations	71
2.2.4 Basic information theory search with known location probabilities	84
2.3 Path planning and search over graphs	108
2.3.1 General BF* and A* algorithms	109
2.3.2 Real-time search and learning real-time A* algorithm	122
2.3.3 Moving target search and the fringe-retrieving A* algorithm	131
2.4 Summary	140
References	140

3 Models of search and decision making	145
3.1 Model of search based on MDP	146
3.1.1 General definitions	146
3.1.2 Search with probabilistic and informational decision rules	152
3.2 Partially observable MDP model and dynamic programming approach	161
3.2.1 MDP with uncertain observations	162
3.2.2 Simple Pollock model of search	166
3.2.3 Ross model with single-point observations	174
3.3 Models of moving target search with constrained paths	179
3.3.1 Eagle model with finite and infinite horizons	180
3.3.2 Branch-and-bound procedure of constrained search with single searcher	184
3.3.3 Constrained path search with multiple searchers	189
3.4 Game theory models of search	192
3.4.1 Game theory model of search and screening	192
3.4.2 Probabilistic pursuit-evasion games	201
3.4.3 Pursuit-evasion games on graphs	206
3.5 Summary	214
References	215
4 Methods of information theory search	218
4.1 Entropy and informational distances between partitions	219
4.2 Static target search: Informational LRTA* algorithm	227
4.2.1 Informational LRTA* algorithm and its properties	228
4.2.2 Group-testing search using the ILRTA* algorithm	234
4.2.3 Search by the ILRTA* algorithm with multiple searchers	244
4.3 Moving target search: Informational moving target search algorithm	254
4.3.1 The informational MTS algorithm and its properties	254
4.3.2 Simple search using the IMTS algorithm	260
4.3.3 Dependence of the IMTS algorithm's actions on the target's movement	269
4.4 Remarks on programming of the ILRTA* and IMTS algorithms	270
4.4.1 Data structures	270
4.4.2 Operations and algorithms	282
4.5 Summary	290
References	290
5 Applications and perspectives	293
5.1 Creating classification trees by using the recursive ILRTA* algorithm	293
5.1.1 Recursive ILRTA* algorithm	294
5.1.2 Recursive ILRTA* with weighted distances and simulation results	297
5.2 Informational search and screening algorithm with single and multiple searchers	299
5.2.1 Definitions and assumptions	299
5.2.2 Outline of the algorithm and related functions	300
5.2.3 Numerical simulations of search with single and multiple searchers	304
5.3 Application of the ILRTA* algorithm for navigation of mobile robots	305

5.4 Application of the IMTS algorithm for paging in cellular networks	310
5.5 Remark on application of search algorithms for group testing	312
References	313
6 Final remarks	316
References	317
Index	319

List of figures

Figure 2.1	Example of gridded sample space and its associated location probabilities	21
Figure 2.2	A general decision process for the probabilistic search procedure	22
Figure 2.3	Examples of search effort distribution	23
Figure 2.4	Initial and final target location density without search. (a) Initial target location density. (b) Target location density without search at time $t = 100$	28
Figure 2.5	Detection probabilities concentrated in the center of the domain and the corresponding target location probabilities. (a) Detection probabilities concentrated in the center of the domain. (b) Target location density at time $t = 100$	28
Figure 2.6	Detection probabilities concentrated in the center of the domain and the corresponding target location probabilities. (a) Detection probabilities concentrated at the sides the domain. (b) Target location density at time $t = 100$	29
Figure 2.7	Search agent trajectories and observed areas. (a) Continuous space and time. (b) Discrete space and time	30
Figure 2.8	Search agent trajectory and search density. (a) Detection probability at starting point $x^{t=0} = (3, 3)$ and a search agent trajectory up to time $t = 100$. (b) Search density $u^{t=100}$ or allocation w at time $t = 100$	31
Figure 2.9	The search density along the search trajectory and the search effort at an arbitrary point. (a) Search density along the agent's trajectory. (b) The search effort as applied to point $x = (5, 5)$ as a funtion of time	32
Figure 2.10	Search density defined by an optimal allocation of efforts according to Equation 2.6. (a) Search density $u^{t=0} = w^*$; available search effort $K = 10$. (b) Search density $u^{t=0} = w^*$; available search effort $K = 100$	37
Figure 2.11	Dynamics of a Markovian target location density and the corresponding search density for an optimal allocation following Equation 2.6. (a) Target location density $v^{t=0}$. (b) Search density $u^{t=0}$; available search effort $K = 10$. (c) Target location density $v^{t=10}$. (d) Search density $u^{t=10}$; available search effort $K = 100$. (e) Target	

location density $v^{t=100}$. (f) Search density $u^{t=100}$; available search effort $K = 1000$	38
Figure 2.12 A search density that corresponds to an optimal allocation as obtained by Algorithm 2.1. (a) Search density $u^{t=0} = w^*$; available search effort $K = 10$. (b) Search density $u^{t=0} = w^*$; available search effort $K = 100$	42
Figure 2.13 Dynamics of Markovian target location density and search density for optimal allocations provided by Algorithm 2.2. (a) Target location density $v^{t=0}$. (b) Search density $u^{t=0}$; available search effort $K = 10$. (c) Target location density $v^{t=10}$. (d) Search density $u^{t=10}$; available search effort $K = 100$. (e) Target location density $v^{t=100}$. (f) Search density $u^{t=100}$; available search effort $K = 1000$	48
Figure 2.14 Dynamics of Markovian target location density and search density for optimal allocations provided by Algorithm 2.3. (a) Target location density $v^{t=0}$. (b) Search density $u^{t=0}$; available search effort $K = 10$. (c) Target location density $v^{t=10}$. (d) Search density $u^{t=10}$; available search effort $K = 100$. (e) Target location density $v^{t=100}$. (f) Search density $u^{t=100}$; available search effort $K = 1000$	54
Figure 2.15 Dynamics of the group-testing search	56
Figure 2.16 Example of binary search tree for static target search	59
Figure 2.17 Example of binary search tree for a moving target search	62
Figure 2.18 Example of the Dorfman group-testing search tree	64
Figure 2.19 Example of the Sterrett group-testing search graph	65
Figure 2.20 Example of binary search tree for Hwang's algorithm	71
Figure 2.21 Example of the search tree for Sobel and Groll's algorithm	75
Figure 2.22 Example of the search tree for Graff and Roeloffs' algorithm	82
Figure 2.23 Example of 'division by half' search tree	85
Figure 2.24 Example of binary tree and maximal binary tree	87
Figure 2.25 Example of optimal binary search tree	93
Figure 2.26 Huffman coding tree corresponding to the optimal binary search tree	96
Figure 2.27 Entropy for two-point sample space	98
Figure 2.28 Example of the GOTA search tree	107
Figure 2.29 Example of the graph of states for the BF* algorithm	114
Figure 2.30 Example of the resulting path obtained by the BF* algorithm with a cumulative evaluation cost function	115
Figure 2.31 Example of the graph of states with estimated costs for the A* algorithm	119
Figure 2.32 Example of the expanded states and the resulting path obtained by the A* algorithm	120
Figure 2.33 Example of the expanded states and resulting paths obtained by two trials of the LRTA* algorithm. (a) Graph of the states with initial estimated costs $c_0(s)$ and costs $\bar{c}(s) = c^*(a^*[s, s_{next}])$. (b) Resulting path and updated costs after the first trial. (c) Resulting path and updated costs after the second trial	129

Figure 2.34 Example of the graph of states for the MTS algorithm	135
Figure 2.35 Example of the expanded states and the resulting paths of the searcher and the target for the MTS algorithm. (a) The movements of the searcher and the target, and the updated costs after Step 1. (b) The movements of the searcher and the target, and the updated costs after Step 2. (c) The movements of the searcher and the target, and the updated costs after Step 3	136
Figure 3.1 Example of the decision tree for MDP	148
Figure 3.2 General MDP for the search procedure	153
Figure 3.3 Example of the Markov process for the target's movements	154
Figure 3.4 Target's movements in the Pollock model of search	167
Figure 3.5 Optimal binary search tree for initial target location probabilities	178
Figure 3.6 Relocation of the search efforts in the branch-and-bound procedure	185
Figure 3.7 Dynamics of the searcher and target in the search-evasion game. (a) Initial searcher and target location probabilities, $t = 0$. (b) Searcher and target location probabilities, $t = 1$. (c) Searcher and target location probabilities, $t = 2$	199
Figure 3.8 Dynamics of pursuit-evasion game	205
Figure 3.9 Diameter and radius of the graph	207
Figure 3.10 Examples of complete, cyclic, and path graphs. (a) Complete graph \mathcal{K}_4 . (b) Cycle graph \mathcal{C}_4 . (c) Path graph \mathcal{P}_4	208
Figure 3.11 Dual definitions of the target's policies. (a) Moving target and unrestricted searcher's movements. (b) Static target and restricted searcher's movements	212
Figure 4.1 Dependence of the Rokhlin distances on location probabilities	227
Figure 4.2 Actual and estimated distances and the neighborhood of the partition in the ILRTA* algorithm	234
Figure 4.3 Correspondence between the Huffman–Zimmerman procedure and the ILRTA* algorithm. (a) Bottom-up Huffman search tree and corresponding partitions of the sample space. (b) The path created by the ILRTA* algorithm in the partitions space with Rokhlin distances between the partitions	240
Figure 4.4 Correspondence between the GOTA and the ILRTA* algorithm. (a) GOTA neighborhoods and weighted distances between the partitions of the sample space. (b) Bottom-up GOTA search tree and corresponding partitions of the sample space. (c) The path created by the ILRTA* algorithm acting in the GOTA conditions	245
Figure 4.5 Partitions space for the search by three and four searchers. (a) Space $\chi(2)$ for $m = 3$ searchers. (b) Space $\chi(2)$ for $m = 4$ searchers	250
Figure 4.6 Actual and estimated distances and the neighborhoods of the partitions in the IMTS algorithm	257
Figure 4.7 Markov process for the Pollock model of search with a fictive point	262
Figure 4.8 Example of the IMTS algorithm's actions for the Pollock model of search with a fictive point. (a) Partitions of the sample space and	

estimated distances. (b) Start with partition θ : Step 1. (c) Choice of fictive partition ξ_3 : Steps 2–4. (d) Choice of fictive partition ξ_1 : Steps 5–7. (e) Choice of fictive partition ξ_2 and stop: Steps 8–9	267
Figure 5.1 Example of the simulation frame during the search by Algorithm 5.2. (a) The states of the searcher and the target and the probabilities at time $t = 1$. (b) The states of the searcher and the target and the probabilities at time $t = 80$	306
Figure 5.2 Example of the simulation frame during the search by Algorithm 5.2 with three searchers. (a) The states of the searchers and the target and probabilities at time $t = 1$. (b) The states of the searchers and the target and probabilities at time $t = 50$	307
Figure 5.3 Location updating initiated by a mobile station. (a) Mobile station is in location area 1 and moves to location area 2. (b) Location updating when mobile station arrives at location area 2	311

Preface

Humans have been involved in search activities from the beginning of their history: searching for food and shelter in prehistoric times; searching for new continents and countries in the Middle Ages; and searching for information in the digital age.

It should not be surprising to see that in the new ‘information age’ some of the world’s leading companies are building their business on the basis of their search abilities. To name just two, Google has changed the world by providing search engines for users to look for specific content, and Facebook enables users to search for their own social identity. Both companies rely on dual search targets in their business models. On the one hand, they search and provide the exact information that their users are looking for. On the other hand, they search and provide lists of potential target users to their clients, such as media publishers and business organizations, according to predefined categories and queries.

It should be noted that all the above-mentioned search activities are probabilistic in nature. An ancient hunter-gatherer could only estimate where he should look for food. And although he could not formalize his thoughts in probabilistic terms and within a statistical framework, he captured information by his senses, processed it in his brain, and moved to areas where he believed that his chances of success would be higher.

Google and Facebook use an identical search scheme in a more formalized and modern manner. Information is gathered from the network and processed by sophisticated algorithms with a similar mission – to increase the probability of a successful search.

This book is about probabilistic search. We address this challenge and provide a more formalized framework for one of the most investigated questions in the history of science. A major part of this challenge is to consolidate the different research directions, schemes, notations, assumptions, and definitions that are related to probabilistic search in various research fields, such as operations research, artificial intelligence, stochastic search, information theory, statistics, and decision theory – to name but a few.

The book is intended for practitioners as well as theorists. It presents some of the main concepts and building blocks that are shared by many probabilistic search algorithms. It includes both well-known methods and our own methods for an information-driven search. It uses information theory concepts that are general enough to cope with the generalized search scheme. Moreover, it focuses on group testing, which is the theory that enables conclusions to be drawn on a group of search points simultaneously, for example, a conclusion made by the ancient hunter-gatherer to avoid a certain search area where the chances of success are lower.

We do not claim to provide a unified and general probabilistic search framework, yet we hope that this book bridges some of the gaps between the various research areas that tackle probabilistic search tasks.

In the book, Algorithm 5.1 was developed and programmed in collaboration with Oded Mery and Niv Shkolnik; Algorithm 5.2 was developed in collaboration with Boaz Golany and Gal Goren and was programmed in collaboration with Gal Goren; and the algorithms for mobile robot navigation discussed in Section 5.3 were developed and programmed in collaboration with Emanuel Salmona. Technical information regarding the cellular networks discussed in Section 5.4 was provided by Sergey Savitsky and Baruch Bar from Celcom. We would like to thank all these people for their help.

We are grateful to Havatzelet Tryster, Shmuel Gal, Yigal Gerchak, Gal Goren, Ron Kennet, Lev Levitin, and Shelemyahu Zaks, who read and commented on parts of the manuscript, and to Boaz Golany for fruitful discussions.

We would also like to express our sincere thanks to Heather Kay, Ilaria Meliconi, Richard Davies, and Prachi Sinha Sahay from John Wiley & Sons, Ltd, and Radhika Sivalingam from Laserwords, Ltd for their kind assistance and goodwill; without their help this book would not be published. Finally, we would like to thank our families for their continuous support and love. The book is dedicated to them and to information-seekers around the world who search for good answers.

This book includes an accompanying website. Please visit www.wiley.com/go/probabilistic-search

Notation and terms

$a^t : X \rightarrow X$ and $\overrightarrow{a}(t)$	a search agent trajectory. For the search in continuous time, it is a smooth function on $X \times [0, t]$, and for the search in discrete time the trajectory is defined as a sequence $\overrightarrow{a}(t) = \langle A^{t=0}, \dots, A^{t-2}, A^{t-1}, A^t \rangle$ of observed areas chosen up to time t .
$a[s_{init}, s_{fin}]$	a path in the graph \mathcal{G} from an initial state s_{init} to a final state s_{fin} ; $a[s_{init}, s_{fin}] = \langle (s_{init} = s^0, s^1), (s^1, s^2), \dots, (s^{t-1}, s^t = s_{fin}) \rangle$, where s^t are states at times $t = 0, 1, 2, \dots$. In the framework of path planning over graphs path $a[s_{init}, s_{fin}]$ has the same meaning as the above-mentioned trajectory $\overrightarrow{a}(t)$ in the framework of search and screening.
$a_i[x, s] \circ a_j[s, y]$	concatenation of the paths $a_i[x, s]$ and $a_j[s, y]$ in the graph \mathcal{G} , where states x, y , and s are such that the resulting path follows the path a_i starting from state x up to state s and then, starting from s , continues with the path a_j up to state y .
$a^t \in \mathcal{A}$	an action chosen or applied at time t .
\overrightarrow{a}	a sequence $\overrightarrow{a} = \langle a^t, a^{t-1}, \dots, a^1, a^0 \rangle$ of actions, which were conducted by the decision-maker at the corresponding times up to time t .
$\overrightarrow{a \circ s}$	a history of system dynamics, which includes both the states and the actions, $\overrightarrow{a \circ s} = \langle a^t, s^t, a^{t-1}, s^{t-1}, \dots, a^1, s^1, a^0, s^0 \rangle$.
$\overrightarrow{a^t \circ \widehat{s}^t}$	a detection or observation history, which is similar to the above history $\overrightarrow{a \circ s}$ of system dynamics, but, in contrast, is defined by using detected states $\widehat{s} \in \mathcal{S}$: $\overrightarrow{a^t \circ \widehat{s}^t} = \langle a^t, \widehat{s}^t, a^{t-1}, \widehat{s}^{t-1}, \dots, a^1, \widehat{s}^1, a^0, \widehat{s}^0 \rangle$.
$A \subset X$	an observed area that is a subset of the locations set X . The observed area is defined as the area that can be screened by a search agent in a time of unity and over which an observation result is obtained.
$\mathcal{A} = \{a_1, a_2, \dots\}$	a finite or countable set of actions. In the GOTA algorithm, actions $a \in \mathcal{A}$ specify partitioning of the set of states. In search

	games, the actions of the search and the target are distinguished, so:
$\mathcal{A}_{[\text{ser}]} = \{\alpha_{[\text{ser}]1}, \alpha_{[\text{ser}]2}, \dots\}$	is a set of actions which can be applied by the searcher; and
$\mathcal{A}_{[\text{tar}]} = \{\alpha_{[\text{tar}]1}, \alpha_{[\text{tar}]2}, \dots\}$	is a set of actions which are available to the target.
$\mathbb{A}_k, k = 1, 2, \dots$	actions which define partitioning in the GOTA algorithm.
$\alpha, \beta, \gamma, \dots$	partitions of the sample space X . By definition, for any partition $\alpha = \{A_1, A_2, \dots, A_m\}$ that consists of the areas $A_i \in X$ it follows that $A_i \cap A_j = \emptyset, i \neq j$, and $\cup_{i=1}^m A_i = X$. An initial partition θ and a final partition γ are distinguished.
\prec and $\alpha \prec \beta$	refinement relation between two partitions α and β : $\alpha \prec \beta$ if for every $B \in \beta$ there exists a set $A \in \alpha$ such that $B \subset A$.
\vee and $\alpha \vee \beta$	multiplication of two partitions α and β : partition $\alpha \vee \beta$ consists of all possible intersections $A \cap B$ of the sets $A \in \alpha$ and $B \in \beta$. The multiplication of m partitions $\alpha_j, j = 1, 2, \dots, m$, is denoted by $\vee_{j=1}^m \alpha_j$.
α_{-i}	a partition resulting from multiplication of m partitions excluding partition α_i , that is, $\alpha_{-i} = \vee_{j=1, j \neq i}^m \alpha_j$.
$c_\kappa : X \rightarrow [0, \infty)$	a function that determines the cost of applying the search effort $\kappa(x, t)$ to the point $x \in X$.
$c(s) = c(a[s_{\text{init}}, s])$	evaluated cost of the path $a[s_{\text{init}}, s]$ in the graph \mathcal{G} specified by real-valued evaluation function c . It is assumed that, in general, $c(s_{\text{fin}}) = c(a[s_{\text{init}}, s_{\text{fin}}]) \neq C(a[s_{\text{init}}, s_{\text{fin}}])$.
$c_{RT}(s)$	evaluated cost of the path $a[s_{\text{init}}, s]$ in the graph \mathcal{G} in the Real-Time (RT) search algorithms on graphs.
$c(s a)$	the cost of following a path a from the state s_{init} to the state $s \in a$, which is not necessarily a final state of the path a .
$\bar{c}(s), \tilde{c}(s),$	for certain implementations of the function c , $\bar{c}(s) = c(s a)$ is the evaluated cost of the path from the initial state s_{init} to state s as found up to the current step, and $\tilde{c}(s) = \tilde{c}(a^*[s, s_{\text{fin}}])$ is the estimated evaluated cost of the optimal path from state s to the final state s_{fin} .
$c^*(s), \bar{c}^*(s), \tilde{c}^*(s)$	for certain implementations of the function c , $c^*(s)$ denotes the actual evaluation cost, $\bar{c}^*(s) = c(s a^*)$ the actual evaluated cost of the optimal path a^* , and $\tilde{c}^*(s)$ the actual cost of the optimal path.
$c(\alpha, \beta)$	the cost of movement from partition α to partition β .
$C(a[s_{\text{init}}, s_{\text{fin}}])$	the cost of the path $a[s_{\text{init}}, s_{\text{fin}}]$. Cost $C(a[s_{\text{init}}, s_{\text{fin}}]) \in [0, \infty)$ is considered as the real cost of the resulting solution.
$\mathcal{C}[X]$	a code that is a set of codewords $\mathcal{w}(x)$ with respect to the (often binary) tree $\mathcal{T}[X]$.
$C(\mathcal{T}[X])$	the total cost of creating a (binary) search tree $\mathcal{T}[X]$. In general, it is defined as the sum of costs of creating the next level of the tree from its current level.
$C[w] \in [0, \infty)$	the cost of a search following the allocation w .

$c : \mathcal{X} \times [0, 1] \rightarrow \mathfrak{C}$	a control function that specifies the actions according to the observed area $A^t \in \mathcal{X}$ chosen at time t and the obtained observation result $z^t = z(A^t, x^t)$.
$c^t = c(A^t, z^t)$	a control at time t that is specified by the control function c . In the basic case $c^t \in \{\text{terminate}, \text{ continue}\}$.
$\mathfrak{C} = \{c_1, c_2, \dots\}$	a control set that includes control actions which specify dataflow control. In the simplest case, the control set includes two basic actions $\mathfrak{C} = \{\text{terminate}, \text{ continue}\}$.
$d(s, s')$	the distance between the vertexes of the graph $\mathcal{G} = (S, E)$ that is the length of the shortest path (i.e., a number of edges in this path), which starts at vertex s and ends at vertex s' .
$d(\alpha, \beta)$	the Rokhlin distance between partitions α and β . Similarly: $d_{Orn}(\alpha, \beta)$ is the Ornstein distance; and $d_{Ham}(\alpha, \beta)$ is the Hamming distance.
$d_p(\alpha, \beta)$	the distance between partitions α and β given probability mass function p over the range.
$\tilde{d}(\alpha, \beta)$	the estimated distance between partitions α and β .
$D(\mathcal{T}[X])$	the depth of the (binary) search tree $\mathcal{T}[X]$ that is defined as $D(\mathcal{T}[X]) = \max_{x \in X} \{l(x)\}$.
$\mathfrak{d}_i^t(\alpha_k)$	a decision rule that is the probability of choosing the action α_k while at time t the system is in the state s_i . In search games, decision rules are defined for the searcher and the target independently, that is, $\mathfrak{d}_{[\text{ser}]i}(\alpha_{[\text{ser}]k}) = \Pr\{\alpha_{[\text{ser}]} = \alpha_{[\text{ser}]k} s = s_i\}$, $\mathfrak{d}_{[\text{tar}]i}(\alpha_{[\text{tar}]k}) = \Pr\{\alpha_{[\text{tar}]} = \alpha_{[\text{tar}]k} s = s_i\}$.
$\overline{\mathfrak{d}}$	a policy $\overline{\mathfrak{d}} = \langle \mathfrak{d}^t, \mathfrak{d}^{t-1}, \dots, \mathfrak{d}^1, \mathfrak{d}^0 \rangle$, which is a sequence of decision rules up to time t . If for all rules it follows that $\mathfrak{d}^t(\alpha_k) = 1$, then the policy $\overline{\mathfrak{d}}$ unambiguously defines the strategy \overline{s} and these terms are used interchangeably.
$E(D \mathcal{T}[n])$	the expected depth of the (binary) search tree $\mathcal{T}[n]$ that is defined over a sample space of size n .
$E(z n, v)$	the expected number of observations that are required to find v targets in the sample space X of size n by using certain group-testing search procedures. In particular: $E_D(z n, m)$ corresponds to the Dorfman procedure; $E_S(z n, m)$ corresponds to the Sterrett procedure; and $E_H(z n, v)$ corresponds to the Hwang procedure.
$E_{SG}(z y, n)$	the expected number of observations over observed areas of size y that are required to find all targets in the sample space X of size n by using the Sobel and Groll algorithm.
$E_{GR}(C y, n, u, v, \ell)$	the expected cost of observations over observed areas of size y that are required to find all targets in the sample space X of size n by using the Graff and Roeloffs algorithm. Parameters ℓ , u , and v correspond to the state of search.
$E_{Pol}(z p_1)$	the expected number of observations required by the Pollock model to detect a target while the a priori target location probability at the point x_1 is p_1 .

$E_{\text{Ross}}(C \tilde{p})$	the minimal expected cost required to pay for detecting the target given the vector \tilde{p} of posterior estimated location probabilities in the Ross model.
$E_{\text{BB}}(\overrightarrow{a}_{j=1,\dots,m}(t))$	the expected number of rendezvous given possible searcher's trajectories $\overrightarrow{a}_{j=1,\dots,m}(t)$ up to time t in the branch-and-bound procedure.
$E_{\text{CSEG}}(\boldsymbol{u}, \boldsymbol{v} t)$	the expected number of rendezvous in the cumulative search-evasion game (CSEG) with respect to the pair $(\boldsymbol{u}, \boldsymbol{v})$ of probability mass functions such that $\boldsymbol{u}^\tau(a)$ is the probability that at time τ the searcher's location is a , and $\boldsymbol{v}^\tau(o)$ is the probability that at time τ the target's location is o . A maximal number $E_{\text{CSEG}}(\boldsymbol{u}, \boldsymbol{v} t)$ over all possible pairs $(\boldsymbol{u}, \boldsymbol{v})$ is the value of the game, denoted by $E_{\text{CSEG}}(\boldsymbol{u}^*, \boldsymbol{v}^* t)$, and the functions \boldsymbol{u}^* and \boldsymbol{v}^* are called optimal policies of the searcher and of the target, correspondingly. The value $E_{\text{CSEG}}(\boldsymbol{u}^*, \boldsymbol{v}^* t)$ is also denoted by $V_{\text{CSEG}}(\boldsymbol{u}^*, \boldsymbol{v}^* t)$.
$E_{\text{PPEG}}(R \boldsymbol{\delta}_{[\text{ser}]}, \boldsymbol{\delta}_{[\text{tar}]})$	an expected reward, which can be obtained by the searcher during the pursuit in the probabilistic pursuit-evasion game (PPEG) with respect to the pair $(\boldsymbol{\delta}_{[\text{ser}]}, \boldsymbol{\delta}_{[\text{tar}]})$ of the searcher's and target's policies.
$E_{\text{PPEG}}(C \boldsymbol{\delta}_{[\text{ser}]}, \boldsymbol{\delta}_{[\text{tar}]})$	an expected cost, which can be paid by the target while evading the searcher in PPEG with respect to the pair $(\boldsymbol{\delta}_{[\text{ser}]}, \boldsymbol{\delta}_{[\text{tar}]})$ of the searcher's and target's policies.
$E_{\text{PPEG}}(\boldsymbol{\delta}_{[\text{ser}]}^*, \boldsymbol{\delta}_{[\text{tar}]}^*)$	a Nash equilibrium of PPEG. A pair $(\boldsymbol{\delta}_{[\text{ser}]}^*, \boldsymbol{\delta}_{[\text{tar}]}^*)$ of the searcher's and target's policies is called the solution of the game.
$E(R \overrightarrow{\boldsymbol{\delta}})$	an expected reward that is the reward expected to be obtained by the decision-maker while a policy $\overrightarrow{\boldsymbol{\delta}}$ is chosen. A maximal expected reward over all possible policies is denoted by $E(R \overrightarrow{\boldsymbol{\delta}}^*)$, where $\overrightarrow{\boldsymbol{\delta}}^*$ is the optimal policy.
$E_{\text{POMDP}}^{\text{inf}}(R \overrightarrow{\boldsymbol{\delta}}(\mathbb{I}^\infty))$	an expected discounted infinite horizon reward of POMDP, in which actions are chosen according to the policy $\overrightarrow{\boldsymbol{\delta}}$. A maximal expected discounted infinite horizon reward over all possible policies is denoted by $E_{\text{POMDP}}^{\text{inf}}(R \overrightarrow{\boldsymbol{\delta}}^*(\mathbb{I}^\infty))$, where $\overrightarrow{\boldsymbol{\delta}}^*$ is the optimal policy. This value is also denoted by $V(\mathbb{I}^\infty)$.
$\mathcal{G} = (S, E)$	a graph with a set $S = \{s_1, s_2, \dots, s_n\}$ of vertexes (or nodes), which represent states of the search process, and a set $E \subset S \times S$ of edges weighted with real numbers or costs.
$diam(\mathcal{G})$	diameter of the graph \mathcal{G} that is the length of the longest path in the graph.
$rad(\mathcal{G})$	radius of the graph \mathcal{G} that is a minimum of the longest paths taken over the vertices of \mathcal{G} .
$H(X)$	the entropy (Shannon entropy) of the sample space X that specifies a lower bound on the expected code length.
$H(x y)$	the conditional entropy of random variable x given random variable y .
$H(\alpha)$	the entropy of partition $\alpha = \{A_1, A_2, \dots, A_m\}$.

$H(\beta \alpha)$	the conditional entropy of partition $\beta = \{B B \subset X\}$ given the partition $\alpha = \{A A \subset X\}$
$I(x; y)$	mutual information for random variables x and y .
$I(\alpha; \beta)$	mutual information for two partitions α and β .
$I(\beta, \alpha \gamma)$	the conditional information gain between partitions β and α with respect to partition γ .
$\mathbb{I}_i(\overrightarrow{\alpha^{t-1} \circ \widehat{s}^{t-1}})$	the probability that at time t the core Markov process is in state $s_i \in \mathcal{S}$ given the detection history $\overrightarrow{\alpha^{t-1} \circ \widehat{s}^{t-1}}$ up to the previous time $t-1$.
\mathbb{I}^t	an information vector $\mathbb{I}^t = (\mathbb{I}^t(s_1), \mathbb{I}^t(s_2), \dots, \mathbb{I}^t(s_n))$, which summarizes all available information regarding choice of the action α^t at time t .
$\kappa^t : X \rightarrow [0, \infty)$	a search effort function that defines the amount of search effort that is applied to the points of the sample space X up to time t .
\mathbf{k}^t	an $n \times t$ matrix $\mathbf{k}^t = \ k(x_i, \tau)\ _{n \times t}$ of the non-weighted efforts $k(x_i, \tau)$, $x_i \in X$, $i = 1, \dots, n$, and $\tau = 0, 1, 2, \dots, t$. The rows of the matrix \mathbf{k}^t are denoted by $\mathbf{k} = \ k'(x_i)\ _{n \times 1}$ and include the non-weighted efforts specified for the points $x_i \in X$.
$K(t) \in [0, \infty)$	available search effort at time t that is divided over the points of the set X according to the search density u^t .
$l(x)$	the number of edges from the root node of the search tree $\mathcal{T}[X]$ to the leaf, which is associated with a single-point area $\{x\}$, $x \in X$. In terms of coding, $l(x)$ is the length of the codeword $w(x)$.
$L = L[\mathcal{C}]$	the average number of the total number of edges from the root node of the search tree $\mathcal{T}[X]$ to the leaf. In terms of coding, $L[\mathcal{C}]$ represents the expected length of the code \mathcal{C} .
$\ell[x, \lambda, \kappa^t(x)]$	a pointwise Lagrange function (or pointwise Lagrangian) $\ell[x, \lambda, \kappa^t(x)] = p^t(x)\varphi(x, \kappa^t(x)) - \lambda c(x, \kappa^t(x))$. With the use of pointwise Lagrangian ℓ , the Lagrangian for an allocation w is the sum $\mathcal{L}[w] = \sum_{x \in X} \ell[x, \lambda, \kappa^t(x)]$.
$\mathcal{L}[w]$	a Lagrange function (or Lagrangian) $\mathcal{L}[w] = P[w] - \lambda C[w]$, where the real number λ is a Lagrange multiplier.
$N(s)$	a set of successors (may be marked) of state s in the graph \mathcal{G} . In the specific algorithms, set $N(s)$ is also referred to as the neighborhood of the state or point s .
$N(\alpha)$	a neighborhood of the partition α , $\alpha \notin N(\alpha)$.
$N_{\text{Huf}}(\alpha)$	a Huffman neighborhood of the partition α , $\alpha \notin N_{\text{Huf}}(\alpha)$.
$N_{\text{GOTA}}(\alpha)$	a GOTA neighborhood of the partition α , $\alpha \notin N_{\text{GOTA}}(\alpha)$.
$N(\alpha, r)$	a neighborhood of the partition α of radius $r > 0$, $\alpha \notin N(\alpha, r)$.
$\overrightarrow{o} : X \rightarrow X$ and $\overrightarrow{o}(t)$	a target trajectory. For the search in continuous time, it is a smooth function on $X \times [0, t]$, and for the search in discrete time the trajectory is defined as a sequence $\overrightarrow{o}(t) = \langle x^{t=0}, \dots, x^{t-2}, x^{t-1}, x^t \rangle$ of points chosen up to time t .
$p^t : X \rightarrow [0, 1]$	a target location probability function that is defined over the locations set X for time t .

$p^t : \mathcal{X} \rightarrow [0, 1]$	an observed location probability function that defines the target location probability $p^t(A)$ in the observed area $A \subset X$.
$\bar{p}^t : X \rightarrow [0, 1]$	an observed location probability function that defines location probabilities $\bar{p}^t(x)$, $x \in X$, given a result of observation of the observed area $A \subset X$.
$\tilde{p}^t : X \rightarrow [0, 1]$	the estimated location probability function that defines the location probabilities $\tilde{p}^t(x)$, $x \in X$, given the observed probabilities $\bar{p}^{t-1}(x)$ at previous time $t - 1$ and the definite or estimated rule of the target's movement.
$P(t)$	the probability of detecting the target up to time t . If $0 \leq \Pr\{\text{target is detected at } x \text{target is located at } x\} \leq 1$ and $\Pr\{\text{target is detected at } x \text{target is not located at } x\} = 0$, then it follows that $P(t) = 1 - \prod_{\tau=0}^{t-1} \sum_{i=1}^n (1 - \psi(x_i, \tau)) \times v(x_i, \tau)$.
$P[w] \in [0, 1]$	the probability of detecting the target by implementing allocation w .
$P(p_1 t)$	a maximal probability of detecting the target at time t in the Pollock model while the a priori target location probability at the point x_1 is p_1 .
$P(\vec{a}_{j=1,\dots,m}(t))$	the probability of detecting the target given possible searcher's trajectories $\vec{a}_{j=1,\dots,m}(t)$ up to time t .
$P_{\text{Eagle}}(\tilde{p} t)$	maximum probability of detecting the target up to time t given the vector \tilde{p} of posterior estimated location probabilities in the Eagle model.
$p_{[\text{ser}]}(s)$ and $p_{[\text{tar}]}(s)$	the probability $p_{[\text{ser}]}(s)$ that the searcher checks the vertex $s \in S$ in the graph $\mathcal{G} = (S, E)$, and the probability $p_{[\text{tar}]}(s)$ that the target is located at the vertex s .
$\pi_{ik}(\vec{\delta})$	a steady state probability for the state s_i and action a_k given policy $\vec{\delta} : \pi_{ik}(\vec{\delta}) = \lim_{t \rightarrow \infty} \Pr\{s^t = s_i, a^t = a_k \vec{\delta}\}$.
$Q(t)$	the probability of not detecting the target up to time t . Under the previous requirement regarding $P(t)$, it follows that $Q(t) = 1 - P(t)$.
$Q(k^t)$	the probability of not detecting the target up to the time moment t by using the efforts $k(x_i, \tau)$ determined by matrix k^t .
$Q[\vec{o}(t)]$	the probability of not detecting the target up to time t subject to the target's trajectory $\vec{o}(t)$.
$Q(\vec{a}_{j=1,\dots,m}(t))$	the probability of not detecting the target given possible searcher's trajectories $\vec{a}_{j=1,\dots,m}(t)$ up to time t : $Q(\vec{a}_{j=1,\dots,m}(t)) = 1 - P(\vec{a}_{j=1,\dots,m}(t)).$
$R(s, a) \in [0, \infty)$	the reward that is obtained by the decision-maker when, after observation of the state $s \in \mathcal{S}$, the action $a \in \mathcal{A}$ is chosen.
$\rho = \ \rho_{ij}\ _{n \times n}$	a transition probability matrix that defines the target's movement according to a discrete time Markov process. The transition probabilities are defined as $\rho_{ij} = \Pr\{x^{t+1} = x_j x^t = x_i\}$ and for the transition matrix ρ it follows that $\sum_{j=1}^n \rho_{ij} = 1$ for all $i = 1, 2, \dots, n$.

$\rho_{ij}(\alpha_k)$	the probability of transition from the state s_i to the state s_j given that the action α_k was applied. For the Markovian system, $\rho_{ij}(\alpha_k) = \Pr\{s^{t+1} = s_j \alpha^t = \alpha_k, s^t = s_i\}$. In the search games, transitions between the states s depend on the actions $\alpha_{[ser]} \in \mathcal{A}_{[ser]}$ of the searcher and the actions $\alpha_{[tar]} \in \mathcal{A}_{[tar]}$ of the target. So
$\rho_{ij}(\overrightarrow{\delta})$	$\rho_{ij}(\alpha_{[ser]k}, \alpha_{[tar]l}) = \Pr\{s^{t+1} = s_j \alpha_{[ser]k}^t = \alpha_k, \alpha_{[tar]l}^t = \alpha_l, s^t = s_i\}$. transition probabilities $\rho_{ij}(\alpha_k)$ with respect to the policy $\overrightarrow{\delta}$: $\rho_{ij}(\overrightarrow{\delta}) = \Pr\{s^{t+1} = s_j s^t = s_i, \overrightarrow{\delta}\}$.
$s \in S$	the state of a searcher or target in the search over graph $\mathcal{G} = (S, E)$ that is a vertex $s \in S$ at which the search or target is located.
$S = \{s_1, s_2, \dots, s_n\}$	a finite set of states in the search over graph \mathcal{G} that is a set of vertexes of the graph \mathcal{G} .
$s^t \in \mathcal{S}$	an abstract state at time t .
$\tilde{s}^t \in \mathcal{S}$	an observed or detected abstract state at time t ; in general $\tilde{s}^t \neq s^t$, where $s^t \in \mathcal{S}$ is a real state of the core process.
\overrightarrow{s}	a strategy that is a sequence $\overrightarrow{s} = \langle s^t, s^{t-1}, \dots, s^1, s^0 \rangle$ of states, which were observed by the decision-maker up to time t .
$\mathcal{S} = \{s_1, s_2, \dots\}$	a finite or countable set of some abstract states. For algorithms on graphs, set \mathcal{S} corresponds to the set S of vertices of the graph \mathcal{G} .
$t = 0, 1, 2, \dots$	times. In the continuous time search, the time period is considered as an interval $[0, T)$ that may be infinite and times $t \in [0, T)$.
$T[X] = (S, E)$	a (binary) search tree, where S is a set of vertices (or nodes) and E is a set of edges, which corresponds to the group-testing algorithm of search over a sample space X .
$\mathbb{T} : \mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}$	a function that specifies the division rule that specifies, for any area $A \in \mathcal{X}$, a pair of areas A_1 and A_2 such that $A_1 \cap A_2 = \emptyset$ and $A_1 \cup A_2 = A$, while one of them can be empty.
$u^t : X \rightarrow [0, \infty)$	a search density function that specifies the distribution of search efforts over the sample space X , either for a single search agent or for a number of search agents.
$v^t : X \rightarrow [0, 1]$	a target location density function that for a single target is equivalent to the estimated location probability function \tilde{p}^t and for multiple targets results from the estimated probabilities of all targets.
$V_{\text{CSEG}}(u^*, v^* t)$	see $E_{\text{CSEG}}(u, v t)$.
$V(\mathbb{I}^\infty)$	see $E_{\text{POMDP}}^{\text{inf}}(R \overrightarrow{\delta}(\mathbb{I}^\infty))$.
$V_{\text{PEGG}}(\overrightarrow{p}^*_{[\text{ser}]}, \overrightarrow{p}^*_{[\text{tar}]})$	a value of the pursuit-evasion game on graph (PEGG), where $\overrightarrow{p}^*_{[\text{ser}]}$ and $\overrightarrow{p}^*_{[\text{tar}]}$ are the optimal probabilistic policies of the searcher and target. This value is also denoted by $V_{\text{PEGG}}(\mathcal{G})$, where \mathcal{G} is a graph on which the game is played.
$V_{\text{PEGG}}(\mathcal{G})$	see $V_{\text{PEGG}}(\overrightarrow{p}^*_{[\text{ser}]}, \overrightarrow{p}^*_{[\text{tar}]})$.

$w : X \rightarrow [0, \infty)$	an allocation function that determines the distribution of search efforts over the sample space X at fixed time t , while for a given time t it is equivalent to the search density u^t . The allocations w that correspond to the search densities over X for all times t form a set W of allocations.
$w(s, s') \in [0, \infty)$	a weight or cost of the edge $(s, s') \in E$ in the graph \mathcal{G} .
$\omega(x)$	a codeword that is a sequence of symbols, which determines a path from the root to the leaf. For a binary tree, the symbols are defined as ones and zeros, and the edges that connect the node with its left successor are associated with, for example, ones, while the edges that connect the node with its right successor are associated with zeros.
$x^t \in X$	target location at time t .
$X = \{x_1, x_2, \dots, x_n\}$	a set of target locations that are considered as points or cells. If set X represents a gridded domain of a Euclidean plane, then each point $x_i \in X$, $i = 1, \dots, n$, is considered as a pair $x_i = (x_{1i}, x_{2i})$ of Cartesian coordinates of the point. In the continuous models, set X is defined as a Cartesian multiplication of two normalized one-dimensional segments, that is, $X = [0, 1] \times [0, 1]$.
$\mathcal{X} = 2^X$	a power set of X that includes all possible subsets of X . Since any observed area A is a subset of X , it is an element of the power set \mathcal{X} , that is, $A \in \mathcal{X}$.
$\chi = \{\alpha_0, \alpha_1, \alpha_2, \dots\}$	a set of all possible partitions of the set X .
$\chi(k)$	a set of partitions such that the size of every partition $\xi \in \chi(k)$ is less than or equivalent to k .
$\varphi_\kappa : X \rightarrow [0, 1]$	a detection function that maps a search effort $\kappa(x, t)$, which is applied to a point $x \in X$ at time t , to the probability of the target's detection at this point x .
$\psi^t : X \rightarrow [0, 1]$	a probability function that specifies the probability of the target's detection at time t . Under an assumption that $0 \leq \Pr\{\text{target is detected at } x \text{target is located at } x\} \leq 1$ and $\Pr\{\text{target is detected at } x \text{target is not located at } x\} = 0$, it follows that $\psi^t(x) = \varphi_\kappa(x) \equiv \varphi(x, \kappa)$.
ψ_{ij}	a probability that the detected state \hat{s} is equivalent to the real state s_i : $\psi_{ij} = \Pr\{\hat{s} = s_j s = s_i\}$ (cf., the above probability $\psi^t(x)$ of the target's detection).
$\psi_{ij}(\alpha_k)$	a probability that the detected process \hat{s} is equivalent to the real state s_i given that the action α_k is applied:
$z : \mathcal{X} \times X \rightarrow [0, 1]$	$\psi_{ij}(\alpha_k) = \Pr\{\hat{s}' = s_j \alpha^t = \alpha_k, s^t = s_i\}$.
$z^t = z(A^t, x^t)$	an observation function that specifies the result of observation of the observed area $A \in \mathcal{X}$ while the target is located at $x \in X$. an observation result at time t . For an errorless detection it is assumed that $z^t = \mathbf{1}(A^t, x^t)$, where $\mathbf{1}$ is the indicator function such that $\mathbf{1}(A^t, x^t) = 1$ if $x^t \in A^t$ and $\mathbf{1}(A^t, x^t) = 0$ otherwise. For an erroneous detection, it is assumed that the observation result does not depend on the target location and is defined by a function ψ^t as $z^t = \psi^t(A^t) = \sum_{x \in A^t} \psi^t(x)$.

1

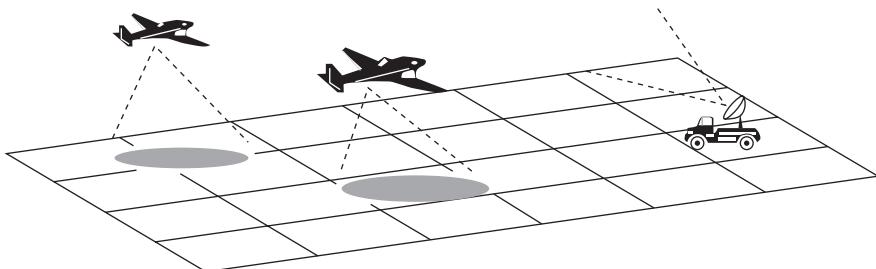
Introduction

The terms ‘search’ and ‘search problem’ arise in a number of problems that appear in different fields of applied mathematics and engineering. In its literal meaning, the notion of search problem corresponds to the class of situations in which an agent (a searcher) is looking for a hidden object (either one of them can be physical or abstract) by screening a certain defined area. The search problem is formulated under various restrictions on the considered search implementation as well as on the agent and the target functionalities.

To illustrate the potential complexity that might be considered in a search problem let us start with a simple search example and gradually add to it various assumptions and conditions. The figure below presents a simple schematic view of a search problem where the target (a vehicle) is located at some point in a given space, while the searcher or searchers (aircraft) are looking for it. An initial classification of the problem depends on the definition of the *sample space*, which can be either *discrete* or *continuous*. In this book we mainly consider the former case, which implies that the target and the searcher move to well-defined points in the space. These discrete positions can also be modeled by nodes in a graph. This type of presentation is popular in the artificial intelligence (AI) literature where cost parameters or weights are often added to the edges of the graph, such that the overall cost of the search is obtained by accumulating these costs along the search path over the edges. When the weights are distributed unevenly, the search procedure can account for different considerations, such as non-homogeneous search distances or search efforts. We consider some of these cases. A second critical feature of the problem is related to the ability of the target to *move* in the space. In the case of a moving target, several versions exist for representing its type and path. Some of the most popular schemes are random moves, Markovian moves, and Brownian moves. We address these assumptions in the relevant sections. In general, optimal solutions exist for static search problems but they often do not exist for dynamic search problems with a moving target. In this book we consider both approaches, and in particular we propose a general search scheme that applies to both cases. A third feature of the search is related to the information available to the searcher. If the location of the target is known, then a *complete-information* search

problem can be mapped to a relatively simpler path-planning or chase-planning problem. These types of problems often appear in the operations research literature.

The origin of these deterministic search problems for a moving target was the *pursuit problem* that was formulated in the eighteenth century. This class of problem is computationally tractable and often focuses on capturing the target with a minimal number of search moves. In this book we focus almost entirely on the *incomplete-information search*, where the exact location of the target is generally unknown to the searcher. Note that there are several methodological concepts for addressing the incomplete-information search, for example, by relying on rough-set theory, fuzzy logic, or on probability theory. We follow the latter *probabilistic search* approach, modeling the incomplete information on the target location by a function that quantifies the probability of the target to be located at any point in the space (we call it the *location probability*). The search then becomes a probabilistic search and in many cases an adaptive one, where the results of the search up to a certain time are used to update the location probability distribution over the space, often by using a Bayesian statistics approach as we do here. A popular extension of the pursuit problem with incomplete information is known as the *search and screening problem*, formulated during World War II by Bernard Koopman. The problem is probabilistic not only with respect to the location of the target, but also with respect to the distribution of the *search efforts* that are applied in a continuous manner by the searcher to the search space.



We follow this approach, although we do not use the notion of distributed efforts. Instead, we assume that the search can be applied to discrete points of the search space. An important extension of the distributed search efforts, under a consideration of a discrete search space, is the *group-testing* search. In group testing the searcher can look for the target in a subspace of the search space and obtain an indication of whether the target is located somewhere in this subspace. The size of the allowed subspace is treated as an input parameter, while the search terminates if the subspace contains only a single point, thus representing complete information on the target location. In this book, we explicitly consider methods of group testing. If the target is static the search can be modeled by a coding theory process (where a code represents the location of the target in the space), while the coding procedures can be easily mapped to obtain the optimal search policy. These cases are often represented by decision trees that have become extremely popular in data-mining applications. In dynamic search, when the target is moving, such isomorphism between coding theory and search is no longer valid, so we propose a methodology that can be extended also to these cases.

There are several variants of the incomplete-information search. We often assume that the target is unaware of the search agent. If this is not the case, then the search process becomes a *search game* relying on some game theory concepts. We will shortly address these search games. Another conventional and realistic assumption is that the searcher's observations are prone to some *observation errors*. In these cases, two types of statistical errors have to be taken into account – either missing the target even though the searcher has searched the right point (a false negative error), or falsely indicating that the target has been found at a certain point (a false positive error). Of these two errors, the false negative one is much more popular, and we consider a few such cases in later chapters. Another version of the incomplete-information search, which is also considered in this book, addresses the situation of *side information*, where the searcher obtains some (incomplete) indication during the search of where the target is located. Another natural extension to all of the above methods is obtained when assuming that there is more than one target or one searcher in the search space. In such a case the question regarding the amount of cooperation among the targets or among search agents arises. A simple example of such cooperation is information sharing between the searchers in order to better estimate the location probability distribution and better utilize the joint search efforts. These extensions are discussed in the book as well.

We must stress the fact that the general formulation of the search problem as presented in this book does not distinguish between search for existing physical objects, such as cell (mobile) phones, people, and devices, and search for abstract entities, such as records in a database, an e-commerce customer on the Internet, a targeted customer type, or a search for feasible solutions of a given problem within a predefined solution space. Some popular tools for such search procedures can be found in the data-mining and statistics literature. We draw clear lines of similarities between search procedures for physical entities and those found in *problem-solving* procedures, typically in *stochastic local search* methods that are used to obtain feasible solutions to a given schematic problem.

In any case, even from the above simple example it can be understood that the search problem in its general form can be very complex, highly variant, and call for different solution schemes, which are partially covered in this book.

In summary, it is important to note that despite the similar properties of the above-mentioned variants of the search problem, there is no formal and unified search theory that captures all these points. Instead, one can find different search procedures and considerations in various research areas, such as operations research, coding theory, information theory, graph theory, computer science, data mining, machine learning, statistics, and AI. We do not claim to present such a unified search theory in this book, but we do try to bridge some of these gaps by formalizing the main properties, procedures, and assumptions that are related to many of these search problems and their variants.

In this chapter we start by discussing the motivation and applications of the search problem. This is done through a brief historical note on the origin of the search problem and its development. Then, we provide a general description of the search problem, focusing on three research fields in which it appears, namely, *search and screening*, *group testing*, and *stochastic local search*. We specifically discuss the core features of the stochastic local search, which provides the main insights on the proposed information-based search approach, and attempt to bridge some of the gap between different search theories. We end the chapter by discussing the structure of the book.

1.1 Motivation and applications

The problem of search is one of the oldest mathematical problems which were originated by the first applications of calculus in the seventeenth century. The problem arose by considering an agent which is required to determine the location of a hidden object and to intercept it. In the second half of the seventeenth century, the problem was explicitly proposed by Claude Perrault to Leibniz and studied in cooperation with Huygens. Perrault himself associated this problem with earlier studies conducted by Fermat and Newton [1].

The first essentially mathematical formulation and analysis of the search problem, according to Nahin [1], relates to the paper of Pierre Bouguer, which was submitted to the French Academy on 16 January 1732. This paper considered the pursuit of a merchant ship by a pirate ship. This is the reason why nowadays such a problem is often called the *pure pursuit problem*. In it the pursuing agent has complete information about the current location of the pursued object, and the task is to find the agent's path which leads to a guaranteed interception of the object in minimal time. In Bouguer's original pursuit problem, it was assumed that the merchant's movement was directed in a straight line. In 1748 a required generalization of the problem in which the pursued object moved in circles was formulated and published in the English *Ladies' Diary Journal* by John Ash in the form of a puzzle.

Formal mathematical considerations of the pursuit problem for various paths of the pursued object and different characteristics of the pursuer were proposed in a review published by John Runkle (1859) in *Mathematical Monthly*. Following this work, the pursuit problem was considered as an application of the theory of differential equations, and after publication of the classical monographs by Isaacs [2] and Pontryagin *et al.* [3], it was often considered as a variant of the optimal control problem.

Nowadays, the terms 'search' and 'search problem' arise in a number of problems that appear in different areas of applied mathematics and engineering. Some of the problems deal with essentially discrete models that can be described by walks over graphs, while other problems address continuous models that require the application of differential equations and certain variation principles. The problem of search in its literal/physical meaning appears in military applications (see, e.g., [4–6]) as the search for an object hidden by the enemy, in geological tasks as a search for minerals, as well as in other fields such as in quality control as the search for faulty units in a production batch [7, 8]. In such tasks, the searcher's detection abilities are restricted by sensor quality, and the order of search actions is often restricted by the physical abilities of the searcher.

Other variants of the search problem are obtained in search for intangible (abstract) items, such as in medical procedures when searching for the right diagnosis using decision trees, in cellular network paging in which the network controller looks for a mobile device address to redirect incoming calls [9], or in search for specific records in a file or database. In such tasks, the search can be modeled by relying on fault detection and group-testing principles [10, 11], by which the searcher is able to test parts of the system in a different order. Note that similar search properties appear in the tasks of data mining and classification, where the target of the search is defined as a certain property or data record.

Contrary to these indicated fields, in the tasks of optimization and operations research the search is considered as a method of problem solving, where the target is defined as a required solution that depends on a certain problem. In particular, the method of local

search [12, 13] for the solutions of *NP*-hard problems is based on sequential consideration of the solutions that are obtained at each step of the search while choosing the next solution from a close neighborhood of the current solution. Moreover, during recent decades search has obtained a similar meaning in the studies of AI [14, 15], where, for example, the A* family of algorithms is being intensively considered and used to solve related problems.

For each of these approaches to the search problem, certain algorithms and results were obtained over the years. Nevertheless, in spite of the clear association and analogies in such problems and methods, the connections between *heuristic search* methods (including the AI version), *search and screening* methods, and *group-testing* methods have not been fully established to date. The present work attempts to close the gap between these approaches to the search problem.

1.2 General description of the search problem

Search problems appear in different applications of invariant methods, formulations, and considerations. In the most general formulation, the problem includes a *searcher* that moves over a search space and looks for a *target* that is located somewhere in the search space. In this context, the searcher and the target can be considered either as physical agents so that the goal of the searcher is to find the target in minimal time (or minimal search effort), or as an abstract object, such as searching for the right medical diagnosis, where the searcher is considered as a problem-solver while the target is considered as a required solution of the problem.

The initial formulation of the problem dealt with the search for a hidden object by a physical agent acting in *continuous space* and continuous or discrete time. In such a formulation, the problem was studied within the framework of stochastic optimization and control theory [16–18], and for most cases of search for a static target, exact solutions of the problem in terms of shorter search path or minimal search time were obtained. When considering search for a moving target, the problem was often addressed by using different approaches, including optimization methods [5, 6, 19] and game theory [20–22].

In contrast to search by physical agents, the abstract formulation of the search problem often considers the actions of the problem-solver over a *discrete space* of candidate solutions, where at each search step a single solution is obtained. In such a formulation, the problem appears both in the models of search over a gridded plane and in the tasks of AI [15, 23] and optimization theory [12], including group-testing methods for the processes of quality control and inspection [7, 8]. Moreover, in many studies [10], a group-testing formulation of the search problem has been considered, while in other applications the problem has been referred to as a problem of screening and detection [6].

A general formulation of the search problem in the group-testing form allows the implementation of a wide set of assumptions regarding the searcher and the target that relate to different tasks. The main assumption corresponds to the *searcher's knowledge* regarding the target, which determines the nature of the search space. In most cases, it is assumed that the search space consists of abstract states that represent either target locations or candidate solutions, and that the probability distribution over the search space is defined. If the probabilities of the states are not available to the searcher, then it can be assumed that this distribution is uniform or that the states are equally probable. This assumption stems naturally from the well-known maximum entropy principle [24].

Other assumptions that correspond to the searcher's knowledge deal with detection and decision making. The first assumption is addressed by a *detection function* that defines the probability of errors of the first and second types. The Type I error corresponds to an erroneous declaration that the target has been found at a certain searched point. The Type II error corresponds to an erroneous declaration that the target is not located at a certain searched point. In most applications, it is assumed that the search is conservative – thus the search is prone to errors when checking a state where the target is located, while the detection of the absence of the target is errorless. Detection results, which can be represented either by Boolean ‘true’ or ‘false’ values or by detection probabilities, are used by the searcher to make decisions about the next state of the target and the next search action. In search by a physical agent, the detection function often corresponds to the properties of the available sensors, for example, when considering a car equipped with radar and cameras to avoid collisions with objects.

In the abstract version of the search problem, the detection function is often defined on the basis of the search space and its metric or topological properties. For example, when considering a problem-solving approach, a set of constraints can define the feasible search space.

Decision making and selection of the next search action usually depend on the detection result and on the information that is available to the searcher. In most optimization and group-testing algorithms, decision making is based on the probabilities of the states and on the search cost of the candidate decisions. The probabilities and costs are updated by the searcher during detection and support the decision making in further stages of the search [25]. In some AI methods [15], in contrast, the search is not defined by probabilistic terms – thus location probabilities are not modeled and decision making is mainly based on the distances (costs) between the states. In the popular A* algorithm of problem solving [14], the searcher uses side information about the required solution and applies an estimated distance (cost) to it, while choosing the next solution from the neighboring candidates.

Depending on the nature of the search problem, two main types of solution neighborhoods are defined. In the first type, the neighborhood includes all feasible solutions that are known at the current search step, and the choice of candidate solution is conducted over this set of feasible solutions. In such a case, some solution of the search problem can be obtained and there exist a number of algorithms, for example, Huffman search [26] or dynamic programming search [27, 28], that obtain the required solution. Nevertheless, these algorithms are mainly applicable to the search for a static target that corresponds to the search for a solution under the assumption that the uncertainty regarding the solution depends on the searcher's actions and not on external factors. In the second solution type, the neighborhood includes a restricted subset of a search space so that the selection procedure requires a small number of calculations and comparisons, and a complete solution is obtained by sequential decision making and acceptance of a number of intermediate solutions that converge to the required solution. Such a search method is referred to as a *local search* and can be implemented in the search both for static and for moving targets. In terms of problem solving, the search for a moving target means that the uncertainty regarding a required solution changes over time and that the solution depends on some external factors beyond the control of the searcher. With some additional assumptions, such search procedures can be associated with a process of sequential decision making in game theory [29, 30] and, in particular, in search games [20–22], where the physical target agent can

move over the space according to its internal rule, while the goal of the search agent is to find the target in minimal time or with minimal effort.

Finally, the formulation and methods of consideration of the search problem depend on the number of search agents and on the number of targets. The abstract search problems usually deal with a single decision-maker looking for a single solution or a group of solutions that represent this solution. In contrast, the search problems that consider the actions of physical agents depend on the number of search agents and the number of target agents and their movement. In such problems, additional questions arise regarding collective decision making, shared (or common) information, as well as communication between the searchers. Similar questions are considered regarding the target agents and their responses to the searchers' actions.

Based on the type of search problem and implemented assumptions, the problem can be addressed and solved by using different methods. Some of these methods are applicable to the problem of search in which physical agents act in a physical space; other methods are used in an abstract search for a solution of optimization or AI problems. In the next section, we consider some of the main approaches to problem-solving search that provide a context for the local search methods considered in the book.

1.3 Solution approaches in the literature

Methods for solving the search problem depend on its formulation and on the implemented assumptions and applied restrictions. For example, in the original pursuit problem or in its game theory formulation, called the *pursuit-evasion problem*, the pursuer has *complete information* on the position of the pursued object at any time. Accordingly, the goal is to determine the path of the pursuer given the velocity of the pursuer, the velocity and trajectory of the pursued object, and the governing rules of its movement. In such a formulation, the solution can be obtained by using suitable variation principles and, in most cases, it has a well-defined structure.

A variant of the pursuit problem with *incomplete information* – the so-called ‘dog in a fog’ problem – was initiated in 1942 by the US Navy’s Antisubmarine Warfare Operations Research Group. In a focused report [4] on the problem and later in a book [19], Koopman named this problem the *search and screening problem*. As indicated in a later report by Frost and Stone [31], the considered theory

is the study of how to most effectively employ limited resources when trying to find an object whose location is not precisely known. The goal is to deploy search assets to maximize the probability of locating the search object with the resources available. Sometimes this goal is stated in terms of minimizing the time to find the search object.

The objective in the search and screening problem, as stated in [31], is ‘to find methods, procedures, and algorithms that describe how to achieve these goals,’ where the pursuer is called the *searcher* and the pursued object is the *target*. It is assumed that the searcher acts under uncertainty, and that the available information on the target location changes during the search. The dependence between the amount of available information and the locations of the searcher and the target is defined by a *detection function*. This function defines the probability of detecting the target corresponding to the locations of the searcher and the

target. In the original pursuit problem, the function is reduced to a value that unambiguously determines the location of the target.

The effects of the searcher's path on the amount of information collected on the target gives rise to the following optimization problem. On the one hand, the goal of the searcher is to intercept the target with minimal resources or in minimal time. On the other hand, the searcher's knowledge about the target location often increases as the search continues. The optimal path thus balances these two terms, collecting information while finding the target in a minimum number of search steps.

An optimal solution of the search and screening problem for a Markovian target and an exponential detection function was obtained in 1980 by Brown [32]. His work initiated discussions and formulations of the search problem from various viewpoints and stimulated further studies. Despite the fact that a complete formal theory of search and screening has not been created, ongoing research work has covered many reasonable applications and aspects of this problem. For reviews of the search and screening theory and various military applications, see [5, 6, 33–36]. Recent results of the theory are also presented in [31, 37]. Further developments of the problem as formulated by Brown [32] are presented in [38].

In search and screening theory, it is assumed that the searcher has such available resources that the reaction of the target to the searcher's actions may be omitted. In special cases, such a reaction is represented by the *detection probability*, which is defined by a detection function [33]. If the behavior of the target strongly depends on the searcher's actions, then the search process is named the *pursuit-evasion game*, which is also called a *search game*.

A search game known as ‘the Princess and the Monster’ is a pursuit-evasion game that was formulated in 1965 by Isaacs [2]. In this game, the goal of the searcher is to find the target, whereas the target is informed about the searcher's behavior and tries to escape interception. In the simplest case, the game is formulated as a *hidden-object problem*, which applies to a search for a static target. In the case of a moving target, a game theory aspect appears in the form of a *rationality assumption* that the target at each step of the game follows the best actions to avoid interception. An overview of the methods and results obtained in this game theory search are presented in the monographs in [20–22] and in [30].

Another formulation of the search problem has its origins in *group testing* for cases where the searcher can observe simultaneously more than a single point in the space at each search action. Such a solution assumes that the costs of the searcher's actions do not depend on the size of the chosen subsets and do not change over time.

Originally group-testing procedures focused on diagnostic tasks and quality inspections. Group testing was proposed during World War II by Dorfman [39] as a testing strategy to identify defective units. The strategy involves testing a set of units simultaneously, taking into account that probably most of these units are acceptable. Therefore, there is a high probability of approving many units by a single group test. If, however, the test indicates that there is a faulty unit in the set, then the set is partitioned into subsets, to each of which finer group testing is applied. This iterative procedure of partitioning and testing terminates when the faulty unit is found. The main motivation of the strategy is to minimize testing efforts in terms of minimum average number of tests in a kind of zooming-in search procedure, as indicated by Dorfman [39]:

Often in testing the results of manufacture, the work can be reduced greatly by examining only a sample of the population and rejecting the whole if the proportion of defectives in the sample is unduly large. In many inspections, however, the objective is to eliminate all the defective members of the population.

The goal of these studies, according to Dorfman [39], is ‘under certain conditions, [to] yield significant savings in effort and expense when a complete elimination of defective units is desired.’

The development of group-testing theory is closely related to other statistical theories, such as Wald’s theory of *statistical decision functions* [40], the theory of *statistical games* [29], and Raiffa’s Bayesian *decision analysis* [41]. An overview of the main methods and results of the theory are presented in the monographs in [42, 43].

Group-testing theory deals mainly with the problem of *statistical decisions*. According to Wald [40], this problem can be formulated as selecting the best action under uncertainty with regard to the state of nature given payoffs, or rewards for possible actions. In the simplest case, the decision-maker chooses from continuing or stopping the testing process. Another decision concerns the size of the test samples, which may depend on the test results.

In relation to the considered search problem, the group-testing formulation has the following decision process [10]. It is assumed that there is a set of possible target locations in the search space. At each step, the action available to the searcher is to choose a subset from the set of locations and check whether the target is located somewhere within this subset or not. The procedure terminates if the searcher finds the target in a single-point subset. In other words, the main considerations concentrate on determining the size and location of the samples, given a constant or variable detection function.

The implementation of group testing to a problem of search for a *static target* was suggested in 1959 by Zimmerman [26]. In contrast to conventional statistical methods at that time, Zimmerman’s procedure is directly related to methods of *information theory*, and particularly the Huffman *coding* procedure [44], to solve the search problem.

The analogy between testing (e.g., finding a defective unit) and coding relies on the fact that each unit in the set can be decoded by the sequenced testing outcomes of subsets to which it belongs [7]. The length of the code is thus analogous to the length of the testing procedure required to identify the defective unit. Zimmerman’s offline method is constructed by using the probability of each unit as the defective one. An optimal testing tree is constructed offline by the Huffman coding procedure, and then the test procedure follows this testing tree, assuming that no new information is revealed. Since the Huffman coding algorithm is optimal in the sense of minimal average code length, the analogous search procedure is optimal in the sense of minimal average search length [26].

Other group-testing methods that are based on coding procedures have been suggested over the years. In 1994, the Zimmerman procedure with the same assumptions was generalized by Abrahams [45] for a Huffman search by multiple searchers, and for additional requirements regarding the connectivity of the set of locations to a search based on the Hu–Tucker coding procedure [46]. Ben-Gal [7] analyzed the popular *weight balance tree* (WBT) group-testing algorithm, which is based on the simple principle of successive partitions of the search set into equiprobable subsets (known also as Shannon–Fano coding). At each step of the algorithm, the subset which contains the searched item (i.e., a faulty unit) is partitioned into equiprobable sub-subsets and the process repeats itself until the searched

item is found. Herer and Raz [8] considered a similar group-testing search for a defective product in a lot produced by a process with a constant failure rate.

Group-testing methods of search based on coding procedures essentially rely on the possibility of checking a whole subset in one search act. Nevertheless, for certain cases of location probabilities, these methods solve the problem in the form of sequences of single-point subsets. For such cases, the methods yield optimal solutions of the hidden-object problem indicated above.

The absence of a general optimal solution for the hidden-object problem contributed to the application of dynamic programming methods, as initiated in 1969 by Ross [47]. In these cases, it is assumed that, similar to the theory of statistical decision functions and hypothesis testing, there is a probability that the target will not be detected at its real location. Thus, equivalent to the Type I error or false negative testing, the hypothesis on the location of the target is correct, although the test statistic leads to the wrong decision. In further studies [48, 49], this problem was formulated as a Markov decision process (MDP), yet the assumption of an imperfect detection, as it is often called, remains and is used for estimating the convergence of the search process.

The first solution for the simplest variant of the considered search problem with a *moving target* was published in 1970 by Pollock [50]. He considered a search for a target moving over two boxes (or two points in the search space) according to known probabilities and presented an optimal threshold policy for the searcher moves in order to find the target in minimal time. Approximate computation of the values of the thresholds was given in [51]. In later studies [52–57], a number of attempts were made to generalize the Pollock model to an arbitrary number of boxes, but a general solution has yet to be found. The search and screening problem and the group-testing search were also formulized and addressed by operations research techniques. However, for both problems, general optimal solutions were not obtained. Hence, for these cases, Ross's statement that *the searching for a moving target is an open problem* [28] remains valid and actual.

Another approach to the search problem was initiated by Dechter and Pearl in 1985 [58] in the field of AI. In these studies, a solution for learning the optimal path toward a hidden object with equal location probabilities was studied. The method, called the A* algorithm, is considered a good and general strategy for the *optimal path-learning problem*. In the introduction to their work, Dechter and Pearl write [58] (authors' italics):

One of the most popular methods of exploiting heuristic information to cut down search time is the *informed best-first strategy*. The general philosophy of this strategy is to use the heuristic information to access the ‘merit’ latent in every candidate search avenue exposed during the search and then continue the exploration along the direction of highest merit.

The success of such an approach and its relevance and association to optimization methods gave rise to considerations of different heuristics in the search problem. In 1990, Korf [59] suggested a modification of the A* algorithm by introducing heuristic learning abilities; the obtained algorithm was named the *Learning Real-Time A** (LRTA*) algorithm [59]. In this algorithm, the searcher moves over the vertices of a graph until the static target in one of the vertices is found.

In 1991, a generalization of Korf's algorithm for an application of search for a moving target was suggested. The algorithm, called the *Moving Target Search* (MTS) algorithm, was presented by Ishida and Korf [60] (see also [61]). Similar to Korf's algorithm, the

MTS algorithm does not use any probabilistic information about the target location and is based on a pessimistic assumption (a worse-case scenario) regarding the target moves.

Further studies in the field addressed the implementation of different heuristics and assumptions in the MTS framework. In 1995, Koenig and Simmons [62] presented the first variant of the algorithm acting on a graph with changing probabilities of the target location and using a constant search strategy [63]. In 2006, Koenig and Likhachev [64] presented a revised version of the Koenig and Simmons algorithm, which allows for changing the strategy during the search. An overview of algorithms with non-constant search strategies, as well as their results, is presented in the paper [65] by Shimbo and Ishida. Variants of A* search algorithms were introduced in a unified framework [66–67] and considered to be associated with the theory of heuristic search on the graphs.

1.4 Methods of local search

One of the most studied heuristics for searching for an optimal solution is called the *local search* approach. Such a search, which is in essence over a virtual domain, acts over a graph or a discrete space of possible solutions called the *search space*. The above-mentioned A* and LRTA* algorithms [58, 59] follow this approach for the tasks of AI [15], while the stochastic local search algorithms [12, 13] implement the local search approach for solving various optimization problems.

The main idea of the local search and Stochastic Local Search (SLS) algorithms has been described in Hoos and Stützle [12]. The searcher starts with some solution of the problem that is represented by an initial location in the search space and sequentially moves from the current location to a neighboring location. The process continues until a location that represents a suitable solution of the problem is reached. The neighborhood of each location includes a set of candidate solutions, thus the decision at each step is obtained on the basis of local information only, which reduces the search space immensely, leading to computational tractability. The local search algorithms, in which the decision making is based on randomized choices of candidate solutions from the neighboring set, are called Stochastic Local Search algorithms.

Formally, the local search algorithms are defined for an instance of the optimization problem and use the following objects [12]:

- a *set of candidate solutions* of the problem called *search positions, locations, or states*;
- a *set of feasible solutions* that is a subset of the set of candidate solutions;
- a *neighborhood relation* that determines the neighboring solutions for each candidate solution;
- a *step function* that, given a current solution, defines the choice of the next solution taken from the neighboring solutions; and
- a *termination condition* that interrupts or ends the search process.

For the SLS algorithms, in addition, an *initialization function* that specifies an initial probability distribution over a set of candidate solutions is often defined. In this case, the

step function defines a probability distribution over the neighbors of the current solution so that the decision making is conducted by using this distribution solely over the neighbors.

As indicated above, the same local search approach is popular in AI algorithms of *solving problems by search* [15]. Similar to classical optimization problems, in such AI algorithms the *set of states* (or solutions) with an *initial state* and a termination condition, which is often called the *goal test*, are defined. In AI algorithms, the neighborhood relation is determined by a *set of actions* that can be conducted by the search agent for any state. In addition to the defined components, the AI algorithms implement a *path cost* that measures the performance of the search process and a *step cost* that is used for decision making while choosing an action.

Based on these principles, a local search algorithm implements the following general procedure [12, 15]:

Given a set of feasible solutions:

- *Set the current solution as an initial solution.*
- *While the termination condition for a current solution does not hold, do the following:*
 - *Determine neighboring solutions or possible actions for a current solution.*
 - *Choose a next solution from the set of neighboring solutions or choose and apply an action from the set of possible actions.*
 - *Set the current solution as an obtained solution.*
- *End while.*
- *Return current solution.*

An implementation of the local search algorithms that follows this outlined procedure depends on the definition of a neighboring relation and on the principles of decision making regarding the next solutions or applied actions. In the optimization tasks [12], the neighboring solutions can be generated from a current solution by changing it according to a certain perturbation/updating rule, often related to the function gradient if it exists. In AI tasks [15], the neighboring solutions are usually obtained by constructive extensions of the current solution. Such an approach, as well as a search over a discrete set of solutions (states), can define a *graph search* that is represented by a walk over the graph of solutions from an initial solution to the final solution. In the graph search, the goal of the searcher is to find a path over the graph vertices such that the path cost is minimal. The path cost is defined as the sum or combination of step costs that depend on the distances between the chosen solutions. The distances are calculated by using suitable metrics or pseudo-metrics that represent the differences between the solutions in the context of the considered problem.

A number of approaches to decision making exist, such that for a given current solution, they provide a choice (set of possible selections) of the next solution from the neighboring solutions. Selection of an action results in the next solution with a new neighborhood and, accordingly, a new set of possible selections. The method of decision making essentially depends on the nature of the problem and available data. However, a general decision-making procedure with solutions and action choices, both in the SLS and in the AI local search, follows the framework of a MDP. As indicated, this framework provides a general description of sequential decision making [49, 68].

Within this framework, for each state that represents a candidate solution of the problem, a set of actions is defined, and an action is chosen independently of the decisions that were obtained for the previous solutions. In other words, the current solution includes by definition all the required information on previously obtained solutions.

The decision making in MDP is conducted on the basis of a certain probabilistic rule and probability updating. Following the SLS approach [12, 13], these probabilities are defined by an *initialization function* that specifies initial probabilities for the candidate solutions that aim to solve the problem, and then the probabilities are updated according to some reasonable probabilistic or informational scheme. In the group-testing tasks [10], such a rule is provided by the Bayesian approach, while in the other cases, non-Bayesian methods are applied. In the AI local search [15], these probabilistic methods are not implemented directly, since a probability function is not defined explicitly over the search space. However, the considered tasks can be formulated by using probabilistic or, more generally, measure theory forms, and can be studied following the unified framework that includes the methods of both SLS and AI local search. Within this approach, we present in this book the *Informational Learning Real-Time A** (ILRTA*) and the *Informational Moving Target Search* (IMTS) algorithms [69, 70].

Similar to LRTA* and MTS, the proposed ILRTA* and IMTS algorithms operate on a graph. However, unlike the former methods, in the ILRTA* and IMTS the graph vertices represent partitions. These partitions stand for the possible results of the searcher's group tests. For example, if the searcher tests a subset of points where the target might be located, the relevant partition of points following the test is represented by the corresponding vertex in the graph. The searcher's decision to move to a certain vertex in the graph represents the selection and execution of a specific group test. The target location can be represented by the *target partition* that corresponds to the target location point and the complementary subset of all the other points. The search is terminated when the searcher selects a partition which is either identical or refined with respect to the target partition; we thus call it the *final partition* since the search ends at that stage. Reaching such a partition implies that the searcher obtains the necessary information regarding the target location, as a result of the group test. Similar to the LRTA* [59] and MTS [60] algorithms, the edges between the vertices in ILRTA* and IMTS are weighted by the corresponding distance measures. However, since these distance measures are between pairs of partitions, the ILRTA* and IMTS algorithms apply specific information theory distance measures, and, in particular, they use the Rokhlin distance metric which is based on the relative entropies of these partitions [71].

As will be further explained in later chapters, similar to other SLS procedures (e.g., [62, 64]), in the ILRTA* and IMTS algorithms the partitions with *known* distances to the current searcher's partition define a *neighborhood*. The distances to all the other points in the graph have to be estimated and refined when new information is obtained from the next group tests. The iterative ILRTA* and IMTS algorithms seek to find the shortest path between the searcher partition and the target partition, and thus select a path that correspond to the group test results. In summary, as in other search algorithms, the proposed ILRTA* and IMTS implement a heuristic search based on distance estimates between the target partition and feasible search partitions. The uniqueness of these methods is that they use informational metrics and non-binary partitions to represent a group-testing search process over a graph of partitions (see also [72, 73]).

1.5 Objectives and structure of the book

This book describes various problems and solution methods that are related to the probabilistic search problem. It aims at providing a somewhat unified understanding of how to use stochastic local search principles combined with information theory measures to address the search problem in general. In the book we describe the main methods that have been used in the context of search problems over the years. We address the problem of search for static and moving targets in the context of group-testing theory and apply the methods and measures of information theory to different A*-type heuristics. It is assumed that the target is located or moves within a discrete set of possible locations. The action available to the searcher is checking a subset of locations (points) to determine whether the target is somewhere in this subset or not. In some cases the subset includes only a single point to represent a regular search. The procedure terminates if the searcher finds the target in a subset that contains only one point. The movements of the target and the searcher are considered in discrete time, and the searcher's goal is to choose a sequence of subsets such that the search procedure terminates in a minimal expected number of steps.

The research aims to suggest a unified framework for a search for both static and moving targets, and to present novel algorithms of search for static and moving targets. The approach is based on the following key ideas that distinguish it from previous known methods.

A key feature of the proposed search procedure is that it acts on the search space that is a *set of partitions* of the sample space, and not on the set of locations. In contrast to known models of search in the form of MDPs, the decision-maker in the proposed search procedure takes into account both the chosen subset of the sample space and a set of subsets, which complete the chosen subset for the full sample space.

Using the set of partitions of the sample space enables different distance measures to be applied between candidate solutions in the proposed search scheme. In particular, we apply the *Rokhlin metric* [71] as a distance measure between partitions, and in certain case we use the *Ornstein metric* [74] as an estimation of the Rokhlin distance. The implementation of such metrics allows us to associate the considered search procedures within a broader context of general *ergodic theory* and *dynamical systems* theory [71].

Since the Rokhlin metric is essentially based on the *relative entropies* of the considered partitions, its application clarifies the connection between search procedures in information theory and coding procedures. This, in turn, allows us to apply information theory methods in considerations of the search procedures.

The rest of the book is organized as follows:

Chapter 2. In this chapter we describe the search problem as it appears in different applications. We start with the search and screening problem and describe the solution methods that use global optimization techniques and local search methods. For the group-testing theory that represents a search with unrestricted searcher abilities, we describe basic information theory methods and combinatorial methods. The search over graphs provides the most general search heuristics as they are implemented in optimization tasks and in AI. Finally, we provide notes on search games over continuous domains and over graphs.

Chapter 3. In this chapter we describe mathematical methods and models that are applied to the search problem. We start with the framework of MDPs and continue with the

methods of search and screening and group-testing search. Finally, we discuss some game theory models of search.

Chapter 4. This chapter focuses entirely on methods of *information theory search*. We then develop a search procedure that is based on the partitions space, as obtained by the searcher and the target moves. For this purpose we use the Rokhlin distance between the partitions as distance measures. We formulate an algorithm of search for static targets, which is based on the LRTA* algorithm, and prove its main properties. We then consider the informational algorithm of search for a moving target, which is based on the MTS algorithm. Finally, we present certain generalizations on the basis of the ILRTA* and MTS algorithms, such as search with multiple searchers for both cooperative and non-cooperative behavior.

Chapter 5. This chapter considers some related applications and perspectives of the search problem. We address a known data-mining problem of data classification and present an algorithm for constructing classification trees by using a recursive version of the ILRTA* algorithm. Another application presents a real-time algorithm of search for static and moving targets by single and multiple searchers.

Chapter 6. This chapter concludes the book with some general observations and future research directions.

References

1. P. J. Nahin (2007). *Chases and Escapes: The Mathematics of Pursuit and Evasion*. Princeton University Press: Princeton, NJ, USA.
2. R. Isaacs (1965). *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley and Sons, Inc.: New York, London, Sydney.
3. L. S. Pontryagin, V. G. Boltyansky, R. V. Gamkrelidze, E. F. Mishchenko (1961). *Mathematical Theory of Optimal Processes*. Fizmatgiz: Moscow. (In Russian).
4. B. O. Koopman (1946). Search and Screening. *Operation Evaluation Research Group Report*, 56, Center for Naval Analysis, Rosslyn, Virginia. See also: B. O. Koopman (1956–57). The Theory of Search, I–III. *Operations Research*, 1956, 4, 324–346; 1956, 4, 503–531; 1957, 5, 613–626.
5. L. D. Stone (1975). *Theory of Optimal Search*. Academic Press: New York, San Francisco, London. Second edition: L. D. Stone (1989). *Theory of Optimal Search*. 2-nd edition. INFORMS: Linthicom, MD.
6. A. R. Washburn (1982). *Search and Detection*. ORSA Books: Arlington, VA.
7. I. Ben-Gal (2004). An Upper Bound on the Weight-Balanced Testing Procedure with Multiple Testers. *IIE Transactions*, **36**, 481–493.
8. Y. T. Herer, T. Raz (2000). Optimal Parallel Inspection for Finding the First Nonconforming Unit in a Batch – An Information Theoretic Approach. *Management Science*, **46**(6), 845–857.
9. A. Bar-Noy, I. Kessler, M. Sidi (1995). Mobile Users: To update or not to update? *Wireless Networks*, **1**(2), 175–185.
10. R. Ahlsweide, I. Wegener (1987). *Search Problems*. John Wiley & Sons: New York.
11. D.-Z. Du, F. K. Hwang (1993). *Combinatorial Group Testing and its Applications*. World Scientific Publishing: Singapore etc.
12. H. H. Hoos, T. Stützle (2005). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers: Amsterdam, etc.

13. W. Michiels, E. Aarts, J. Korst (2007). *Theoretical Aspects of Local Search*. Springer-Verlag: Berlin, Heidelberg.
14. J. Pearl (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley: Reading, Massachusetts.
15. S. Russell, P. Norvig (2010). *Artificial Intelligence. A Modern Approach*. 3-rd edition. Pearson Education Inc.: Upper Saddle River, NJ.
16. M. Aoki (1967). *Optimization of Stochastic Systems*. Academic Press: New York, London.
17. K. J. Åström (1970). *Introduction to Stochastic Control Theory*. Academic Press: New York.
18. D. P. Bertsekas, S. E. Shreve (1978). *Stochastic Optimal Control: The Discrete Time Case*. Academic Press: New York.
19. B. O. Koopman (1980). *Search and Screening: General Principles with Historical Applications*. Pergamon Press: New York.
20. S. Alpern, S. Gal (2003). *The Theory of Search Games and Rendezvous*. Kluwer Academic Publishers: New-York, etc.
21. S. Gal. *Search Games*. Academic Press: New York, 1988.
22. A. Y. Garnaev (2000). *Search Games and Other Applications of Game Theory*. Springer-Verlag: Berlin.
23. D. Knuth (1998). *Sorting and Searching. The Art of Computer Programming*. V. 3. Second edition. Addison-Wesley: Reading, Massachusetts.
24. T. M. Cover, J. A. Thomas (1991). *Elements of Information Theory*. John Wiley & Sons: New York, etc.
25. R. S. Sutton, A. G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press: Cambridge, MA.
26. S. Zimmerman (1959). An Optimal Search Procedure. *American Mathematics Monthly*, **66**, 690–693.
27. S. M. Ross (1969). A Problem in Optimal Search and Stop. *Operations Research*, **17**(6), 984–992.
28. S. M. Ross (1983). *Introduction to Stochastic Dynamic Programming*. Academic Press: New York.
29. D. Blackwell, M. A. Girshik (1954). *Theory of Games and Statistical Decisions*. Dover Publications Inc.: New York.
30. P. D. Grünwald, A. P. David (2004). Game Theory, Maximum Entropy, Minimum Discrepancy and Robust Bayesian Decision Theory. *The Annals of Statistics*, **32**(4), 1367–1433.
31. J. R. Frost, L. D. Stone (2001). *Review of Search Theory: Advances and Applications to Search and Rescue Decision Support*. US Coast Guard Research and Development Center, Groton.
32. S. S. Brown (1980). Optimal Search for a Moving Target in Discrete Time and Space. *Operations Research*, **28**(6), 1275–1289.
33. O. Hellman (1985). *Introduction to the Optimal Search Theory*. Nauka: Moscow. (In Russian).
34. O. Hellman (1972). On the Optimal Search for a Randomly Moving Target. *SIAM J. Applied Mathematics*, **22**, 545–552.
35. H. R. Richardson (1986). Search Theory. Professional Paper 449. Center for Naval Analysis, Alexandria, Virginia.
36. A. R. Washburn (1998). Branch and Bound methods for a Search Problem. *Naval Research Logistics*, **45**, 243–257.
37. D. C. Cooper, J. R. Frost, R. Quincy Robe (2003). *Compatibility of Land SAR Procedures with Search Theory*. US Department of Homeland Security, Washington.

38. F. Dambreville, J.-P. Le Cadre (1999). *Detection of a Markovian Target with Optimization of the Search Efforts under Generalized Linear Constraints*. Technical Report 1278. Institut de Recherche en Informatique et Systèmes Aléatoires, France.
39. R. Dorfman (1943). The Detection of Defective Members of Large Population. *Annals of Mathematical Statistics*, **14**, 436–440.
40. A. Wald (1950). *Statistical Decision Functions*. John Wiley & Sons: Chichester, etc.
41. H. Raiffa (1968). *Decision Analysis. Introductory Lectures on Choices under Uncertainty*. Addison-Wesley: Reading, Massachusetts.
42. J. O. Berger (1985). *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag: Berlin.
43. M. H. DeGroot (1970). *Optimal Statistical Decisions*. McGraw-Hill Company: New York, etc.
44. D. A. Huffman (1952). A Method of the Construction of Minimal-Redundancy Codes. *Proceedings of I.R.E.*, **40**, 1098–1101.
45. J. Abrahams (1994). Parallelized Huffman and Hu-Tucker Searching. *IEEE Transactions on Information Theory*, **40**(2), 508–510.
46. T. C. Hu, A. C. Tucker (1971). Optimum Computer Search Trees and Variable-Length Alphabetical Codes. *SIAM J. Applied Mathematics*, **21**, 514–532.
47. S. M. Ross (1969). A Problem in Optimal Search and Stop. *Operations Research*, **17**(6), 984–992.
48. S. M. Ross (2003). *Introduction to Probability Models*. Academic Press: San Diego, London, etc.
49. D. J. White (1993). *Markov Decision Processes*. John Wiley & Sons: Chichester, etc.
50. S. M. Pollock (1970). A Simple Model of Search for a Moving Target. *Operations Research*, **18**, 883–903.
51. P. J. Schweitzer (1971). Threshold Probabilities when Searching for a Moving Target. *Operations Research*, **19**(3), 707–709.
52. J. N. Eagle (1984). The Optimal Search for a Moving Target when the Search Path is Constrained. *Operations Research*, **32**, 1107–1115.
53. D. A. Grundel (2005). Searching for a Moving Target: Optimal Path Planning. *Proceedings of IEEE Conference on Networking, Sensing and Control*, 19–22 March, 2005, 867–872.
54. I. M. MacPhee, B. P. Jordan (1995). Optimal Search for a Moving Target. *P.E.I.S.*, **9**, 159–182.
55. S. Singh, V. Krishnamurthy (2003). The Optimal Search for a Moving Target when the Search Path is Constrained: The Infinite-Horizon Case. *IEEE Transactions on Automatic Control*, **48**(3), 493–497.
56. A. Stenz (1994). Optimal and Efficient Path Planning for Partially-Known Environments. *Proceeding of IEEE International Conference on Robotics and Automation*, San Diego, CA, USA, May 8–13 1994, vol. 4, 3310–3317.
57. L. C. Tomas and J. N. Eagle (1995). Criteria and Approximate Methods for Path-Constrained Moving-Target Search Problems. *Naval Research Logistics*, **42**, 27–38.
58. R. Dechter, J. Pearl (1985). Generalized Best-First Search Strategies and the Optimality of A*. *Journal of ACM*, **32**(3), 505–536.
59. R. E. Korf (1990). Real-Time Heuristic Search. *Artificial Intelligence*, **42**(2–3), 189–211.
60. T. Ishida, R. E. Korf (1991). Moving Target Search. *Proceedings of International Joint Conference on Artificial Intelligence* (IJCAI-91), 204–210.
61. T. Ishida, R. E. Korf (1995). Moving Target Search: A Real-Time Search for Changing Goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**(6), 609–619.
62. S. Koenig, R. G. Simmons (1995). Real-Time Search on Non-Deterministic Domains. *Proceedings of International Joint Conference on Artificial Intelligence* (IJCAI-95), 1660–1667.

63. S. Koenig (2001). Mini-Max Real-Time Heuristic Search. *Artificial Intelligence*, **129**(1–2), 165–197.
64. S. Koenig, M. Likhachev (2006). Real-Time Adaptive A*. *Proceedings of AAMAS'06*, 2006, 281–288.
65. M. Shimbo, T. Ishida (2003). Controlling the Learning Process of Real-Time Heuristic Search. *Artificial Intelligence*, **146**, 1–41.
66. V. Bulitko, G. Lee (2006). Learning in Real-Time Search: A Unifying Framework. *J. Artificial Intelligence Research*, **25**, 119–157.
67. V. Bulitko, N. Sturtevant, J. Lu, T. Yau (2007). Graph Abstraction in Real-Time Heuristic Search. *J. Artificial Intelligence Research*, **30**, 51–100.
68. C. Derman (1970). *Finite State Markov Decision Processes*. Academic Press: New York.
69. E. Kagan, I. Ben-Gal (2013). *A Group-Testing Algorithm with Online Informational Learning*. To appear in *IIE Transactions*.
70. E. Kagan, I. Ben-Gal (2013). *Moving Target Search Algorithm with Informational Distance Measures*. To appear in *Entropy*.
71. V. A. Rokhlin (1967). Lectures on the Entropy Theory of Measure-Preserving Transformations. *Russian Mathematical Surveys*, **22**, 1–52.
72. S. Jaroszewicz (2003). *Information-Theoretical and Combinatorial Methods in Data-Mining*. PhD Thesis: University of Massachusetts, Boston.
73. J. Peltonen (2004). *Data Exploration with Learning Metrics*. PhD Thesis: Helsinki University of Technology, Finland.
74. D. S. Ornstein (1974). *Ergodic Theory, Randomness, and Dynamical Systems*. Yale University Press: New Haven and London.

2

Problem of search for static and moving targets

As indicated in Chapter 1, the problem of search can be formulated in different ways and can be addressed by the use of several approaches that depend on the considered theoretical or practical task. In this chapter we consider three main approaches that cover most of the tasks in which search problems appear.

The first is the search and screening approach that considers the problem of a physical search for a hidden object or a moving target by a search agent. The problem was initially formulated in such a form for the purpose of military applications and later applied to a wide class of operations research and engineering tasks. Under the assumption that the moving target is informed about the searcher's actions and tries to escape from the searcher, the problem can be considered as a search game.

The second approach implements the methods of group-testing theory and is popular in industrial and quality engineering applications. The main assumption behind this approach is that the searcher can check simultaneously a *subset* of the domain in which the target acts, and that the order of the checks can be chosen arbitrarily.

The third approach is associated more with a stochastic local search. It often deals with problem-solving tasks by searching for solutions in a feasible search space as is often considered in optimization theory and in AI tasks. In many cases, such problems are formulated as search over a graph of solutions and implement either methods of global optimization or, more likely, the methods of local search that are applied to *NP*-hard problems. Under suitable assumptions regarding the search space, such an approach forms a general framework for different search problems that can be formulated as a search in discrete space and time.

In the following sections, we present an overview of these approaches including some remarks on terminology.

2.1 Methods of search and screening

The problem of *search and screening* is associated with search tasks in which it is required to find the target in a given domain by a finite number of search agents with restricted search abilities. The probability of the target to be located at a point in the domain is defined by the *location probability* density function. The search agents start with some initial assumption or knowledge about the location probabilities, and at each search step they obtain information regarding the target location in a certain restricted area. If the target is static and the observations are error free, then often the location probabilities do not depend on time and are updated according to the results of the search. For a moving target, the location probabilities are determined both by the movement rules of the target and by the search results at each step. The goal of the search and screening process is to find such a search policy, that is, the set of rules of the search agents' movements that maximizes the probability of detecting the target in minimal time or with minimal search effort.

The study of the problem of search in the form of search and screening was initiated in 1942 by the US Navy's Antisubmarine Warfare Operations Research Group. In 1946, Koopman published a detailed report [1] (a brief review of this report was given in [2]), where he presented the widely accepted formulation of the problem. In later studies, the problem was reformulated using different assumptions about the detection probability and the detection conditions [3], and a large body of reports and manuscripts, mainly focused on military and security applications, was published [4–7]. In the next section, a search and screening scheme over a gridded plane in discrete time is introduced and accompanied by general remarks on the continuous version of this search problem.

2.1.1 General definitions and notation

Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of points that corresponds to a gridded rectangular domain of a Euclidean plane, such that each point $x_i \in X$, $i = 1, \dots, n$, represents a cell in which a target can be located. Alternatively, for certain versions of the search problem, X can denote a set of nodes in a graph, on which the search is conducted and where the target can be located. In any case, the set X of all target locations is called here the *locations set*. We assume that the cells do not overlap, so the elements of set X can be considered as discrete coordinates of the centers of cells. For simplicity but without loss of generality, it is assumed that the points are indexed in linear order.

Let $t = 0, 1, 2, \dots$ be discrete times, and suppose that for any time t a probability mass function $p^t : X \rightarrow [0, 1]$ is defined to represent some knowledge of the target location. In particular, for each point $x_i \in X$ and each time t , $p^t(x_i)$ is the probability that at time t the target is located at x_i . Indeed, for any t it follows that $0 \leq p^t(x_i) \leq 1$ and $\sum_{i=1}^n p^t(x_i) = 1$. The function p^t is called here the *location probability function*, while its values $p^t(x_i)$ are called *location probabilities*.

The locations set X represents the area over which the target can act. In the general case the target location $x^t \in X$ at time t is unknown to the search agent during the search process. The location probabilities $p^t(x_i)$, on the other hand, are known to the search agent and can be thought of as quantities defined or estimated by the search agent or by an additional external agent that represent the accumulated knowledge of the target location. Note that a uniform probability can represent a situation where the agent has no knowledge or assumptions regarding the target location. The locations set X along with the defined

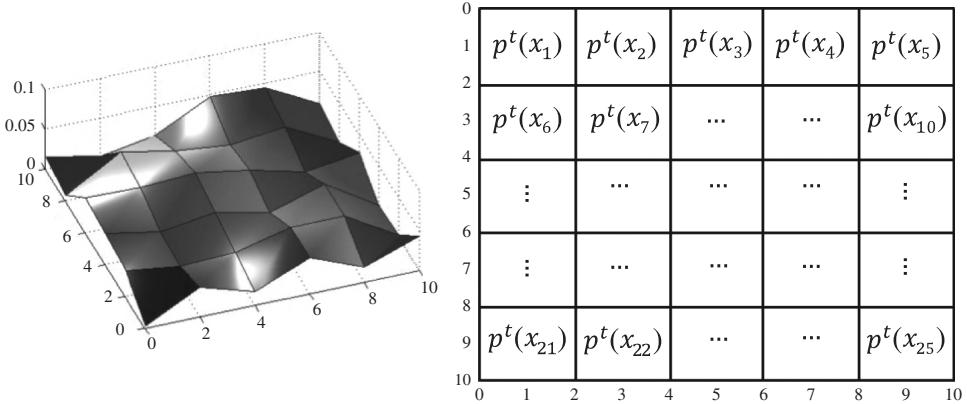


Figure 2.1 Example of gridded sample space and its associated location probabilities.

location probabilities function p^t is often defined as the *sample space* and differs from the *search space* on which the search agent acts and over which the search is conducted, as will be explained later. An example of a gridded sample space X and the corresponding location probabilities $p^t(x_i)$, which are represented for illustrative purposes by levels of gray, are shown in Figure 2.1, where darker tints represent higher location probabilities.

In the figure the indexing of points x_i over the two-dimensional sample space of size $n_1 \times n_2 = n$ is obtained according to the conventional indexing equation $i = ((j_1 - 1)n_1 + j_2)$, where $j_1 = 1, \dots, n_1$ and $j_2 = 1, \dots, n_2$.

The search and screening process is performed as follows. The search agent screens the set X by a suitable sensor that detects whether or not the target is located at a certain *observed area* (a subset of the space) $A = \{x_1, x_2, \dots, x_m\} \subset X$. In the simplest case, the observed subset includes a single point, while for various sensors the observed area A can include more than a single point, depending on the sensors' resolution. The engineering characteristics of the sensors determine the agent's ability to obtain information about the target location within an observed area. The ability of the search agent to detect the target is represented by a *detection function* φ which for any time t defines the probability of detecting the target, given that it is located within the observed area. An application of the detection function results in a probability distribution over the observed area that is obtained and recalculated after the observation. An errorless detection function is often denoted by z , called the *observation function*, and results unambiguously in a 'true' value if the target is indeed located in the observed area at time t , and in a 'false' value otherwise.

After observing the area A and obtaining the *observation result*, the agent updates the location probabilities $p^t(x_i)$, $x_i \in X$, over the entire sample space. We refer to these updated probabilities as the *observed probabilities* and denote them by $\bar{p}^t(x_i)$, $x_i \in X$. If the target is static and this is known to the search agent, then the observed probabilities can be considered as equivalent to the location probabilities at the next time $t + 1$, $p^{t+1}(x_i) = \bar{p}^t(x_i)$, $x_i \in X$. In the case of search for a moving target, another layer of complexity is added to the search problem. Then the search agent has to apply some known (or estimated) rule of the target's movement to the observed probabilities and obtain the next time step, namely, the *estimated location probabilities* $\tilde{p}^{t+1}(x_i)$, $x_i \in X$. Note that, in the general case, since the estimated location probabilities $\tilde{p}^t(x_i)$ are calculated on the basis of observation results and the

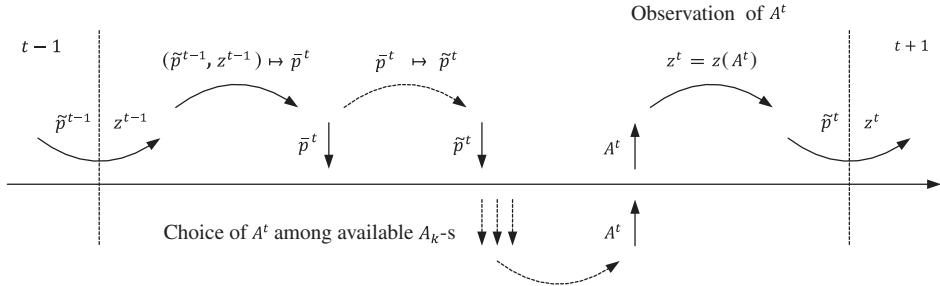


Figure 2.2 A general decision process for the probabilistic search procedure.

assumed target's movement, they differ from the location probabilities $p^t(x_i)$ except at the initial time $t = 0$.

Given the estimated probabilities $\tilde{p}^{t+1}(x_i)$, the searcher has to follow a certain decision-making scheme regarding the selected area to be observed. Then it has to move to a state that provides an observation that is most promising in some sense. If there are few such states, it can choose one of them arbitrarily and continue with the next search step. The search process is illustrated in Figure 2.2.

The choice of the next target location is specified by the rule of the target's movement over set X . Since usually the search agent is not informed of the target's selections, such a rule can be represented by a specific stochastic process that defines the target location probabilities at time $t + 1$, given the location probabilities at previous times $p^{t+1}(x^{t+1} = x_i | x^t = x_j, x^{t-1} = x_k, \dots, x^1 = x_0), x_i \in X$. For example, if it is assumed that the target's movement is governed by a Markov process with known *transition probability* matrix $\rho = [\rho_{ij}]_{n \times n}$, where $\rho_{ij} = \Pr\{x^{t+1} = x_j | x^t = x_i\}$ and $\sum_{j=1}^n \rho_{ij} = 1$ for each $i = 1, 2, \dots, n$, then the estimated location probabilities $\tilde{p}^{t+1}(x_i)$ for the next time $t + 1$ are defined as $\tilde{p}^{t+1}(x_i) = \sum_{j=1}^n \rho_{ij} \bar{p}^t(x_j)$, $i = 1, 2, \dots, n$. On the other hand, if the target is static, then the transition matrix ρ is an $n \times n$ unit matrix, and $p^t(x_i) = p^0(x_i)$, $x_i \in X$, for all times $t = 1, 2, \dots$. In Sections 3.1 and 3.2 we will consider a search for a Markovian target in particular.

As indicated above, the ability of the search agent to obtain information regarding the target location in the observed area A is determined by the *detection function*. This function, $\varphi : X \rightarrow [0, 1]$, maps a *search effort* $\kappa(x, t)$, which has been applied to a point $x \in X$ up to time t , to the probability of detecting the target at point x [6–8]. Such a mapping results in the *detection probability* $\varphi(x, \kappa)$, defined as follows:

$$\varphi(x, \kappa) = \Pr\{\text{target is detected at } x | \text{target is located at } x, \text{ applied search effort is } \kappa\},$$

where a time index is omitted. Thus, for each point $x \in X$ and any search effort $\kappa(x, t)$, it follows that $0 \leq \varphi(x, \kappa) \leq 1$, while in general $\sum_{x \in X} \varphi(x, \kappa) \neq 1$.

The dependence of the detection probability $\varphi(x, \kappa)$ on the search effort $\kappa(x, t)$ can be defined in different ways. Usually, the following relation is assumed for limit cases [8]: if the searcher does not check point x , then the target obviously cannot be detected at this point, that is, a zero search effort $\kappa(x, t) = 0$ applies such that $\varphi(x, 0) = 0$; but if the searcher applies a search effort $\kappa(x, t) \rightarrow \infty$ to point x , then detection probability $\varphi(x, \kappa)$ reaches its upper bound, that is, $0 < \lim_{\kappa(x, t) \rightarrow \infty} \varphi(x, \kappa) \leq 1$.

A strict definition of the detection function φ_κ and its correspondence to the search effort κ^t depends on the applied sensors of the search agent and on the progress of the search process over time. In many of the considered applications [7–9], for any time t and up to the time when the target is found, the detection function is defined according to the Koopman *random-search equation* [1, 2]

$$\varphi(x, \kappa) = 1 - e^{-\kappa(x, t)}. \quad (2.1)$$

According to this equation, the observation function z mentioned above corresponds to an infinite search effort $\kappa(x, t) = \infty$ that yields $\varphi(x, \kappa) = 1$, decaying at an exponential rate with the applied effort.

The distribution of the detection probability over sample space X is determined by a distribution of the search effort κ^t that for any time t is specified by the sensors' abilities and the decision policy of the searcher. Popular examples of the search effort distribution over an observed area A at the beginning of search at $t = 0$ are shown in Figure 2.3 [10].

A search effort distribution that corresponds to a magnetometer or radar with its focus at the point \bar{x} is usually defined by the normal distribution

$$\kappa(x, 0) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \bar{x})^2}{2\sigma^2}\right).$$

In the case of a two-dimensional sample space, the bi-normal distribution

$$\kappa(x, 0) = \frac{1}{2\pi\sigma_1\sigma_2} \exp\left(-\frac{(x_1 - \bar{x}_1)^2}{2\sigma_1^2} - \frac{(x_2 - \bar{x}_2)^2}{2\sigma_2^2}\right)$$

is often used, assuming independence between the axes. The side-looking sonar law is usually related to an ‘on-the-ground’ vehicle that screens the area around its center. For any direction of the sonar, this law is often defined by some continuous approximation of a Poisson distribution, $\kappa(x, 0) = \lambda^x \exp(-\lambda)/x!$. The definite range law corresponds to the indicator function $\mathbf{1}(A, x)$ of the observed area A such that $\mathbf{1}(A, x) = 1$ if $x \in A$ and $\mathbf{1}(A, x) = 0$ otherwise, and describes the test that unambiguously results in ‘true’ or ‘false’ values.

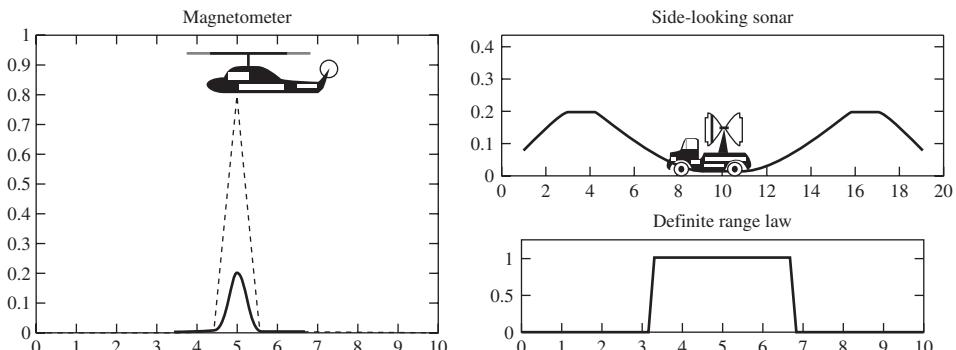


Figure 2.3 Examples of search effort distribution.

A search effort $\kappa(x, t)$ that is applied to point x at time $t > 0$ and depends on the number of search agents and on their search effort κ^t is considered as a *cumulative function* over time; that is, the effort that has been applied to point x up to time t is defined as $\kappa(x, t) = \int_0^t u(x, \tau) d\tau$ for a continuous time search and as $\kappa(x, t) = \sum_{\tau=0}^{t-1} u(x, \tau)$ for a discrete time search. The function $u^t : X \rightarrow [0, \infty)$ specifies the distribution of the search efforts over sample space X and is called the *search density* function. For a search by a number of search agents, the search density function u^t represents the combined effect of simultaneous search by these agents, while for search by a single agent, u^t mainly defines the movement of the search agent over the sample space. Following this terminology, the estimated location probability function \tilde{p}^t is also called in some of the classical literature the *target location density* $v^t : X \rightarrow [0, 1]$. For multiple targets, the density v^t is computed from the location probabilities of all the targets and represents their joint density over the sample space [8, 11].

Note that for any time t and period $(t, t + \varepsilon)$ it follows that $\kappa(x, t) = \sum_{t+\varepsilon}^{t+1} u(x, t) = u(x, t)$, $0 < \varepsilon \ll 1$. In addition, the search effort $\kappa(x, t)$ unambiguously determines the detection function, for example, according to the Koopman random-search equation (Equation 2.1). On the basis of such a relation, some authors [11] apply the term ‘search density’ immediately to the detection function φ_κ without an explicit definition of the search effort κ^t . Such an approach is effective and clear if each observed area A consists of a single point only. An example that implements such an assumption is considered below.

2.1.2 Target location density for a Markovian search

Let us consider a search for a Markovian target in a discrete sample space $X = \{x_1, x_2, \dots, x_n\}$, while the actions of the search agent and the target are conducted in discrete times $t = 0, 1, 2, \dots$. Let us assume that the target’s movements over sample space X are governed by a Markov process with known transition probabilities $\rho_{ij} = \Pr\{x^{t+1} = x_j | x^t = x_i\}$, $\sum_{j=1}^n \rho_{ij} = 1$, for each $i = 1, 2, \dots, n$. In addition, we assume that for time $t = 0$ the initial location probabilities $p^{t=0}(x_i) = \Pr\{x = x_i\}$, $i = 1, \dots, n$, $\sum_{i=1}^n p^t(x_i) = 1$, are defined.

Assume that the target can be detected at point $x \in X$ if it is located at x , and that it cannot be erroneously detected at x if it is not located at this point, that is,

$$\begin{aligned} 0 \leq \Pr\{\text{target is detected in } x | \text{target is located at } x\} &\leq 1, \\ \Pr\{\text{target is detected in } x | \text{target is not located at } x\} &= 0. \end{aligned} \quad (2.2)$$

This is a widely used assumption that omits the identification of fictional targets. A generalization of this assumption for group-testing models of search is described later.

Following this assumption, the target location density $v^t(x)$ at point $x \in X$, which is represented by the estimated location probability function \tilde{p}^t , can be defined by a mapping $\psi^t : X \rightarrow [0, 1]$ that specifies the probability of detecting the target at time t . Denote by D^t the event that the target is detected at time t . Then, following the definition of the detection function φ_κ and the search effort κ^t , the detection probability is defined as

$$\psi(x, t) = \Pr\{D^t | \text{target is located at } x\}.$$

We now apply the above-formulated scheme within the framework of a Markovian search [11]. If the search agent is informed that there is a single target, then the Markovian search terminates when the target is found. If, in contrast, the search agent is not informed about the number of targets, then such a search continues following the same strategy, but with additional counting of the number of detections. Denote by \bar{D}^t the negated event of D^t , that is, $\bar{D}^t = \{\text{target is not detected at time } t\}$, $t = 0, 1, 2, \dots$. Then, for any target location $x^t \in X$ at time t , a Markovian property of the search follows:

$$\psi(x_i, t) = \Pr\{D^t | x^t = x_i, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} = \Pr\{D^t | x^t = x_i\}.$$

The Markovian property of the target's movement implies that it does not depend on the searcher's actions up to termination of the search. Thus, given the target's transition probabilities ρ_{ij} , $i, j = 1, \dots, n$, the independence property holds:

$$\rho_{ij} = \Pr\{x^{t+1} = x_j | x^t = x_i, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} = \Pr\{x^{t+1} = x_j | x^t = x_i\}.$$

Under these assumptions, the target location density $v(x, t)$ at point $x \in X$ at time $t = 0, 1, 2, \dots$ is specified by the following theorem.

Theorem 2.1 [11]. *For any point $x_j \in X$, $j = 1, \dots, n$, the target location density $v(x_j, t)$ at times $t = 0, 1, 2, \dots$ satisfies the following recursive equation under the Markovian property:*

$$v(x_j, t+1) = \frac{\sum_{i=1}^n (1 - \psi(x_i, t)) \rho_{ij} v(x_i, t)}{\sum_{i=1}^n (1 - \psi(x_i, t)) v(x_i, t)},$$

with the initial condition $v(x_i, 0) = p^{t=0}(x_i)$, $i = 1, \dots, n$, where $p^{t=0}$ determines a given initial location probability over the sample space X .

Proof. Following the above assumptions, the target location density at time t is defined by the probability $v(x, t) = \Pr\{x^t = x | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}$ of not detecting the target up to that time. Accordingly, for the next time $t + 1$ it follows that

$$\begin{aligned} v(x_j, t+1) &= \Pr\{x^{t+1} = x_j | \bar{D}^t, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} \\ &= \frac{\Pr\{x^{t+1} = x_j, \bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}}{\Pr\{\bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}} \end{aligned}$$

Let us first consider the probability term in the denominator. The probability $\Pr\{\bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}$ of not detecting the target at point $x_i \in X$ at time t , given that it has not been detected at previous times, is as follows:

$$\begin{aligned} \Pr\{x^{t+1} = x_j, \bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} &= \sum_{i=1}^n \Pr\{x^t = x_i, x^{t+1} = x_j, \bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} \\ &= \sum_{i=1}^n \Pr\{x^{t+1} = x_j, \bar{D}^t | x^t = x_i, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} \\ &\quad \times \Pr\{x^t = x_i | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}. \end{aligned}$$

From the Markovian property it follows that the probability of not detecting the target, given that it is located at point x_i and has not been detected at previous times, is

$$\Pr\{\bar{D}^t | x^t = x_i, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} = \Pr\{\bar{D}^t | x^t = x_i\},$$

where the probability term on the right hand side, according to this definition, is $\Pr\{\bar{D}^t | x^t = x_i\} = 1 - \psi(x_i, t)$.

The target location probability at point $x_i \in X$ at time t , given that it has not been detected at previous times, is given by the target location density, that is, $\Pr\{x^t = x_i | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} = v(x_i, t)$. Thus,

$$\Pr\{\bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} = \sum_{i=1}^n (1 - \psi(x_i, t)) \times v(x_i, t).$$

Now let us consider the probability term in the numerator, $\Pr\{x^{t+1} = x_j, \bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}$, of the target location at point $x_j \in X$ at time $t+1$ and of not detecting the target at time t , given that it has not been detected at previous times. We obtain

$$\begin{aligned} & \Pr\{x^{t+1} = x_j, \bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} \\ &= \sum_{i=1}^n \Pr\{x^t = x_i, x^{t+1} = x_j, \bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} \\ &= \sum_{i=1}^n \Pr\{x^{t+1} = x_j, \bar{D}^t | x^t = x_i, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} \\ &\quad \times \Pr\{x^t = x_i | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}. \end{aligned}$$

As indicated above, $\Pr\{x^t = x_i | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} = v(x_i, t)$; hence, let us consider the probability $\Pr\{x^{t+1} = x_j, \bar{D}^t | x^t = x_i, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}$.

From the independence property it follows that

$$\begin{aligned} & \Pr\{x^{t+1} = x_j, \bar{D}^t | x^t = x_i, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} = \Pr\{x^{t+1} = x_j, \bar{D}^t | x^t = x_i\} \\ &= \Pr\{x^{t+1} = x_j | x^t = x_i\} \times \Pr\{\bar{D}^t | x^t = x_i\}. \end{aligned}$$

The first probability term in the last line of the equation is the transition probability $\Pr\{x^{t+1} = x_j | x^t = x_i\} = \rho_{ij}$, while the second probability term, as indicated above, is the probability of not detecting the target, $\Pr\{\bar{D}^t | x^t = x_i\} = 1 - \psi(x_i, t)$.

Thus,

$$\Pr\{x^{t+1} = x_j, \bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} = \sum_{i=1}^n \rho_{ij} \times (1 - \psi(x_i, t)) \times v(x_i, t),$$

which determines the numerator of the fraction in the required recursive formula. ■

This theorem was proven by Ciervo [11] and provides a simple way to calculate the target location density which for each time specifies a location probability distribution over the sample space. Ciervo's next theorem provides the probability of detecting the target under the Markovian property.

Denote by $Q(t) = \Pr\{\bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}$ the probability of not detecting the target up to time t . Then, the probability of detecting the target up to time t is $P(t) = 1 - Q(t)$.

Theorem 2.2 [11]. *The probability $P(t)$ of detecting the target up to time t is specified by the following formula:*

$$P(t) = 1 - \prod_{\tau=0}^{t-1} \sum_{i=1}^n (1 - \psi(x_i, \tau)) \times v(x_i, \tau)$$

with initial conditions $\psi(x_i, 0) = 0$ and $v(x_i, 0) = p^{t=0}(x_i)$, $i = 1, \dots, n$.

Proof. Let $Q(t+1)$ be the probability of not detecting the target up to time $t+1$.

Then,

$$\begin{aligned} \frac{Q(t+1)}{Q(t)} &= \frac{\Pr\{\bar{D}^t, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}}{\Pr\{\bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}} = \Pr\{\bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} \\ &= \sum_{i=1}^n \Pr\{x^t = x_i, \bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} \\ &= \sum_{i=1}^n \Pr\{\bar{D}^t | x^t = x_i\} \times \Pr\{x^t = x_i | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} \\ &= \sum_{i=1}^n (1 - \psi(x_i, t)) \times v(x_i, t), \end{aligned}$$

from which it follows that

$$Q(t+1) = Q(t) \sum_{i=1}^n (1 - \psi(x_i, t)) \times v(x_i, t).$$

This equation represents a recursive equation for calculating the probability of not detecting the target up to time $t+1$ given such a probability for time t . Thus, for any $t = 0, 1, 2, \dots$ it follows that

$$Q(t) = \prod_{\tau=0}^{t-1} \sum_{i=1}^n (1 - \psi(x_i, \tau)) \times v(x_i, \tau),$$

and from the definition of $P(t)$ one obtains the required equation. ■

The next example illustrates the theorems presented above.

Example 2.1 Let us apply Theorems 2.1 and 2.2 to a search over a square sample space X of size $n = 10 \times 10$. The initial target location densities $v(x_i, 0)$, $i = 1, \dots, n$, are defined by using an initial location probability function $p^{t=0}$ that corresponds to a combination of two bi-normal distributions

$$p_k^{t=0}(x) = \frac{1}{2\pi\sigma_k^2} \exp\left(-\frac{(x_{k1} - \bar{x}_{k1})^2 + (x_{k2} - \bar{x}_{k2})^2}{2\sigma_k^2}\right), \quad k = 1, 2.$$

Movements of the target are governed by a discrete time Markov process with transitions probability matrix $\rho = ||\rho_{ij}||_{n \times n}$ that corresponds to a diffusion process with equal probabilities of staying at the current location, or moving one step to each of four possible directions – north, south, east, and west.

The initial target location densities $v(x_i, 0) = p^{t=0}(x_i)$, $i = 1, \dots, n$, that are defined over the square sample space X with $\sigma_1 = 3$, $\sigma_2 = 2$ and their corresponding coordinates of the centers of distributions (3, 7) and (7, 3) are shown in Figure 2.4a. The target

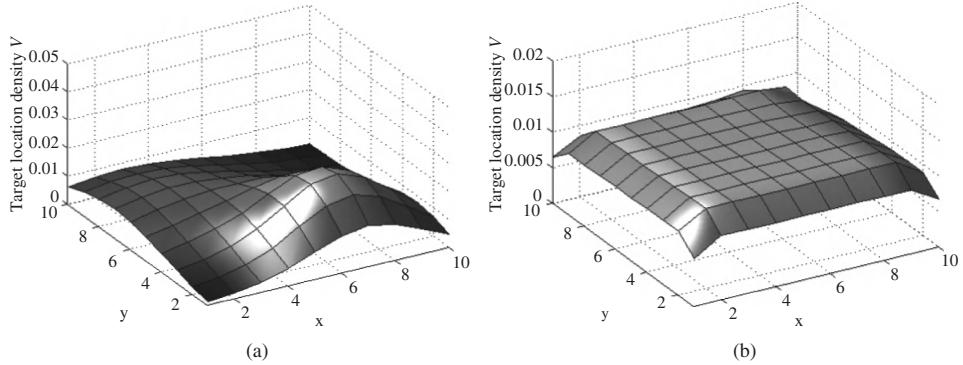


Figure 2.4 Initial and final target location density without search. (a) Initial target location density. (b) Target location density without search at time $t = 100$.

location probabilities $v^{t=100}(x) = v^{t=0}(x) \times \rho^{100}$ at time $t = 100$ without assuming a search process are shown in Figure 2.4b.

Assume that the detection probability function ψ^t does not depend on time, that is, $\psi^{t=0} = \psi^{t=1} = \psi^{t=2} \dots$, and that the detection probabilities $\psi^t(x)$ can be defined by a bi-normal distribution

$$\psi^{t=0}(x) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x_1 - \bar{x}_1)^2 + (x_2 - \bar{x}_2)^2}{2\sigma^2}\right)$$

with center $(5, 5)$. Such detection probabilities correspond to the concentration of search effort in the center of the domain, while at the sides of the domain less search effort is applied.

The detection probabilities are shown in Figure 2.5a, and the resulting target location density at time $t = 100$, given that the target has not been found at that time, is shown in Figure 2.5b. At time $t = 100$ the probability of not detecting the target and the probability

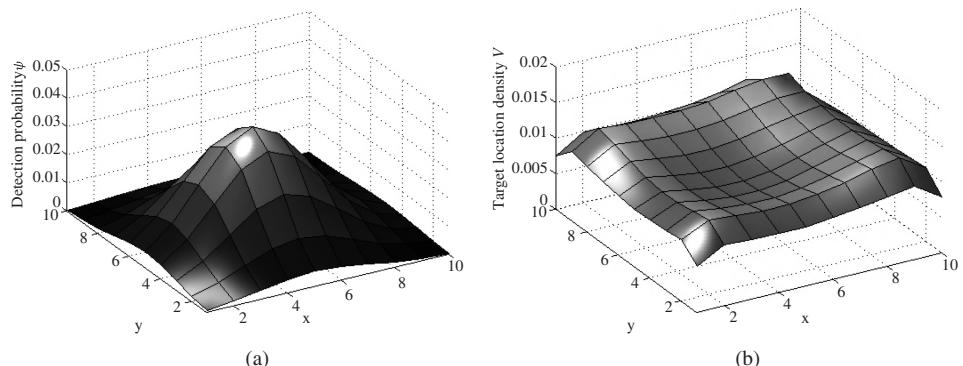


Figure 2.5 Detection probabilities concentrated in the center of the domain and the corresponding target location probabilities. (a) Detection probabilities concentrated in the center of the domain. (b) Target location density at time $t = 100$.

of detecting the target, as calculated by Theorem 2.2, are $Q(100) = 0.36$ and $P(100) = 1 - Q(100) = 0.64$, respectively.

Let us now inspect a case where the detection probabilities $\psi^t(x)$ correspond to a concentration of search efforts at the edges of the domain, while lower search efforts are applied to the center of the domain, thus

$$\psi^{t=0}(x) = C - \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x_1 - \bar{x}_1)^2 + (x_2 - \bar{x}_2)^2}{2\sigma^2}\right),$$

where the bi-normal distribution has the same parameters as used above and $0 < C \leq 1$.

The detection probabilities are shown in Figure 2.6a, and the resulting target location density at time $t = 100$, given that the target has not been found at that time, is shown in Figure 2.6b. The probabilities of not detecting the target and the probability of detecting the target up to time $t = 100$, as calculated according to Theorem 2.2, are $Q(100) = 0.054$ and $P(100) = 1 - Q(100) = 0.946$, respectively.

Note that since the initial target location densities $v(x_i, 0)$, $i = 1, \dots, n$, are concentrated at the domain's edges, the use of a detection probability function that corresponds to a concentration of the search effort on the edges of the domain results in a higher probability of detecting the target at a given time. As time increases, the probability of detecting the target also increases, and for time $t > 500$ it is close to 1 for both the considered cases. ■

The equations presented above provide a direct computational scheme of target location probabilities over time and show the dependence of the target location on the distribution of the search efforts at a given time, given that the target has not been found. In the case of a large number of search agents, the search effort distribution can be generalized to show how these efforts can be distributed among several agents over the search domain. In the case of search by a single agent, such information cannot be utilized directly, as will be shown in later sections.

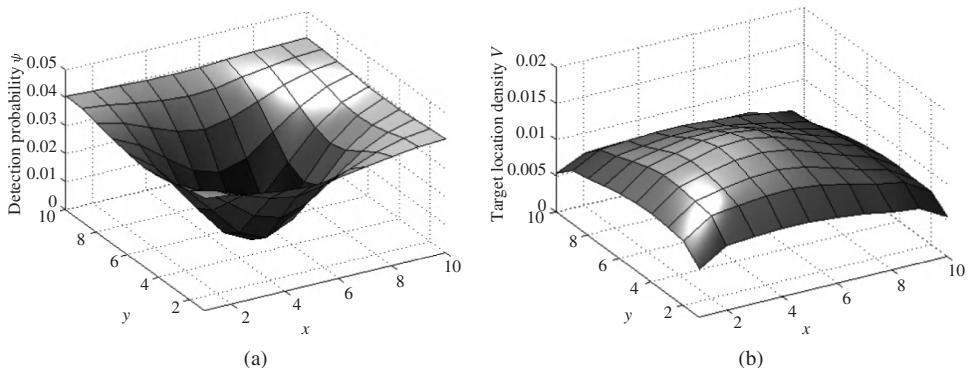


Figure 2.6 Detection probabilities concentrated in the center of the domain and the corresponding target location probabilities. (a) Detection probabilities concentrated at the sides the domain. (b) Target location density at time $t = 100$.

2.1.3 The search-planning problem

In the previous section, we considered the dependence of the target density v^t on the detection function ψ^t and the detection probability ψ^t . Recall that for a single target search, the target density v^t is equivalent to the estimated location probability function \tilde{p}^t . Moreover, at the initial time $t = 0$ the estimated location probabilities are equal to the initial target location probabilities, that is, $\tilde{p}^{t=0}(x) = p^{t=0}(x)$ for each point x in the sample space X . It is further assumed that the search agent knows the initial target location probabilities $p^{t=0}(x)$, $x \in X$. In addition, let us assume that the search agent is equipped with known sensors that imply a corresponding distribution of search effort $\kappa(x, t)$, $x \in A \subset X$, over an observed area A at each time t (e.g., as shown in Figure 2.3). As defined above, recall that under the assumption (Equation 2.2) that there are no fictive targets, it follows that for any point $x \in X$, the value of detection probability $\psi^t(x)$ is equivalent to the value given by a detection function $\varphi_\kappa(x)$ for a search effort $\kappa(x, t)$.

The search-planning problem is as follows [7, 10]. Given the functions indicated above, design a trajectory of the search agent over the sample space X , such that the probability $P(t)$ of detecting the target up to time t reaches its maximum. In other words, given the initial target location probability function $p^{t=0}$ and the detection function φ_κ , it is required to associate the search effort κ^t to the target density v^t .

The required trajectory of the search agent depends on the sensors' features that determine the distribution of search efforts over the observed area. Examples of the trajectories of the search agent for continuous and discrete time are shown in Figure 2.7.

In the case of continuous time, the agent moves over a smooth trajectory $a^t : X \rightarrow X$ and observes a rectangular area with sweep width specified by sensor range. Otherwise, for a discrete time search, the trajectory is defined by a sequence $\overrightarrow{a}(t) = \langle A^{t=0}, \dots, A^{t-2}, A^{t-1}, A^t \rangle$ of observed areas; if the radius of the observed area is greater than the step of the agent, then for the discrete agent's trajectory the observed corridor is still continuous.

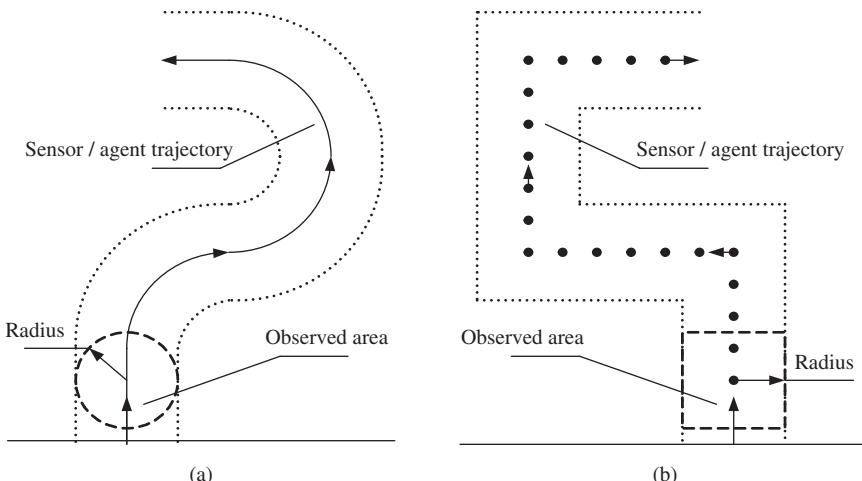


Figure 2.7 Search agent trajectories and observed areas. (a) Continuous space and time. (b) Discrete space and time.

Let us consider a trajectory of the agent in terms of search density function $u^t : X \rightarrow (0, \infty)$ which defines a distribution of the search efforts over the sample space X (see Section 2.1.1). For any time t , the search density over X is called the *allocation* and determined by the *allocation function* $w : X \rightarrow [0, \infty)$. Thus, at any time a certain allocation w is selected among all possible allocations over X from the set W of allocations. If the observed area includes more than one point, then the search effort $\kappa(A, t)$ that is applied to the observed area A at time t is defined as $\kappa(A, t) = \int_A u(x, t) dx = \int_A w(x) dx$ for a continuous sample space or $\kappa(A, t) = \sum_{x \in A} u(x, t) = \sum_{x \in A} w(x)$ for a discrete sample space, where $w \in W$ is the allocation at time t .

Assume that, given a search effort κ^t , the probability of detecting the target is defined by the Koopman random-search equation (Equation 2.1), that is, $\varphi(x, \kappa) = 1 - e^{-\kappa(x, t)}$. Then, to increase the detection probability at point $x \in X$ it is required to increase the search effort $\kappa^t(x)$. If, in addition, the search effort is defined by a cumulative time function $\kappa(x, t) = \int_0^t u(x, \tau) d\tau$ or $\kappa(x, t) = \sum_{\tau=0}^t u(x, \tau)$, then, to obtain the increasing detection probability, it is sufficient, but not necessary, to find a trajectory $a^t : X \rightarrow X$, $a(x, \tau) \in X$, $0 \leq \tau \leq t$, such that for any point $a^t(x)$ in the trajectory the allocation values $w(a^t(x))$ also increase with t . To illustrate these concepts let us consider the following example.

Example 2.2 As indicated above, let the sample space X be a square of size $n = 10 \times 10$, and let the detection probability be defined by a bi-normal distribution with $\sigma_1 = \sigma_2 = 1$. Assume that the search agent starts at point $x^{t=0} = (3, 3)$ and up to time $t = 100$ it follows the trajectory shown in Figure 2.8a. Then, the search density $u^{t=100}$ that is obtained for time $t = 100$ is as shown in Figure 2.8b.

The agent starts at the point $x = (3, 3)$ and then follows the trajectory a^t which includes returns to previous visited points. The search density is accumulated in the visited points and increases at each point according to the number of returns of the search agent.

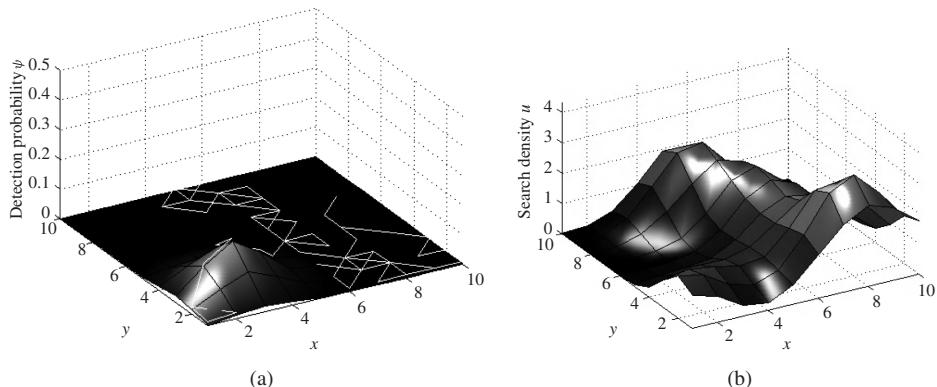


Figure 2.8 Search agent trajectory and search density. (a) Detection probability at starting point $x^{t=0} = (3, 3)$ and a search agent trajectory up to time $t = 100$. (b) Search density $u^{t=100}$ or allocation w at time $t = 100$.

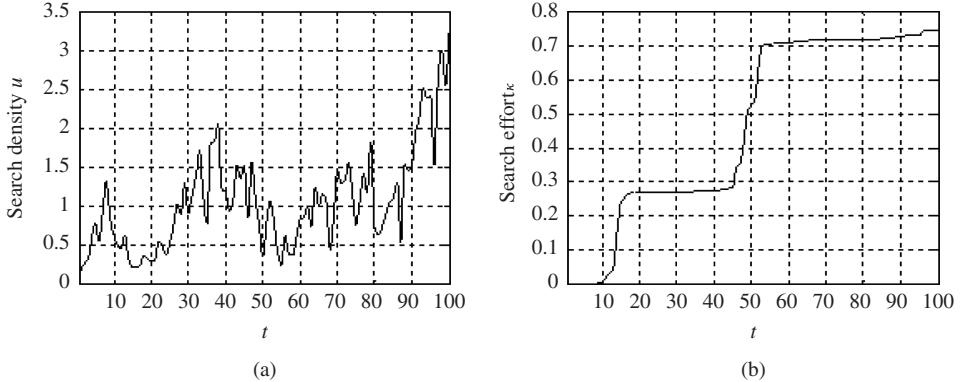


Figure 2.9 The search density along the search trajectory and the search effort at an arbitrary point. (a) Search density along the agent's trajectory. (b) The search effort as applied to point $x = (5, 5)$ as a function of time.

The values of the search density $u^t(a^t(x))$ at points $a^t(x)$ of the trajectory are shown in Figure 2.9a, and a search effort $\kappa(x, 100) = \sum_{\tau=0}^{100} u(x, \tau)$ for an arbitrary chosen point $x = (5, 5)$ is shown in Figure 2.9b.

The search density along the agent's trajectory depends on the trajectory and the detection probabilities as determined by the characteristics of the sensors. As can be seen, the search effort is a non-decreasing function which is defined as a cumulative function of the search density at a certain point and its value depends on the number of returns to that point by the searcher. ■

In general, the dependence of detection probability on the search effort as well as the relation between the search density and the search effort are determined by the conditions of the search tasks. Nevertheless, the indicated requirements on the search density function u^t can be common for different search and screening tasks and are used for distinguishing the class of required trajectories. Specifically, given the detection function φ_κ , the search density u^t is called a *search plan* [7] if for any time t it defines an allocation $w \in W$ and for any point $x \in X$ it increases with t . Consequently, an *optimal search plan* is the one that maximizes the probability $P(t)$ of detecting the target up to time t . A trajectory of the search agent that corresponds to the optimal search plan provides a solution of the search and screening problem.

Below, we describe the Forward and Backward (FAB) algorithm [12, 13], which constructs optimal search plans for a search in discrete time and space. For the implementation of this algorithm, it is assumed that an optimal search plan exists. Stone's theorems [7], described next, guarantee its existence for both continuous and discrete time search.

To formulate and prove Stone's theorems let us first consider the search and screening problem as an optimization problem. Let w be a feasible allocation, and denote by $P[w]$ the probability of detecting the target if the search efforts are distributed according to this allocation w . Given the detection function φ_κ and the target location probabilities $p^t(x)$,

$x \in X$, probability $P[w]$ is defined as follows:

$$\begin{aligned} P[w] &= \int_X^t p(x)\varphi(x, \kappa^t(x))dx \quad \text{for continuous sample space } X, \\ P[w] &= \sum_{x \in X} p^t(x)\varphi(x, \kappa^t(x)) \quad \text{for discrete sample space } X. \end{aligned} \quad (2.3)$$

In addition, let us introduce a *cost function* $c_\kappa : X \rightarrow [0, \infty)$ that specifies the cost of applying the search effort $\kappa(x, t)$ to the point $x \in X$, and denote by $C[w]$ the cost of the search according to allocation w . On the basis of the function c_κ , cost $C[w]$ is defined as follows:

$$\begin{aligned} C[w] &= \int_X c(x, \kappa^t(x))dx \quad \text{for continuous sample space } X, \\ C[w] &= \sum_{x \in X} c(x, \kappa^t(x)) \quad \text{for discrete sample space } X. \end{aligned} \quad (2.4)$$

The *basic search problem* is formulated as follows [7]. Given the set W of all possible allocations and a constant $K < \infty$ that restricts a given search budget, find an allocation $w^* \in W$ such that

$$P[w^*] \geq P[w] \quad \text{and} \quad C[w^*] \leq K \quad \text{for all allocations } w \in W. \quad (2.5)$$

The next proposed arguments guarantee the existence of an allocation w^* that solves the above search problem. Since we are primarily interested in a *discrete time and space* search, the specific theorems by Stone regarding such a search [7] will be presented with complete proofs. Similar statements regarding continuous time and space search can be found in Stone's book [7]; see also the monographs in [6, 8, 15]. The formulations of the statements and their proofs are based on the optimization techniques that implement the Lagrange multiplier method. For suitable information on this technique and illustrative examples see, for example, [16, 17].

In the basic search problem, the cost $C[w]$ is considered as a constraint for the optimization of probability $P[w]$. If for any allocation $w \in W$ the cost $C[w]$ is finite, that is, $C[w] < \infty$, then the basic search problem can be defined as a constrained optimization problem. The corresponding *Lagrange function* (or *Lagrangian*) \mathcal{L} has the form

$$\mathcal{L}[w] = P[w] - \lambda C[w],$$

where $0 \leq \lambda < \infty$ is a real number called the *Lagrange multiplier*. According to the definitions of $P[w]$ and $C[w]$, as given by Equations 2.3 and 2.4, for each point $x \in X$ the Lagrange function ℓ , called the *pointwise Lagrangian*, is as follows:

$$\ell[x, \lambda, \kappa^t(x)] = p^t(x)\varphi(x, \kappa^t(x)) - \lambda c(x, \kappa^t(x)).$$

Then, the Lagrangian $\mathcal{L}[w]$ for an allocation w is defined as the sum of *pointwise Lagrangians*

$$\mathcal{L}[w] = \sum_{x \in X} \ell[x, \lambda, \kappa^t(x)].$$

Let the cost $C[w]$ be finite for any allocation $w \in W$. If for an allocation $w^* \in W$ and a Lagrange multiplier λ , $0 \leq \lambda < \infty$, it follows that

$$\ell[x, \lambda, w^*(x)] \geq \ell[x, \lambda, w(x)], \quad w \in W, x \in X,$$

then one can say that the pair (λ, w^*) maximizes (with respect to the allocations set W) the pointwise Lagrangian ℓ . Similarly, if for the pair (λ, w^*) it follows that

$$\mathcal{L}[w^*] \geq \mathcal{L}[w], \quad w \in W,$$

then one can say that the pair (λ, w^*) maximizes (with respect to W) the Lagrangian \mathcal{L} .

Assume that $C[w] < \infty$ for any allocation $w \in W$ and that for any point $x \in X$ function $p^t(x)\varphi_\kappa(x)$ is *concave* while the cost function $c_\kappa(x)$ is *convex* with respect to the search effort κ^t . Then, the necessary and sufficient conditions for the existence of an optimal allocation over discrete sample space X are given by the following theorem.

Theorem 2.3 [7]. *Let $w^* \in W$ be an allocation such that $C[w^*] \geq C[w]$. Then allocation $w^* \in W$ is optimal if and only if there exists a positive finite Lagrange multiplier $\lambda \in [0, \infty)$ such that the pair (λ, w^*) maximizes the pointwise Lagrangian ℓ .*

Proof. Let us start with the proof of sufficiency. Let (λ, w^*) be a pair that maximizes the pointwise Lagrangian ℓ . Thus,

$$\ell[x, \lambda, w^*(x)] \geq \ell[x, \lambda, w(x)], \quad w \in W, x \in X.$$

Since $C[w]$ is finite for any $w \in W$, summation over X on both sides of the inequality and use of Lagrangian functions result in the following inequality:

$$P[w^*] - \lambda C[w^*] \geq P[w] - \lambda C[w].$$

From this inequality it follows that $P[w^*] - P[w] \geq \lambda(C[w^*] - C[w])$. According to the assumptions, $\lambda > 0$ and $C[w^*] \geq C[w]$. Hence, $\lambda(C[w^*] - C[w]) \geq 0$, from which we obtain $P[w^*] - P[w] \geq 0$ and $P[w^*] \geq P[w]$, which for finite costs $C[w]$ gives the required inequality (2.5).

Now let us prove the necessity of the theorem requirements. Let $w \in W$ be an allocation and let (c, p) be a pair of finite cost $c < \infty$ and probability p such that $c \geq C[w]$ and $p \leq P[w]$ hold. Denote by

$$CP = \{(c, p) : c \geq C[w], p \leq P[w], w \in W\}$$

a set of such pairs over all allocations $w \in W$. Let $w_1, w_2 \in W$ be two allocations and two pairs $(c_1, p_1), (c_2, p_2) \in CP$ such that $c_1 \geq C[w_1]$, $p_1 \leq P[w_1]$ and $c_2 \geq C[w_2]$, $p_2 \leq P[w_2]$. From the convexity of the cost function $c_\kappa(x)$ and concavity of the function $p^t(x)\varphi_\kappa(x)$ it follows that $C[w]$ is convex and $P[w]$ is concave. Then, for any real number

$\theta \in [0, 1]$ it follows that $\theta w_1 + (1 - \theta)w_2 = w \in W$, and

$$\theta c_1 + (1 - \theta)c_2 \geq \theta C[w_1] + (1 - \theta)C[w_2] \geq C[w],$$

$$\theta p_1 + (1 - \theta)p_2 \geq \theta P[w_1] + (1 - \theta)P[w_2] \geq P[w].$$

Thus, $\theta(c_1, p_1) + (1 - \theta)(c_2, p_2) = (c, p) \in CP$ and the set CP is convex.

Let $CP^* = \{(c, p) : c \geq C[w^*], p \leq P[w^*], w^* \in W\}$ be a set of pairs (c, p) for an optimal allocation w^* . The set CP^* is convex and from the optimality of w^* it follows that $CP \cap CP^*$ lies on the boundary of CP . Hence, according to the separation theorems (see, e.g., [18, 19]; for implementation examples see, e.g., [16, 17]), there is a line that separates CP and CP^* and is defined by two values λ^p , $\lambda \in [0, 1]$, such that $\lambda^p P[w^*] - \lambda C[w^*] \geq \lambda^p p - \lambda c$. Thus, from the definition of the cost c and the probability p , it follows that

$$\lambda^p P[w^*] - \lambda C[w^*] \geq \lambda^p P[w] - \lambda C[w].$$

If $\lambda = 0$ then for any $\lambda^p \in [0, 1]$ the inequality holds, while if $\lambda > 0$ then from the assumption $C[w^*] \geq C[w]$ it follows that $\lambda^p > 0$. Hence, by specifying $\lambda^p = 1$ we obtain the required inequality (2.5). ■

The proof of Theorem 2.3 is presented for a case of two-dimensional optimization, where both cost c and probability p are real numbers. In general, the same consideration can be used for the case of vector optimization, in which cost c is represented by a multi-dimensional vector. Such a consideration and its generalizations, as well as a similar theorem for the search in continuous space and time, can be found in Stone [7].

This theorem guarantees the existence of optimal allocation for each time. To illustrate an application of the theorem, let us consider a search in which the detection function φ_κ is defined by the Koopman random-search equation (Equation 2.1) and the cost term c_κ is equal to the search effort κ^t . Such assumptions were implemented by Koopman [1] for a continuous time and continuous space search; here we use these assumptions for the search in discrete time and discrete space.

Let $X = \{x_1, x_2, \dots, x_n\}$ be a sample space with definite target location probabilities $p^t(x)$, $x \in X$, $t = 0, 1, 2, \dots$. Since the cost is equal to the search effort, that is, $c(x, \kappa^t(x)) = \kappa(x, t)$, and $\varphi(x, \kappa) = 1 - e^{-\kappa(x,t)}$, the pointwise Lagrangian ℓ takes the following form:

$$\ell[x, \lambda, \kappa^t(x)] = p^t(x)\varphi(x, \kappa^t(x)) - \lambda c(x, \kappa^t(x)) = p^t(x)(1 - e^{-\kappa(x,t)}) - \lambda\kappa(x, t).$$

From the requirements $d\ell/d\kappa = p^t(x)e^{-\kappa(x,t)} - \lambda = 0$ and $d^2\ell/d\kappa^2 = -p^t(x)e^{-\kappa(x,t)} < 0$ it follows that the Lagrangian ℓ reaches its maximum for the search effort $\kappa^t(x) = \ln(p^t(x)/\lambda)$, $\lambda > 0$. According to the assumption on the equivalence of the cost and the search effort, it follows that $c(x, \kappa^t(x)) = \ln(p^t(x)/\lambda)$. In addition, since $w(x) = \kappa^t(x)$ at each point $x \in X$, we have $w(x) = \ln(p^t(x)/\lambda)$. From the definition (Equation 2.3) of the probability of detecting the target using allocation w , it follows that for the obtained search effort

$$P[w] = \sum_{x \in X} p^t(x) \left(1 - \frac{\lambda}{p^t(x)}\right).$$

Hence,

$$\left(1 - \frac{\lambda}{p^t(x)}\right) \geq 0 \quad \text{and} \quad \frac{\lambda}{p^t(x)} \leq 1, \quad x \in X.$$

This results in the requirement that $w(x) = \ln(p^t(x)/\lambda) \geq 0$ for each point $x \in X$ and $\lambda > 0$. Since for the obtained search effort Lagrangian ℓ reaches its maximum, the allocation w which meets the obtained requirement is optimal. In other words, for optimal allocation w^* it follows that $w^*(x) = \ln(p^t(x)/\lambda)$ while $\ln(p^t(x)/\lambda) \geq 0$ and $w^*(x) = 0$ otherwise.

Let us now find a value λ^* of the multiplier λ that corresponds to optimal allocation w^* . Since

$$P[w^*] = \sum_{x \in X} p^t(x) \left(1 - \frac{\lambda^*}{p^t(x)}\right) = \sum_{x \in X} p^t(x) - \sum_{x \in X} \lambda^* = 1 - n\lambda^* \leq 1,$$

the value λ^* is bounded as follows: $0 \leq \lambda^* \leq 1/n$.

Recall that, by the assumptions, $w(x) = \kappa^t(x) = c(x, \kappa^t(x))$; thus $C[w^*] = \sum_{x \in X} w^*(x)$. Let $K \in [0, \infty)$ be a value of the available search effort. Then $C[w^*] = K$ and

$$\sum_{x \in X} w^*(x) = \sum_{x \in X} \ln\left(\frac{p^t(x)}{\lambda^*}\right) = \sum_{x \in X} \ln(p^t(x)) - n \ln(\lambda) = K.$$

Hence, for optimal allocation w^* multiplier λ has the value

$$\lambda^* = \exp\left(\frac{\sum_{x \in X} \ln(p^t(x)) - K}{n}\right).$$

Indeed, for all values of the probabilities and available effort it follows that $\lambda^* > 0$, while from the previously obtained bound $\lambda^* \leq 1/n$ it follows that $K \geq \sum_{x \in X} \ln(p^t(x)) - n \ln(n)$; thus the obtained value λ^* meets the requirements provided by the probability $P[w^*]$.

Finally, by substitution of the obtained value λ^* into the equation that defines optimal allocation w^* , it follows that for any point $x \in X$

$$w^*(x) = \begin{cases} \ln(p^t(x)) - \frac{\sum_{x \in X} \ln(p^t(x)) - K}{n} \\ \text{if } \ln(p^t(x)) - \frac{\sum_{x \in X} \ln(p^t(x)) - K}{n} \geq 0 \\ \text{otherwise.} \end{cases} \quad (2.6)$$

For the case of equivalent location probabilities $p^t(x_1) = p^t(x_2) = \dots = p^t(x_n) = 1/n$, optimal allocation w^* has intuitively clear form $w^*(x) = \ln\left(\frac{1}{n}\right) - \frac{n \ln(1/n) - K}{n} = \frac{K}{n}$ and requires distribution of the available search effort K in equal proportions over all points $x \in X$ of the sample space.

Implementation of the obtained optimal allocation is given by the following example.

Example 2.3 Assume that, similar to Example 2.1, the target location probabilities $p^{t=0}(x)$, $x \in X$, at time $t = 0$ are defined by a combination of two bi-normal distributions, as is shown in Figure 2.4a. Then, the optimal allocation w^* specifies the search density

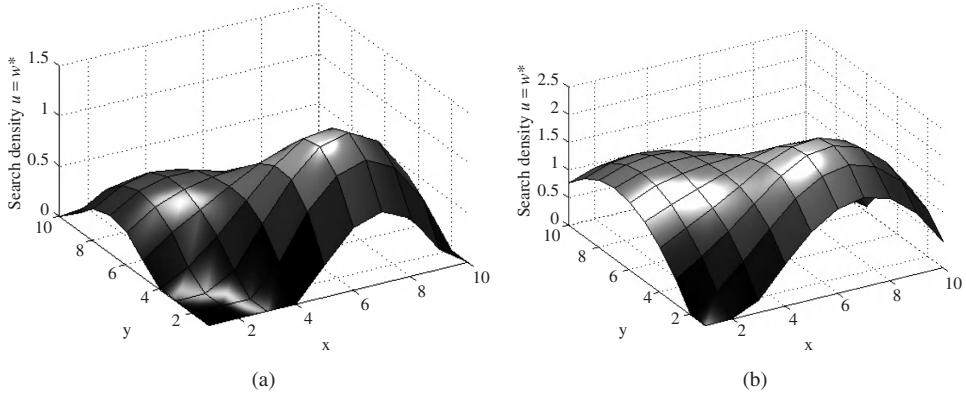


Figure 2.10 Search density defined by an optimal allocation of efforts according to Equation 2.6. (a) Search density $u^{t=0} = w^*$; available search effort $K = 10$. (b) Search density $u^{t=0} = w^*$; available search effort $K = 100$.

$u^{t=0}$ for time $t = 0$ and is calculated by Equation 2.6. Examples of the search density $u^{t=0} = w^*$ for two different efforts $K = 10$ and $K = 100$ are shown in Figure 2.10.

The probabilities $P[w^*]$ of detecting the target according to the search densities (allocations), which are shown in Figure 2.10a, b, are $P[w^*] = 0.993$ and $P[w^*] = 0.997$, respectively. In comparison, if the search is governed by the search densities that correspond to the detection probabilities shown in Figures 2.5a and 2.6a, then the probabilities $P(0)$ of detecting the target at the initial step $t = 0$ are $P(0) = 0.027$ and $P(0) = 0.054$, respectively. ■

The consideration above addresses an optimal allocation w^* that gives a search density u^t for any fixed time t and a constant available effort K . An implementation of similar considerations over the sequence of times $t = 0, 1, 2, \dots$ results in an optimal search plan that solves the search and screening problem, obtaining the search density. When considering such a dynamical problem, the constant search effort K has to be substituted by the applied search effort $K(t) \in [0, \infty)$, which depends on the searcher's velocity and sensor abilities and increases with time. In the simplest case, such an applied effort can be defined as $K(t) = K_0 + |A|t$, where K_0 is the initial effort at $t = 0$ and $|A|$ specifies the size of the area A that is covered by the sensor in one time step given the agent's velocity. If, for example, the agent moves with velocity \vec{V} [7], while its sensor is defined by a definite range law (see Figure 2.3) and covers an observed area with radius r , then $|A| = 2r|\vec{v}|$ (see [20]). An illustration of changing the search density in time is provided by the following example.

Example 2.4 Let the target location probabilities $p^{t=0}(x)$, $x \in X$, at time $t = 0$ be defined as in Examples 2.1 and 2.3, and, similar to Example 2.1, let the target be governed by a diffusion-type Markov process. Moreover, let the applied search effort be $K(0) = K(1) = 10$ and $K(t) = k \times t$ with constant $k = 10$, and assume that the considered search follows Equation 2.6. Then, search density u^t and target location density v^t for times $t = 1$, $t = 10$, and $t = 100$ are as shown in Figure 2.11.

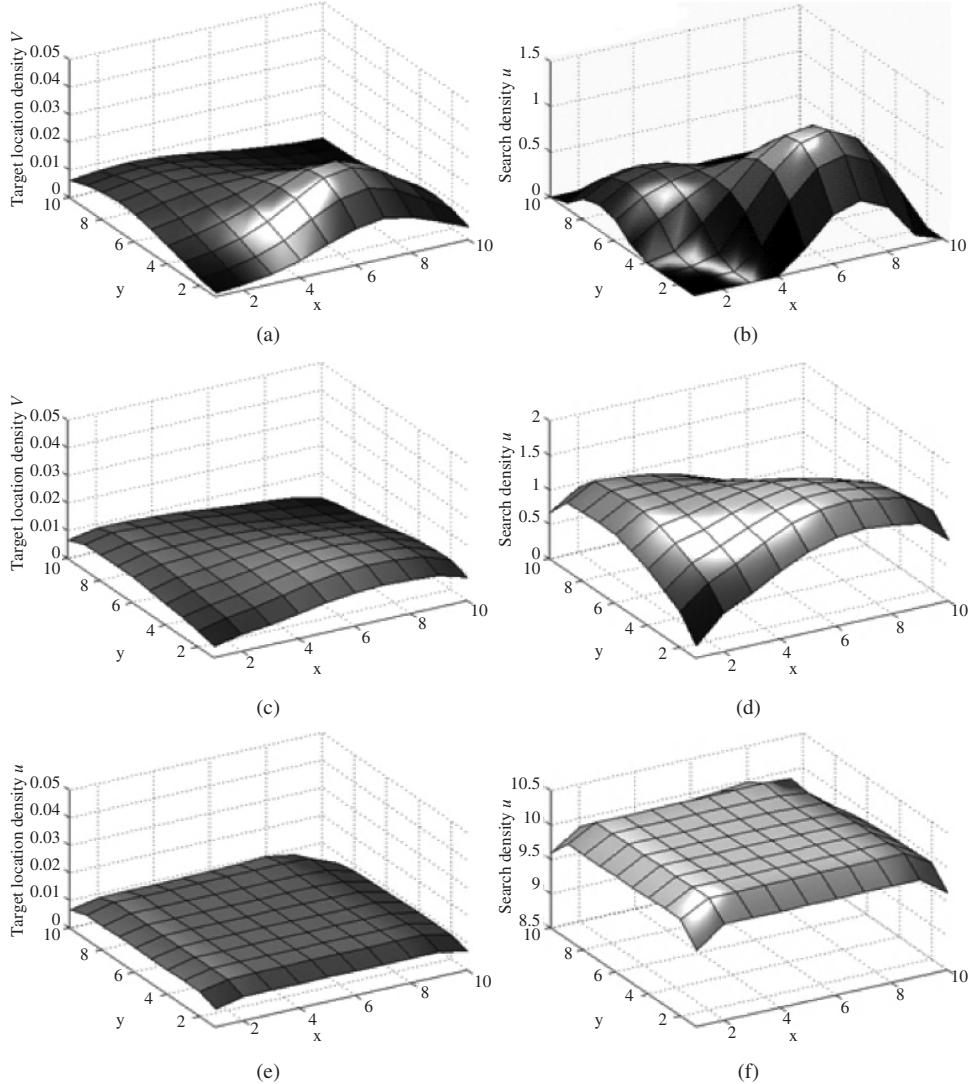


Figure 2.11 Dynamics of a Markovian target location density and the corresponding search density for an optimal allocation following Equation 2.6. (a) Target location density $v^{t=0}$. (b) Search density $u^{t=0}$; available search effort $K = 10$. (c) Target location density $v^{t=10}$. (d) Search density $u^{t=10}$; available search effort $K = 100$. (e) Target location density $v^{t=100}$. (f) Search density $u^{t=100}$; available search effort $K = 1000$.

The corresponding detection probabilities of detecting the target at times $t = 1$, $t = 10$, and $t = 100$ are $P[u^{t=0}] = 0.992$, $P[u^{t=10}] = 0.997$, and $P[u^{t=100}] = 1.0$, respectively.

Note that with respect to different target location densities (see Figures 2.10a and 2.11c), the allocation shown in Figure 2.10b differs from the one shown in Figure 2.11d, in spite of the equal efforts with $K = 100$. ■

In certain search problems, especially for the search in discrete time and space, the necessary and sufficient conditions of optimality can be presented in a constructive form, such that it results in the corresponding search algorithms. First, let us consider the algorithm that provides an optimal allocation w^* at each time t under the requirements of Example 2.3. This algorithm was obtained by Stone [7] within the framework of search and screening theory.

Let us fix a time t and assume that over the sample space $X = \{x_1, x_2, \dots, x_n\}$ the location probabilities $p^t(x)$, $x \in X$, are well defined. Assume, in addition, that the detection function φ_κ is defined by the Koopman random-search equation (Equation 2.1) with the search effort $\kappa^t(x_i) = \sigma_i k$, where $\sigma_i > 0$, $i = 1, \dots, n$, and $k \geq 0$ is a value given for each point $x \in X$ according to the location probability $p(x)$ as presented below. Note that, since the probabilities are fixed and do not depend on t , we omit the index t in the notation of the probabilities and of k . Then, the Koopman equation (Equation 2.1) takes the following form:

$$\varphi(x_i, k) = 1 - e^{-\sigma_i k}, \quad i = 1, \dots, n.$$

Let us now find the values of k such that the corresponding allocation w is optimal according to Theorem 2.3, that is, such that the Lagrangian $\ell[x, \lambda, k] = p(x)\varphi(x, k) - \lambda c(x, k)$ reaches its maximum given some cost function c_κ .

Assume that the points x_i are indexed in such an order that, given the weighting coefficients $\sigma_i > 0$, it follows that $\sigma_1 p(x_1) \geq \sigma_2 p(x_2) \geq \dots \geq \sigma_n p(x_n)$, and denote

$$h_i = \ln(\sigma_i p(x_i)) - \ln(\sigma_{i+1} p(x_{i+1})), \quad i = 1, \dots, n-1.$$

Let $k_{11} = h_1/\sigma_1$ be the value of the search effort that is applied to point x_1 . Then the pointwise Lagrangian $\ell[x, \lambda, k]$ at point x_1 is

$$\ell[x_1, \lambda, k_{11}] = p(x_1)(1 - e^{-\sigma_1 k_{11}}) - \lambda c(x_1, k_{11}),$$

and we need to find λ and $c(x_1, k)$ such that for $k = k_{11}$ it follows that

$$\frac{d}{dk} \ell[x_1, \lambda, k]|_{k=k_{11}} = p(x_1)\sigma_1 e^{-\sigma_1 k_{11}} - \frac{d}{dk} \lambda c(x_1, k)|_{k=k_{11}} = 0.$$

Since for the defined search effort $\sigma_1 e^{-\sigma_1 k_{11}} = \sigma_1 e^{-h_1} = \sigma_2 p(x_2)/p(x_1)$, from the last equivalence it follows that

$$\frac{d}{dk} \lambda c(x_1, k)|_{k=k_{11}} = \sigma_2 p(x_2),$$

which can be obtained by specifying $\lambda = \sigma_2 p(x_2)$ and $c(x_1, k) = k$. Hence, the optimal allocation w^* with respect to the cost $C[w^*] = k_{11}$ is defined by $w^*(x_1) = k_{11}$ and $w^*(x_i) = 0$, $i = 2, \dots, n$. This case corresponds to a search in which the available search effort is allocated to the first point, while an observation's negative result implies that the target is located at one of the other points.

Now let us consider point x_2 with search effort $k_{22} = h_2/\sigma_2$. Then, by similar considerations of the pointwise Lagrangian $\ell[x_2, \lambda, k_{22}]$ as above, this results in $\lambda = \sigma_3 p(x_3)$ and $c(x_2, k) = \text{const}$, and we can specify $c(x_2, k) = k_{22}$.

Let us apply these values to the Lagrangian $\ell[x, \lambda, k]$ at point x_1 . We need to define a search effort k such that the equivalence equation holds:

$$\frac{d}{dk} \ell[x_1, \lambda, k] = p(x_1) \sigma_1 e^{-\sigma_1 k} - \sigma_3 p(x_3) = 0.$$

Denote by $k_{12} = h_2/\sigma_1$ the joint search effort and assume that $k = k_{11} + k_{12} = (h_1 + h_2)/\sigma_1$. Then,

$$p(x_1) \sigma_1 e^{-\sigma_1 k} = p(x_1) \sigma_1 e^{-h_1} e^{-h_2} = p(x_1) \sigma_1 \frac{\sigma_2 p(x_2)}{\sigma_1 p(x_1)} \times \frac{\sigma_3 p(x_3)}{\sigma_2 p(x_2)} = \sigma_3 p(x_3),$$

and the equivalence $d\ell[x_1, \lambda, k]/dk = 0$ is satisfied. Hence, the search effort that is applied to point x_1 by the optimal allocation is $w^*(x_1) = k_{11} + k_{12}$, while that applied to point x_2 is $w^*(x_2) = k_{22}$, while $w^*(x_i) = 0$, $i = 3, \dots, n$. The cost for this allocation w^* is $C[w^*] = k_{11} + k_{12} + k_{22}$.

The same result follows from an allocation $w(x_1) = k_{11} + \varepsilon k_{12}$, $w(x_2) = \varepsilon k_{22}$, $w(x_i) = 0$, $i = 3, \dots, n$, where $0 \leq \varepsilon \leq 1$, that is optimal with respect to the cost $C[w] = k_{11} + \varepsilon k_{12} + \varepsilon k_{22}$.

Similar consideration for the allocations over j points, $1 \leq j \leq n - 1$, results in the following definition of the allocation:

$$w(x_i) = \frac{1}{\sigma_i} \sum_{i=1}^{j-1} h_i + \varepsilon h_j \quad \text{for } 1 \leq i \leq j \quad \text{and} \quad w(x_i) = 0 \quad \text{for } j < i \leq n. \quad (2.7)$$

Let us find the value of ε such that the allocation w is optimal with respect to a given cost value K . Denote by $K(j) = \sum_{i=1}^j (1/\sigma_i)(h_i + h_{i+1} + \dots + h_j)$ partial sums of the search efforts. For example, for the considered points x_1 and x_2 this gives

$$K(1) = \frac{1}{\sigma_1} h_1 = k_{11} \quad \text{and} \quad K(2) = \frac{1}{\sigma_1} (h_1 + h_2) + \frac{1}{\sigma_2} h_2 = k_{12} + k_{22}.$$

In addition, assume that $K(0) = 0$. Then, for

$$\varepsilon = (K - K(j-1))/(K(j) - K(j-1)),$$

where j is an index such that $K(j-1) \leq K \leq K(j)$, an allocation (Equation 2.7) is optimal with respect to the cost K . If for the cost K it follows that $K \geq K(n-1)$, that is, there are enough efforts for distribution over all sample space, then the value of ε is defined by

$$\varepsilon = (K - K(j-1)) / \sum_{i=1}^n \frac{1}{\sigma_i}.$$

The consideration [7] above forms a basis for the algorithm that generates an optimal allocation. The outline of the algorithm is as follows.

Algorithm 2.1 (Stone's algorithm) [7]. Given location probabilities $p(x_i)$, weighting coefficients $\sigma_i > 0$, $i = 1, \dots, n$, and available cost K :

1. For all $i = 1, \dots, n$ do: set $k_i = 0$.
2. Set $K(0) = 0$.

3. For each $i = 1, \dots, n - 1$ do:
 - 3.1. Set $h_i = \ln(\sigma_i p(x_i)) - \ln(\sigma_{i+1} p(x_{i+1}))$.
 - 3.2. Set $K(i) = K(i - 1) + h_i \left(\frac{1}{\sigma_1} + \dots + \frac{1}{\sigma_i} \right)$.
 - 3.3. If $K(i) \geq K$ then: Break.
 - 3.4. For all $j = 1, \dots, i$ do: set $k_j = k_j + \frac{h_i}{\sigma_j}$.
4. If $K(i) \geq K$ then:
 - 4.1. Set $\varepsilon = (K - K(i - 1))/(K(i) - K(i - 1))$.
Else:
 - 4.2. Set $\varepsilon = (K - K(i - 1))/\sum_{j=1}^n \frac{1}{\sigma_j}$.
 - 4.3. Set $h_n = 1$.
 5. For all $j = 1, \dots, i$ do: set $w(x_j) = k_j + \varepsilon \frac{h_i}{\sigma_j}$.
 6. For all $j = i + 1, \dots, n$ do: set $w(x_j) = 0$.
 7. Return w .

■

The algorithm results in an optimal allocation w given a cost K that specifies the overall available search efforts. The weighting coefficients $\sigma_i > 0$ determine the search conditions at points x_i , $i = 1, \dots, n$. The actions of the algorithm are illustrated by the following example.

Example 2.5 Let the target location probabilities $p^{t=0}(x)$, $x \in X$, at time $t = 0$ be equal to the ones used previously in Examples 2.1, 2.3, and 2.4 (see Figures 2.4a and 2.11a). Given the cost K that specifies the available search effort, Algorithm 2.1 results in an optimal allocation w^* that corresponds to the search density $u^{t=0}$. Examples of the search density $u^{t=0} = w^*$ for two different search efforts $K = 10$ and $K = 100$ and unit weights $\sigma_1 = \sigma_{10} = \dots = \sigma_{100}$ are shown in Figure 2.12.

Note that, if the available search effort is relatively small, then, from Figure 2.12a, Algorithm 2.1 allocates most of the effort to the region with the maximal target location probability. This allocation is different from the one obtained by Equation 2.6 as seen in Figure 2.10a. However, if there is enough search effort for distribution over the domain, then, as shown in Figure 2.12b, the allocation obtained by Algorithm 2.1 is similar to that obtained by Equation 2.6, as seen in Figure 2.10b. ■

As follows from Examples 2.3 and 2.5, the allocation depends on the assumption regarding the search effort. If, as assumed in Equation 2.6, the search effort is infinitely divisible among the points of the domain, then for any available search effort it is distributed over all points of the domain proportionally to the target location probabilities. If, as implemented in Algorithm 2.1, the search effort is distributed sequentially over the points and cannot be infinitely divided, then for small efforts the search follows a greedy policy.

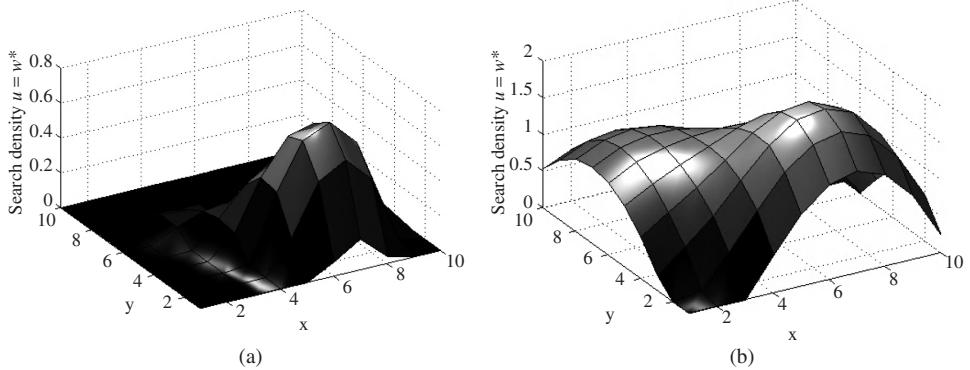


Figure 2.12 A search density that corresponds to an optimal allocation as obtained by Algorithm 2.1. (a) Search density $u^{t=0} = w^*$; available search effort $K = 10$. (b) Search density $u^{t=0} = w^*$; available search effort $K = 100$.

As indicated above, Stone's algorithm results in the optimal allocation for given target location probabilities that do not change over time. For changing location probabilities that correspond, for example, to search for a moving target, the algorithm can be applied for each probability distribution that is obtained by relying on the target's movement rule, as will be discussed later.

On the basis of Stone's Algorithm 2.1, Brown [12] suggested an algorithm of search for a Markovian target that applies the same assumptions regarding the detection function. Brown's algorithm results in an optimal search plan that, in contrast to the static target search, is unique with respect to the detection function as defined by the Koopman random-search equation (Equation 2.1). The algorithm follows a literal formulation of the above search-planning problem that, in the considered case, is formalized in the form of a nonlinear programming minimization problem, while its solution is characterized by using the Kuhn–Tucker necessary and sufficient condition [12].

Let us formulate the Kuhn–Tucker conditions for the general minimization problem with non-negative variables [16, 17]. Let $\vec{\kappa} = (\kappa_1, \kappa_2, \dots, \kappa_n)$ be a vector in an n -dimensional Euclidean space \mathcal{E}^n , while $f : \mathcal{E}^n \rightarrow \mathbb{R}$ and $g_i : \mathcal{E}^n \rightarrow \mathbb{R}$, $i = 1, 2, \dots, m$, are real-valued functions. In addition, let $\mathcal{K}_i \in (-\infty, \infty)$, $i = 1, 2, \dots, m$, be some real numbers. In the minimization problem, it is required to find a vector $\vec{\kappa}$ such that

$$f(\kappa_1, \kappa_2, \dots, \kappa_n) \rightarrow \min, \quad (2.8)$$

$$\text{subject to } \kappa_i \geq 0, \text{ and } g_i(\kappa_1, \kappa_2, \dots, \kappa_n) \leq \mathcal{K}_i, \quad i = 1, 2, \dots, m. \quad (2.9)$$

Vector $\vec{\kappa}$ that satisfies the problem requirements is called the *optimal solution* of the problem.

The Kuhn–Tucker necessary and sufficient conditions regarding the optimal solution of the minimization (Equations 2.8 and 2.9) are formulated as follows. Let $\vec{\kappa}$ be an optimal solution of the minimization problem (Equations 2.8 and 2.9). Then, there exist some multipliers $\lambda_i \geq 0$, $i = 1, 2, \dots, m$, and $\mu_j \geq 0$, $j = 1, 2, \dots, n$, such that

$$\lambda_i(\mathcal{K}_i - g_i(\kappa_1, \kappa_2, \dots, \kappa_n)) = 0, \quad i = 1, 2, \dots, m, \quad (2.10)$$

and for each $j = 1, 2, \dots, n$ it follows that

$$\frac{\partial \phi}{\partial h_j} + \sum_{i=1}^m \lambda_i \frac{\partial g_i}{\partial h_j} - \mu_j = 0, \left(\frac{\partial \phi}{\partial h_j} + \sum_{i=1}^m \lambda_i \frac{\partial g_i}{\partial h_j} \right) h_j = 0. \quad (2.11)$$

Based on the same conventions, let function ϕ and functions g_i , $i = 1, 2, \dots, m$, be convex. Then, if for vector \vec{h} the requirements (Equations 2.48 and 2.11) do hold, then this vector is an optimal solution of the minimization problem (Equation 2.8).

Now let us consider Brown's algorithm [12], which builds an optimal solution of the search-planning problem. Let $\vec{o}(t) = \langle x^{t=0}, \dots, x^{t-2}, x^{t-1}, x^t \rangle$ be a target trajectory up to time t . Trajectory $\vec{o}(t)$ consists of the points of the sample space $X = \{x_1, x_2, \dots, x_n\}$ as chosen by the target during its movement over the sample space. Under the assumption that the target's movement is governed by a stochastic process, for any trajectory $\vec{o}(t)$ a probability $p(\vec{o}(t)) \in [0, 1]$ can be determined such that the target chooses the trajectory $\vec{o}(t)$, that is,

$$p(\vec{o}(t)) = \Pr\{\text{target follows trajectory } \vec{o}(t)\}.$$

Assume that detection function φ_κ is defined according to the Koopman random-search formulation (Equation 2.1), with the search effort function κ^t , such that for any point $x \in X$ and any time $t = 0, 1, 2, \dots$ the search effort $\kappa(x, t) = \sigma(x, t)k(x, t)$, where $\sigma(x, t) > 0$, $x \in X$, $t = 0, 1, 2, \dots$, are weighting coefficients, and $k(x, t) \geq 0$ are the values that, as indicated above, are equal to the cost of observing point $x \in X$ at time t . The Koopman equation for such a search effort takes the following form:

$$\varphi(x_i, k) = 1 - e^{-\sigma(x_i, t)k(x_i, t)}, \quad i = 1, \dots, n, \quad t = 0, 1, 2, \dots$$

This definition of the detection probability function φ is similar to the definition which was implemented in Algorithm 2.1, with an additional assumption that both the weighting coefficients $\sigma(x, t)$ and the values $k(x, t)$ depend on time. For this detection probability function, the probability of not detecting the target at time t is

$$1 - \varphi(x_i, k) = e^{-\sigma(x_i, t)k(x_i, t)}, \quad i = 1, \dots, n, \quad t = 0, 1, 2, \dots$$

Assume that the search is Markovian (see Section 2.1.2), that is, the result of the observation at time t does not depend on the observation result at previous times. In other words, assume that for the event $D^t = \{\text{target is detected at time } t\}$ and for the opposite event $\bar{D}^t = \{\text{target is not detected at time } t\}$, $t = 0, 1, 2, \dots$, it follows that

$$\Pr\{D^t | x^t = x_i, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} = \Pr\{D^t | x^t = x_i\},$$

$$\Pr\{\bar{D}^t | x^t = x_i, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\} = \Pr\{\bar{D}^t | x^t = x_i\}.$$

Finally, as is usual in search-planning problems, assume that the search effort is cumulative over time (see Section 2.1.3). In the considered case, this implies that $\kappa(x_i, t) = \sum_{\tau=0}^t u(x_i, \tau) = \sum_{\tau=0}^t \sigma(x_i, \tau)k(x_i, \tau)$, $i = 1, \dots, n$, where $u(x_i, \tau)$ is the value of the search density function $u^t : X \rightarrow [0, \infty)$ at point x_i at time τ .

Let $K(t) \in [0, \infty)$ be the search effort that is available to the search agent at time t and that can be divided over the points at this time. Note that in previous considerations the

effort $K(t)$ was specified relatively to the agent's abilities, while according to Algorithm 2.1 it was defined as a function of location $K(j) \equiv K(x_j)$, $j = 1, \dots, n$, and the location probabilities. In Brown's algorithm, similar to the search in continuous space and continuous time [8, 21], the effort $K(t)$ is an arbitrary positive real number that is often defined according to the requirements of a specific task.

Denote by $\mathcal{O} = \{\vec{o} \in X^{(t)} : p(\vec{o}(t)) > 0\}$ the set of all possible trajectories of the target up to time t , where $X^{(t)}$ stands for a Cartesian product of t copies of X , and let $\mathbf{k}^t = \|k(x_i, \tau)\|_{n \times t}$ be an $n \times t$ matrix of the non-weighted efforts $k(x_i, \tau)$, $x_i \in X$, $i = 1, \dots, n$, $\tau = 0, 1, 2, \dots, t$, that were applied during the search up to time t to the points of the sample space. Then, under these assumptions, the probability $Q(\mathbf{k}^t)$ of not detecting the target up to time t by using the efforts $k(x_i, \tau)$ from matrix \mathbf{k}^t is defined as follows:

$$Q(\mathbf{k}^t) = \sum_{\vec{o} \in \mathcal{O}} p(\vec{o}) \exp\left(-\sum_{\tau=0}^t \sigma(x^\tau, \tau) k(x^\tau, \tau)\right),$$

where $x^\tau \in \vec{o}(t)$, $\tau = 0, 1, 2, \dots, t$, is a point on the trajectory $\vec{o}(t)$.

Hence, the search-planning problem is formulated as an optimization problem that requires minimization of the probability $Q(\mathbf{k}^t)$ of not detecting the target under the above constraints. That is, it is required to find a matrix \mathbf{k}^t such that

$$Q(\mathbf{k}^t) \rightarrow \min, \quad (2.12)$$

$$\text{subject to: } k(x_i, \tau) \geq 0 \text{ for any } x_i \in X, \text{ and } \sum_{i=1}^n k(x_i, \tau) \leq K(\tau). \quad (2.13)$$

As indicated above, by using both the specified weighting coefficients $\sigma(x_i, \tau)$ and the $k(x_i, \tau)$ values from the matrix \mathbf{k}^t , one can solve the optimization problem (2.12–2.13) and define an optimal search density function u^t up to time t . Recall also that the search density u^τ at fixed time $\tau = 0, 1, 2, \dots, t$ results in the allocation w . However, note that, in contrast to Algorithm 2.1, the allocation w at time τ , as defined by the solution of the problem (2.12–2.13), is not necessarily optimal.

Let us consider the solution of the optimization problem (2.12–2.13). Since with the above assumptions function Q is convex [12], this problem is similar to Equations 2.8 and 2.9, and the Kuhn–Tucker optimality conditions (Equations 2.10 and 2.11) can be applied.

Lemma 2.1 [12]. *A matrix \mathbf{k}^t is an optimal solution of the minimization problem (2.12–2.13) if and only if there exist values $\mu^\tau \geq 0$, $\tau = 0, 1, 2, \dots, t$, such that for all $i = 1, \dots, n$ it follows that*

$$\begin{aligned} \frac{\partial}{\partial k(x_i, \tau)} Q(\mathbf{k}^t) &= \mu^\tau \text{ if } k(x_i, \tau) > 0, \\ \frac{\partial}{\partial k(x_i, \tau)} Q(\mathbf{k}^t) &\geq \mu^\tau \text{ if } k(x_i, \tau) = 0, \end{aligned}$$

and for each $\tau = 0, 1, 2, \dots, t$ it follows that

$$k(x_i, \tau) \geq 0 \quad \text{and} \quad \sum_{i=1}^n k(x_i, \tau) \leq K(\tau). \quad (2.14)$$

Proof. Consider the Kuhn–Tucker equations (Equations 2.10 and 2.11) for the problem (2.12–2.13). Note that functions $g^\tau(k^t) = \sum_{i=1}^n k(x_i, \tau)$ are increasing and linear over the values $k(x_i, \tau)$; hence, for each $i = 1, \dots, n$ it follows that $\sum_{\tau=1}^t \lambda^\tau \frac{\partial g^\tau(k^t)}{\partial k(x_i, \tau)} = -\lambda_i^\tau$ with $\lambda_i^\tau \geq 0$. By use of the values λ_i^τ , the Kuhn–Tucker equations (Equations 2.10 and 2.11) for the considered problem obtain the following form ($i = 1, 2, \dots, n$, $\tau = 0, 1, 2, \dots, t$):

$$\lambda_i^\tau \left(K(\tau) - \sum_{j=1}^n k(x_j, \tau) \right) = 0, \quad (2.15)$$

$$\left(\frac{\partial}{\partial k(x_i, \tau)} Q(k^t) - \lambda_i^\tau \right) k(x_i, \tau) = 0, \quad (2.16)$$

$$\frac{\partial}{\partial k(x_i, \tau)} Q(k^t) = \lambda_i^\tau + \mu^\tau. \quad (2.17)$$

Let $k(x_i, \tau) > 0$, $i = 1, 2, \dots, n$. Then, if in the requirement of Equation 2.14 it follows that $\sum_{i=1}^n k(x_i, \tau) = K(\tau)$, then Equation 2.15 holds for any $\lambda_i^\tau \geq 0$, while if $\sum_{i=1}^n k(x_i, \tau) < K(\tau)$, this condition holds only if $\lambda_i^\tau = 0$. Let us now consider Equation 2.16. Recall the weight $\sigma(x_i, \tau) > 0$ for any $i = 1, 2, \dots, n$ and $\tau = 0, 1, 2, \dots, t$. Then, according to the definition of the function Q , its partial derivative with respect to $k(x_i, \tau)$ is $\partial Q(k^t)/\partial k(x_i, \tau) = -\sigma(x_i, \tau)Q(k^t)$. So, Equation 2.16 obtains the form $(-\sigma(x_i, \tau)Q(k^t) - \lambda_i^\tau)k(x_i, \tau) = 0$, which holds if $Q(k^t) = 0$ and $\lambda_i^\tau = 0$. Hence, from Equation 2.17 we obtain the first requirement of the lemma. Note that in this case $\mu^\tau = 0$, as well.

Now, let $k(x_i, \tau) = 0$, $i = 1, 2, \dots, n$. In this case, Equation 2.16 holds for any $\lambda_i^\tau \geq 0$, while Equation 2.15 with $K(\tau) = 0$ is true for any $\lambda_i^\tau \geq 0$, but if $K(\tau) > 0$, then it requires $\lambda_i^\tau = 0$. Hence $\lambda_i^\tau \geq 0$ and from Equation 2.17 we obtain the second requirement of the lemma. ■

This lemma specifies the necessary and sufficient conditions for the optimality of the search density u^t . As indicated above, allocations that are specified by density u^t are not necessarily optimal for intermediate times τ . However, under the implemented assumptions, the optimality of the search density u^t is satisfied by the optimality of the search densities at times $\tau = 0, 1, 2, \dots, t$. To obtain the algorithm that generates such densities, let us specify the transformation of the allocation that corresponds to the search density u^{t-1} at time $\tau - 1$ into the allocation that is defined by the search density u^τ at time τ . Such a transformation is called *reallocation* and the problem of finding optimal components of the matrix k^t at time $\tau = 0, 1, 2, \dots, t$ is called the *reallocation problem*. Formally, this problem is defined as follows [12].

Let, as shown above, $k^t = ||k(x_i, \tau)||_{n \times t}$ be an $n \times t$ matrix of non-weighted efforts $k(x_i, \tau)$, $x_i \in X$, $i = 1, \dots, n$, $\tau = 0, 1, 2, \dots, t$, and let $\overrightarrow{k} = ||k'(x_i)||_{n \times 1}$ be a vector that specifies non-weighted efforts to the points $x_i \in X$ of the sample space X . Define a function *reallocate* (k^t , \overrightarrow{k} , τ) that for a given τ changes the values $k(x_i, \tau)$ of the matrix k^t to the values $k'(x_i)$ defined by the vector \overrightarrow{k} , $i = 1, \dots, n$, and returns the resulting matrix, namely:

realloc(\mathbf{k}^t , $\overrightarrow{\mathbf{k}}$, τ):

1. For all $i = 1, \dots, n$ do:

Set $k(x_i, \tau) = k'(x_i)$, where $k(x_i, \tau)$ is an element of the matrix \mathbf{k}^t , and $k'(x_i)$ is an element of the vector $\overrightarrow{\mathbf{k}}$.

2. Return updated matrix \mathbf{k}^t .

Then, given a matrix $\mathbf{k}^t = ||k(x_i, \tau)||_{n \times t}$, the reallocation problem at time τ requires a vector $\overrightarrow{\mathbf{k}} = ||k'(x_i)||_{n \times 1}$ to be found such that

$$Q(\text{realloc}(\mathbf{k}^t, \overrightarrow{\mathbf{k}}, \tau)) \rightarrow \min, \quad (2.18)$$

$$\text{subject to : } k'(x_i) \geq 0 \text{ for any } x_i \in X, \text{ and } \sum_{i=1}^n k'(x_i) \leq K(\tau). \quad (2.19)$$

In other words, the reallocation problem requires an optimal allocation to be found at time τ , while the detection function is defined by the Koopman formulation (Equation 2.1). Since for a fixed time τ both the weighting coefficients $\sigma(x_i, \tau)$ and the non-weighted efforts $k(x_i, \tau)$, are constant for each x_i , $i = 1, \dots, n$, the reallocation problem can be solved by Stone's previously defined algorithm, Algorithm 2.1. In fact, Brown showed that the optimal solution of the problem (2.12–2.13) is obtained by optimal reallocations at times $\tau = 0, 1, 2, \dots, t$.

Theorem 2.4 [12]. *A matrix \mathbf{k}^t is an optimal solution of the minimization problem (2.12–2.13) if and only if its τ th row solves the reallocation problem at time τ , $\tau = 0, 1, 2, \dots, t$.*

Proof. Let us fix a time τ and consider a minimization problem of (2.12–2.13) for the τ th row $\mathbf{k}^t(\tau)$ of the matrix \mathbf{k}^t . In such a case, the probability of not detecting the target is defined as

$$Q(\mathbf{k}^t(\tau)) = \sum_{\overrightarrow{\sigma} \in \mathcal{O}: x^\tau \in \overrightarrow{\sigma}} p(x^\tau) \exp(-\sigma(x^\tau, \tau)k(x^\tau, \tau)), \quad (2.20)$$

where the sum is calculated over all the target's trajectories such that at time τ the target is located at point x^τ , while the constraints (Equation 2.13) obtain the form of inequalities $0 \leq k(x_i, \tau) \leq K(\tau)$, $i = 1, \dots, n$.

We apply Lemma 2.1 to the minimization problem of the probability $Q(\text{realloc}(\mathbf{k}^t(\tau), \overrightarrow{\mathbf{k}}, \tau)) = Q(\overrightarrow{\mathbf{k}})$ with the determined constraints. According to the requirements of the lemma, vector $\overrightarrow{\mathbf{k}}$ solves the minimization problem (2.18–2.19) if and only if for a fixed τ there exists a value $\mu^\tau \geq 0$ such that the requirements of the lemma hold. Thus, if for each $\tau = 0, 1, 2, \dots, t$ the row $\mathbf{k}^t(\tau)$ of the matrix \mathbf{k}^t solves the optimization problem (Equations 2.18 and 2.19), then there exist corresponding values $\mu^\tau \geq 0$, $\tau = 0, 1, 2, \dots, t$, as required by Lemma 2.1 applied to the original minimization problem (2.12–2.13). ■

The proven Theorem 2.4 along with Lemma 2.1 form the basis of Brown's algorithm of search for a moving target. The algorithm builds optimal search density u^t up to time t by evaluating Stone's Algorithm 2.1 and solving the reallocation problem at each time $\tau = 0, 1, 2, \dots, t$. Brown's algorithm is outlined as follows.

Algorithm 2.2 (Brown's algorithm) [12]. Given location probabilities $p^0(x_i)$, weighting coefficients $\sigma(x_i, \tau) > 0$, available costs $K(\tau)$, $i = 1, \dots, n$, $\tau = 0, 1, 2, \dots, t$, initial matrix of non-weighted search efforts k_0^t , and a number ε , $0 < \varepsilon \ll 1$:

1. Initialize matrix k^t : Set $k_{next}^t = k_0^t$.
2. Do
 - 2.1. Set $k_{current}^t = k_{next}^t$
 - 2.2. For each $\tau = 1, \dots, t$ do:
 - 2.2.1. For each $i = 1, \dots, n$ do:
Obtain the target location probabilities $p^\tau(x_i)$.
 - 2.2.2. Find-the optimal solution of the reallocation problem at time τ :
Set $k = Algorithm\ 2.1(p^\tau(x_i), \sigma(x_i, \tau), K(\tau))$.
 - 2.2.3. Conduct reallocation by use of vector \vec{k} :
Set $k_{next}^t = reallocate(k_{next}^t, \vec{k}, \tau)$.
While $|Q(k_{next}^t) - Q(k_{current}^t)| \geq \varepsilon$
 3. Return k_{next}^t .

■

At the first trial from $\tau = 1$ until $\tau = t$, Algorithm 2.2 creates a myopic search plan, while in the next cycles of trials it refines this plan up to the optimal solution of the minimization problem (2.12–2.13) with respect to the constant ε [12]. The actions of the algorithm are illustrated by the following example.

Example 2.6 Similar to the previous examples, assume that the target location probabilities $p^{t=0}(x)$, $x \in X$, at time $t = 0$ are combined from two bi-normal distributions (see Figures 2.4a and 2.11a) and that the target's movement is governed by the diffusion-type Markov process.

Let the applied search efforts be $K(0) = K(1) = 10$ and $K(\tau) = k \times \tau$ with constant $k = 10$, as in Example 2.4. Search density u^τ and target location density v^τ for the times $\tau = 1, \tau = 10$, and $\tau = t = 100$ with corresponding efforts $K(1) = 10$, $K(10) = 100$, and $K(100) = 1000$ are shown in Figure 2.13.

From the figure it follows that when the available search effort is $K = 10$, Algorithm 2.2 allocates this effort at the point with maximal location probability (see Figure 2.13a, b). When the available search effort is $K = 100$, the obtained allocation (see Figure 2.13d) is similar to the allocation that was obtained by Algorithm 2.1 (see Figures 2.13d and 2.12b). Finally, for a search effort of $K = 1000$, Algorithm 2.2 allocates this effort approximately uniformly with respect to the target search density (see Figure 2.13f). ■

The implementation of Algorithm 2.2 requires computation of the probabilities $p(\vec{o}(t))$ of the target's trajectories $\vec{o}(t) = \langle x^{t=0}, \dots, x^{t-2}, x^{t-1}, x^t \rangle$, $x^t \in X$. In the case of *unconstrained Markovian motion* of the target, these probabilities are defined as follows:

$$p(\vec{o}(t)) = p_r(x^{\tau=0})\rho(x^0, x^1)\rho(x^1, x^2)\dots\rho(x^{\tau=t-1}, x^{\tau=t}), \quad (2.21)$$

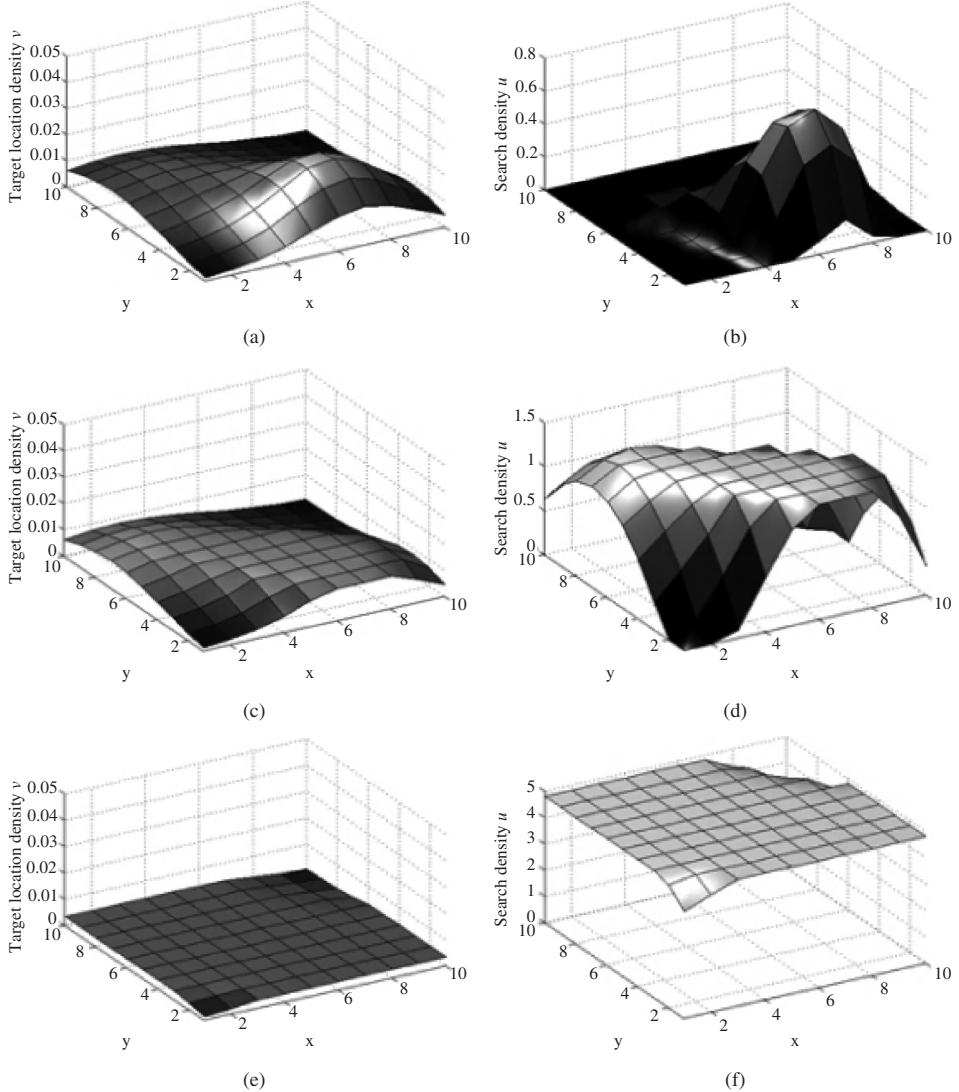


Figure 2.13 Dynamics of Markovian target location density and search density for optimal allocations provided by Algorithm 2.2. (a) Target location density $v^{t=0}$. (b) Search density $u^{t=0}$; available search effort $K = 10$. (c) Target location density $v^{t=10}$. (d) Search density $u^{t=10}$; available search effort $K = 100$. (e) Target location density $v^{t=100}$. (f) Search density $u^{t=100}$; available search effort $K = 1000$.

where $p_r(x^{\tau=0})$ is the probability that the target starts at point $x^{t=0}$, and $\rho(x^\tau, x^{\tau+1})$, $\tau = 0, 1, 2, \dots, t$, is the probability that the target moves from point x^τ to point $x^{\tau+1}$. In the case of *constrained Markovian motion* of the target, the trajectory probabilities $p(\overrightarrow{o}(t))$ are defined as

$$p(\vec{o}(t)) = p_r(x^{\tau=0})\rho(x^0, x^1)\rho(x^1, x^2)\dots\rho(x^{\tau=t-1}, x^{\tau=t})p_s(x^{\tau=t}), \quad (2.22)$$

and include the probability $p_s(x^{\tau=t})$ that the target arrives at point $x^{\tau=t}$ at the end of its motion. If all points of the sequence $x^{\tau=t} \in X$ in Equation 2.22 satisfy $p_s(x^{\tau=t}) = 1$, then the sequence defines a trajectory probability (Equation 2.21) of unconstrained motion.

By using probabilities (Equations 2.21 and 2.22) the target location probabilities $p^\tau(x)$, $x \in X$, required in Algorithm 2.2 (Line 2.1.1) can be calculated recursively as follows [12]. Fix a point $x \in X$ and define two functions:

$$\begin{aligned} \text{reach}(k^t, x, \tau) &= \sum p_r(x^0) \rho(x^0, x^1) \rho(x^1, x^2) \dots \\ &\quad \rho(x^{\tau-1}, x) e^{-\sum_{\tilde{\tau}=1}^{\tau-1} \sigma(x^{\tilde{\tau}}, \tilde{\tau}) k(x^{\tilde{\tau}}, \tilde{\tau})}, \text{ and} \\ \text{survive}(k^t, x, \tau) &= \sum \rho(x, x^{\tau+1}) \rho(x^{\tau+1}, x^{\tau+2}) \dots \\ &\quad \rho(x^{\tau-1}, x^t) p_s(x^t) e^{-\sum_{\tilde{\tau}=\tau+1}^t \sigma(x^{\tilde{\tau}}, \tilde{\tau}) k(x^{\tilde{\tau}}, \tilde{\tau})}. \end{aligned}$$

The first ‘reach’ function represents the probability that the target has not been detected up to time τ and has reached point $x \in X$. The second ‘survive’ function represents the probability that the target has not been detected at point $x \in X$ starting from time τ up until the end of the search. Both probabilities can be calculated by the following recursion procedures [12]:

$$\begin{aligned} \text{reach}(k^t, x, 0) &= p_r(x^0), \\ \text{reach}(k^t, x, \tau + 1) &= \sum_{i=1}^n \text{reach}(k^t, x_i, \tau) \exp(-\sigma(x_i, \tau) k(x_i, \tau)) \rho(x_i, x); \\ \text{survive}(k^t, x, t) &= p_s(x^t), \\ \text{survive}(k^t, x, \tau - 1) &= \sum_{i=1}^n \rho(x, x_i) \exp(-\sigma(x_i, \tau) k(x_i, \tau)) \text{survive}(k^t, x_i, \tau). \end{aligned}$$

Accordingly, an application of the search efforts k^t results in the target location probabilities $p^\tau(x)$, $x \in X$, at time τ that are calculated as a multiplication of the *reach* and *survive* probabilities, that is,

$$p^\tau(x) = \text{reach}(k^t, x, \tau) \times \text{survive}(k^t, x, \tau - 1).$$

Since calculation of the target location probabilities $p^\tau(x)$ requires complete recursions over all possible trajectories, an implementation of such techniques in Algorithm 2.2 is possible only for relatively small sample spaces. However, under the requirements of Lemma 2.1 and Theorem 2.4 that are satisfied for a Markovian search and the Koopman detection function, Brown’s Algorithm 2.2 results in optimal search density u^t . In Chapter 3, the constrained target motion is addressed by using dynamic programming techniques.

Finally, let us consider a general FAB algorithm of search for a moving target [13, 15, 22] that implements a similar recursive computation of the target location probabilities and results in optimal search density u^t .

Assume that, similar to Section 2.1.2, the target cannot be erroneously detected at point $x \in X$ if it is not located at that point. Recall that, following such an assumption, the implementation of the detection function φ_k results in the probability $\psi(x, t) = \Pr[D^t | \text{target is located at } x]$ of detecting the target at point x at time t , where D^t stands for

the event that the target is detected at time $t = 0, 1, 2, \dots$. Denote by $\bar{\psi}(x, t)$ the probability that the target has not been detected at point x at time t , that is, $\bar{\psi}(x, t) = 1 - \psi(x, t)$. Accordingly, the function

$$\bar{\Psi}(t) = \Pr\{\bar{D}^t\} = \prod_{i=1}^n \bar{\psi}(x_i, t)$$

where $\bar{D}^t = \{\text{target is not detected at time } t\}$, $t = 0, 1, 2, \dots$, defines the probability of not detecting the target at time t . Recall also that the target transition probabilities ρ_{ij} , $i, j = 1, \dots, n$, are defined, in general, as $\rho_{ij} = \Pr\{x^{t+1} = x_j | x^t = x_i, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}$ (see Section 2.1.2).

Next we consider transition at time t given that the target has not been detected at the previous times. Then, the transition probabilities are denoted by

$$\rho_{ij}^t = \rho(x_i, x_j, t).$$

Let $Y = ||y^\tau||_{\tau=0}^t$ be a sequence of random variables such that for any $\tilde{\tau} < \tau$, $\tilde{\tau}, \tau = 0, 1, 2, \dots, t$, it follows that y^τ does not depend on $x^{\tilde{\tau}}$. Define the probability of not detecting the target as

$$Q(\bar{\psi}) = E\left(\sum_{\tau=0}^t y^\tau \prod_{\tilde{\tau}=0}^{\tau} \bar{\psi}(\tilde{\tau})\right),$$

where E stands for the expectation. For a sequence Y such that $y^\tau = 0$ for $\tau < t$ and $y^t = 1$, $Q(\bar{\psi})$ gives the probability of not detecting the target up to time t , that is, $Q(\bar{\psi}) = \Pr\{\bar{D}^t, \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}$. Note that in contrast to the conditional probability $\Pr\{\bar{D}^t | \bar{D}^{t-1}, \bar{D}^{t-2}, \dots, \bar{D}^0\}$ considered in Section 2.1.2, $Q(\bar{\psi})$ specifies a joint probability of not detecting the target. Then, given a final time t , the search-planning problem requires a feasible function $\bar{\psi}^t : X \rightarrow [0, 1]$ to be found, such that it minimizes the probability $Q(\bar{\psi})$ of not detecting the target.

A solution of the defined minimization problem is obtained by the FAB algorithm based on the following theorem, specifying sufficient (but not necessary) conditions for the optimal search density. Functions p_r and p_s that are defined in the formulation of the theorem have the same meaning as the *reach* and *survive* functions specified above.

Theorem 2.5 [13]. *Assume that before starting the search and after its completion the target cannot be detected, that is, $\Pr\{\bar{D}^{\tau=0}\} = \Pr\{\bar{D}^{\tau \geq t}\} = 1$. Then the probability $Q(\bar{\psi})$ of not detecting the target is specified by the following recurrent formulas:*

$$Q(\bar{\psi}) = \sum_{i=1}^n p_r(x_i, t) \bar{\psi}(x_i, t) p_s(x_i, t) \text{ for } t = 0,$$

$$Q(\bar{\psi}) = E\left(\sum_{\tau=0}^{t-1} y^\tau \prod_{\tilde{\tau}=0}^{\tau} \bar{\psi}(\tilde{\tau})\right) + \sum_{i=1}^n p_r(x_i, t) \bar{\psi}(x_i, t) p_s(x_i, t) \text{ for } t > 0;$$

where the values $p_r(x_i, t)$ and $p_s(x_i, t)$, $i = 1, \dots, n$, are defined by FAB recursive equations as follows:

$$p_r(x_i, 0) = p^0(x_i), \quad (2.23a)$$

$$p_r(x_j, \tau + 1) = \sum_{i=1}^n \rho(x_i, x_j, \tau) \bar{\psi}(x_i, \tau) p_r(x_i, \tau), \quad j = 1, \dots, n; \quad (2.23b)$$

$$p_s(x_i, t) = E(Y|x^t = x_i), \quad (2.24a)$$

$$p_s(x_i, \tau) = E(Y|x^\tau = x_i) + \sum_{j=1}^n \rho(x_i, x_j, \tau) \bar{\psi}(x_j, \tau + 1) p_s(x_j, \tau + 1). \quad (2.24b)$$

$$\tau = 1, 2, \dots, t - 1.$$

Proof. Denote by $\mathbf{1}(x_i, t)$ an indicator function of the event $\{x^t = x_i\}$, $i = 1, \dots, n$, that is, $\mathbf{1}(x_i, t) = 1$ if $x^t = x_i$ and $\mathbf{1}(x_i, t) = 0$ otherwise. Then, since $\sum_{i=1}^n \mathbf{1}(x_i, t) = 1$ for any $t = 0, 1, 2, \dots$, for probability $Q(\bar{\psi})$ of not detecting the target it follows that

$$\begin{aligned} Q(\bar{\psi}) &= E \left(\sum_{\tau=0}^t y^\tau \prod_{\tilde{\tau}=0}^{\tau} \bar{\psi}(\tilde{\tau}) \right) \\ &= E \left(\sum_{\tau=0}^{t-1} y^\tau \prod_{\tilde{\tau}=0}^{\tau} \bar{\psi}(\tilde{\tau}) \right) + E \left(\prod_{\tilde{\tau}=0}^{\tau-1} \bar{\psi}(\tilde{\tau}) \bar{\psi}(\tau) \sum_{\tilde{\tau}=\tau}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+1}^{\tilde{\tau}} \bar{\psi}(\tilde{\tau}') \right) \\ &= E \left(\sum_{\tau=0}^{t-1} y^\tau \prod_{\tilde{\tau}=0}^{\tau} \bar{\psi}(\tilde{\tau}) \right) \\ &\quad + \sum_{i=1}^n E \left(\mathbf{1}(x_i, t) \prod_{\tilde{\tau}=0}^{\tau-1} \bar{\psi}(\tilde{\tau}) \bar{\psi}(\tau) \sum_{\tilde{\tau}=\tau}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+1}^{\tilde{\tau}} \bar{\psi}(\tilde{\tau}') \right). \end{aligned}$$

From the Markov property, for the terms of the last addendum it follows that

$$\begin{aligned} &E \left(\mathbf{1}(x_i, t) \prod_{\tilde{\tau}=0}^{\tau-1} \bar{\psi}(\tilde{\tau}) \bar{\psi}(\tau) \sum_{\tilde{\tau}=\tau}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+1}^{\tilde{\tau}} \bar{\psi}(\tilde{\tau}') \right) \\ &= E \left(\mathbf{1}(x_i, t) \prod_{\tilde{\tau}=0}^{\tau-1} \bar{\psi}(\tilde{\tau}) \right) \bar{\psi}(x_i, \tau) E \left(\sum_{\tilde{\tau}=\tau}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+1}^{\tilde{\tau}} \bar{\psi}(\tilde{\tau}') | x^\tau = x_i \right). \end{aligned}$$

For $i = 1, \dots, n$, let us specify

$$\begin{aligned} p_r(x_i, \tau) &= E \left(\mathbf{1}(x_i, \tau) \prod_{\tilde{\tau}=0}^{\tau-1} \bar{\psi}(\tilde{\tau}) \right) \quad \text{and} \\ p_s(x_i, \tau) &= E \left(\sum_{\tilde{\tau}=\tau}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+1}^{\tilde{\tau}} \bar{\psi}(\tilde{\tau}') | x^\tau = x_i \right). \end{aligned}$$

Then, it is required to demonstrate that these values are governed by the recursive Equations 2.3 and 2.24.

At the initial time the search has not been started, so the probability of not detecting the target is equal to 1, that is, $\prod_{\tilde{\tau}=0}^0 \bar{\psi}(\tilde{\tau}) = 1$. Thus, the expectation that the target reaches the point x_i , $i = 1, \dots, n$, at the initial time is exactly its location probability $p^0(x_i)$. Hence, Equation 2.23a holds.

Let us consider the probability $p_r(x_j, \tau + 1)$, $j = 1, \dots, n$. Since $\sum_{i=1}^n \mathbf{1}(x_i, \tau) = 1$, it follows that $p_r(x_j, \tau + 1) = \sum_{i=1}^n E \left(\mathbf{1}(x_i, \tau) \mathbf{1}(x_j, \tau + 1) \prod_{\tilde{\tau}=0}^{\tau} \bar{\psi}(\tilde{\tau}) \right)$, $\tau = 1, 2$,

$\dots, t - 1$. Then, by the independence of the probability of not detecting the target at previous times on the current target location and by using the target transition probabilities,

$$\begin{aligned} p_r(x_j, \tau + 1) &= \sum_{i=1}^n \rho(x_i, x_j, \tau) E\left(\mathbf{1}(x_i, \tau) \prod_{\tilde{\tau}=0}^{\tau} \bar{\Psi}(\tilde{\tau})\right) \\ &= \sum_{i=1}^n \rho(x_i, x_j, \tau) \bar{\Psi}(\tau) E\left(\mathbf{1}(x_i, \tau) \prod_{\tilde{\tau}=0}^{\tau-1} \bar{\Psi}(\tilde{\tau})\right) \\ &= \sum_{i=1}^n \rho(x_i, x_j, \tau) \bar{\Psi}(\tau) p_r(x_i, \tau), \end{aligned}$$

which corresponds to Equation 2.23b.

Now let us consider the survive probability $p_s(x_i, \tau) = E\left(\sum_{\tilde{\tau}=\tau}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+1}^{\tilde{\tau}} \bar{\Psi}(\tilde{\tau}') | x^\tau = x_i\right)$. If the target has not been detected before the end of the search at time t , then it indeed will not be detected after this time. Thus, the probability of not detecting the target at times $\tau \geq t$ is equal to 1, that is, $\prod_{\tilde{\tau}=t}^{\infty} \bar{\Psi}(\tilde{\tau}) = 1$. Hence, $p_s(x_i, t) = E\left(\sum_{\tilde{\tau}=t}^t y^{\tilde{\tau}} | x^t = x_i\right) = E(Y|x^t = x_i)$, as required by Equation 2.24a.

Finally, since $\sum_{\tilde{\tau}=\tau}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+1}^{\tilde{\tau}} \bar{\Psi}(\tilde{\tau}') = y^{\tau+1} + \bar{\Psi}(\tau + 1) \sum_{\tilde{\tau}=\tau+1}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+2}^{\tilde{\tau}} \bar{\Psi}(\tilde{\tau}')$, it follows that $p_s(x_i, \tau) = E(Y|x^\tau = x_i) + E\left(\sum_{\tilde{\tau}=\tau+1}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+2}^{\tilde{\tau}} \bar{\Psi}(\tilde{\tau}') | x^{\tau+1} = x_i\right)$. Then, similar to the above reasons regarding independence and transitions, we obtain

$$\begin{aligned} p_s(x_i, \tau) &= E(Y|x^\tau = x_i) + \sum_{j=1}^n \rho(x_i, x_j, \tau) \\ &\quad \times E\left(\bar{\Psi}(\tau + 1) \sum_{\tilde{\tau}=\tau+1}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+2}^{\tilde{\tau}} \bar{\Psi}(\tilde{\tau}') | x^{\tau+1} = x_j\right) \\ &= E(Y|x^\tau = x_i) + \sum_{j=1}^n \rho(x_i, x_j, \tau) \bar{\Psi}(x_j, \tau + 1) \\ &\quad \times E\left(\sum_{\tilde{\tau}=\tau+1}^t y^{\tilde{\tau}} \prod_{\tilde{\tau}'=\tau+2}^{\tilde{\tau}} \bar{\Psi}(\tilde{\tau}') | x^{\tau+1} = x_j\right) \\ &= E(Y|x^\tau = x_i) + \sum_{j=1}^n \rho(x_i, x_j, \tau) \bar{\Psi}(x_j, \tau + 1) p_s(x_j, \tau + 1), \end{aligned}$$

■

which corresponds to Equation 2.24b.

Note that the proof of the theorem holds for search for a single target, but the theorem cannot be applied for search for multiple targets.

The goal of the search planning is to obtain a function ψ that specifies a distribution of the search efforts over the sample space X , such that the probability $Q(\bar{\psi})$ of not detecting the target reaches its minimum. Given a set Ψ of functions ψ , a function $\psi^* \in \Psi$ such that $Q(\bar{\psi}^*) = \min_{\psi \in \Psi} \{Q(\bar{\psi})\}$ is called *critical* or *optimal* for a given search-planning problem.

The correspondence of the functions ψ with the above allocations w and search densities u^t is as follows. According to the assumptions that the target cannot be erroneously detected at point $x \in X$ if it is not located at x , for a fixed time t the values of $\psi^t(x) = \varphi(x, \kappa)$, where φ is a detection function and κ is a search effort. If at each time the agent checks a single point, that is, if $A^t = \{x^t\}$, $t = 0, 1, 2, \dots$, then $\varphi(x, \kappa) = w(x) = u^t(x)$. Hence, finding the function ψ^* results in a solution for the search-planning problem in terms of optimal allocation and search density.

The formulation of Washburn's FAB algorithm [13] is based on the following remark. Let $\psi_1, \psi_2 \in \Psi$ be two feasible probability functions, and assume that for any time $\tau < t$ it follows that $\psi_1(x_i, \tau) = \psi_2(x_i, \tau)$, $i = 1, \dots, n$. Then, based on Equations 2.23 and 2.24 it follows that at time t the difference between the probabilities $Q(\bar{\psi}_1)$ and $Q(\bar{\psi}_2)$ of not detecting the target when using functions ψ_1 and ψ_2 , correspondingly, is defined by the difference between the probabilities $\psi_1(x_i, t)$ and $\psi_2(x_i, t)$, that is,

$$Q(\bar{\psi}_1) - Q(\bar{\psi}_2) = \sum_{i=1}^n p_r(x_i, t)(\bar{\psi}_1(x_i, t) - \bar{\psi}_2(x_i, t))p_s(x_i, t).$$

Thus, a critical function value ψ^* can be obtained as a result of sequential minimizations of the sum $\sum_{i=1}^n p_r(x_i, \tau) \bar{\psi}(x_i, \tau)p_s(x_i, \tau)$ for times $\tau < t$. The FAB algorithm implements such minimizations as follows.

Algorithm 2.3 (Washburn algorithm) [13, 22]. Given location probabilities $p^0(x_i)$, $i = 1, \dots, n$, a sequence $Y = \{y^\tau\}_{\tau=0}^t$ of random variables, and a set Ψ of available functions ψ :

1. Set initial function $\psi \in \Psi$ and calculate initial $\bar{\psi}$.
2. While ψ is not critical do:
 - 2.1. For each $i = 1, \dots, n$ do: Set initial $p_s(x_i, t)$ according to Equation 2.24.
 - 2.2. For each $i = 1, \dots, n$ do: Set initial $p_r(x_i, 0)$ according to Equation 2.23a.
 - 2.3. For each $\tau = 1, 2, \dots, t$ do:
 - 2.3.1. Set $\psi = \operatorname{argmin}_{\vartheta \in \Psi} \sum_{i=1}^n p_r(x_i, \tau) \bar{\vartheta}(x_i, \tau)p_s(x_i, \tau)$
 - 2.3.2. For each $i = 1, \dots, n$ do: Set $p_r(x_i, \tau)$ according to Equation 2.23b.
3. Return $\psi^* = \psi$. ■

The idea of Washburn's algorithm above is similar to the idea of Brown's Algorithm 2.2. Nevertheless, in Line 2.3.1 it requires the solution of an optimization problem over functions $\psi \in \Psi$ that, in general, requires more complex methods than the execution of Algorithm 2.1, which is applied in Line 2.2.2 of Brown's Algorithm 2.2. Execution of the algorithm for the Koopman detection function is illustrated by the following example.

Example 2.7 Let the target location probabilities $p^{t=0}(x)$, $x \in X$, at time $t = 0$ be defined as in Example 2.6 and let, as noted above, the target's movement be governed by a diffusion-type Markov process. In addition, let the applied search efforts be $K(0) = K(1) = 10$ where $K(\tau) = k \times \tau$ with constant $k = 10$. Assume that the detection probability corresponds to the Koopman equation and, similar to Algorithm 2.2, is defined by $\psi(x, k) = 1 - e^{-\sigma(x, t)k(x, t)}$, $x \in X$, $t = 0, 1, 2, \dots$, where weights $\sigma(x, t) > 0$ and the non-weighted efforts $k(x, t)$ are cumulative in time.

The search density u^τ and the target location density v^τ that are obtained by Algorithm 2.3 for the times $\tau = 1$, $\tau = 10$, and $\tau = t = 100$ with corresponding efforts $K(1) = 10$, $K(10) = 100$, and $K(100) = 1000$ are shown in Figure 2.14.

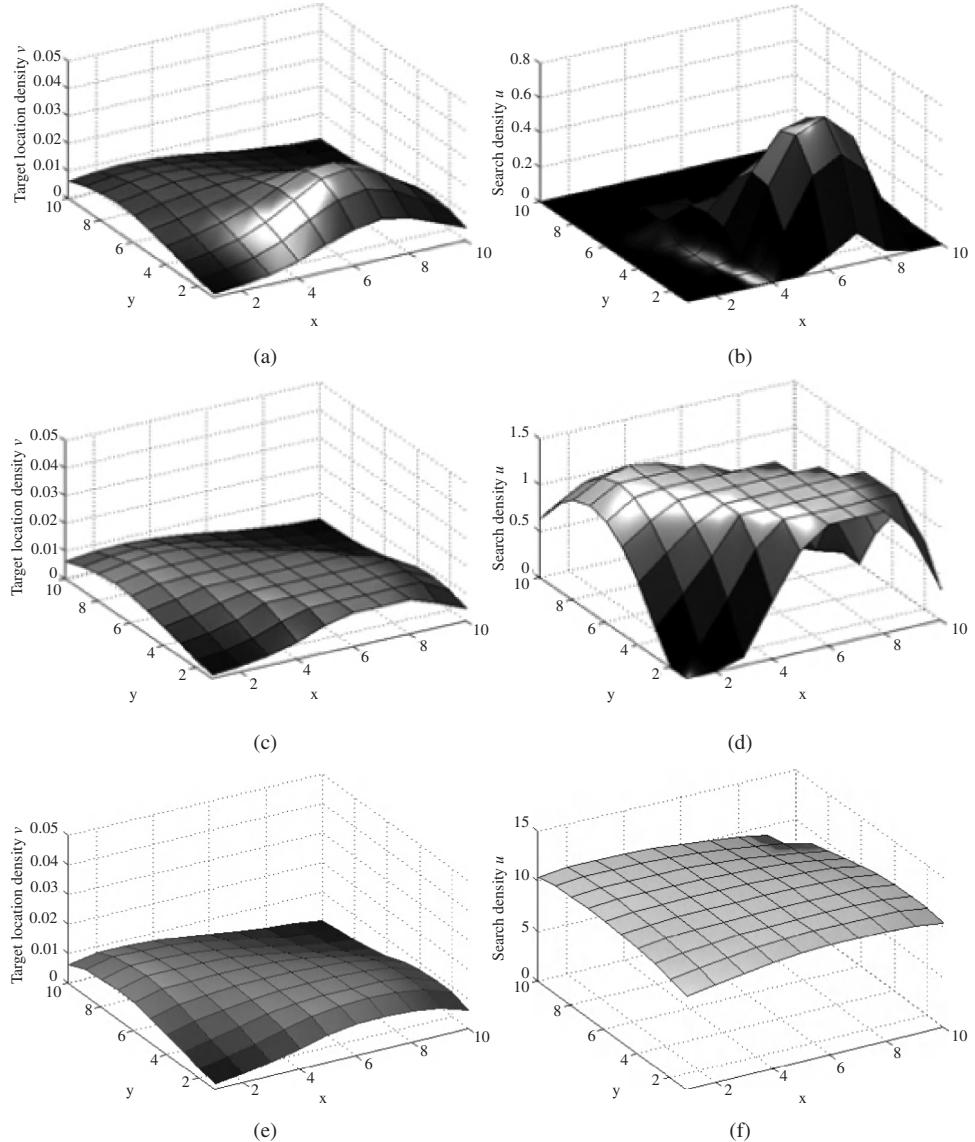


Figure 2.14 Dynamics of Markovian target location density and search density for optimal allocations provided by Algorithm 2.3. (a) Target location density $v^{t=0}$. (b) Search density $u^{t=0}$; available search effort $K = 10$. (c) Target location density $v^{t=10}$. (d) Search density $u^{t=10}$; available search effort $K = 100$. (e) Target location density $v^{t=100}$. (f) Search density $u^{t=100}$, available search effort $K = 1000$.

Accordingly, the probability of not detecting the target by using these search densities is $Q(\bar{\psi}) = 0.01$.

Based on a comparison of the densities shown in Figure 2.14 with those that were obtained by Brown's algorithm (see Figure 2.13) it follows that at the beginning of the search with relatively small search efforts both Algorithms 2.2 and 2.3 result in the same search densities (see Figures 2.13b and 2.14b). While the available search effort increases, the difference between the obtained search densities increases as well (see Figures 2.13d, 2.14d, 2.13f, and 2.14f). This observation illustrates the fact that the conditions presented by Theorem 2.5 are sufficient but not necessary for the optimality of the chosen detection probability function. ■

Note that all the considered algorithms result in search density functions u^t that relate the search effort κ^t to the target density v^t over the set $X = \{x_1, x_2, \dots, x_n\}$, while certain trajectories of the search agents are obtained by relying on additional assumptions and algorithms applied to the search densities obtained. Nevertheless, sequential creation of the search density, as implemented in Brown's Algorithm 2.2 and in Washburn's Algorithm 2.3, gives rise to methods in which the trajectories of the search agent are obtained by a local search approach. In the following chapters, we consider solution methods of search and screening problems, especially Stewart's branch-and-bound algorithm [14, 23].

2.2 Group-testing search

The problem of search in the context of *group-testing theory* appears in applications where the searcher (or a finite number of searchers) can observe a set of possible target locations simultaneously. The searcher starts with a set of locations and at each search step observes a subset of this set, or another set, to obtain the observation result. In the case of a certain detection that corresponds to errorless observations, if the target is not located somewhere in the chosen subset, then the observation result is denoted by '0' or 'false'; otherwise, the observation result is denoted by '1' or 'true.' When considering uncertain detection that corresponds to the possibility of erroneous observations, the observation result is often represented by a value from the interval $[0, 1]$.

A group-testing formulation of the search problem generalizes the search and screening problem above and frames a search process where the observed areas are represented by any subsets of the target locations set. The goal of the search process in the group-testing formulation is also different. In contrast to the search and screening process that aims at maximizing the probability of target detection by a suitable distribution of search efforts, in the group-testing search, the goal is often to find such a search policy that minimizes the expected number of search steps up to a certain detection of the target.

Similar to the search and screening problem, the studies of group-testing procedures were initiated in 1942 during World War II. The initial problem, which has been considered by the Price Statistics Branch of the Research Division of the Office of Price Administration, USA, was to find an optimal procedure for testing blood samples for the presence of an antigen. Such a procedure was published in 1943 by Dorfman [24], who also presented other possible applications of the group-testing method. A direct modification of the Dorfman method was suggested in 1956 by Sterrett [25, 26], and the obtained method formed the basis for further studies (for a brief history of the group-testing approach and the role of D. Rosenblatt see [27]). Nowadays, the group-testing approach is applied in different

fields [28–30], including quality control, and implements different methods for creating optimal testing policies [27, 31, 32]. In the next section, we mainly consider the group-testing methods of search that are based on statistical methods [31–33] and on methods of information theory [34, 35].

2.2.1 General definitions and notation

Let, as defined above (see Section 2.1.1), $X = \{x_1, x_2, \dots, x_n\}$ be a *locations set* such that each point $x_i \in X$, $i = 1, \dots, n$, represents the possible target location, and let $t = 0, 1, 2, \dots$ be discrete times. Recall that in the consideration of the search and screening problem it was assumed that the search agent screens the set X by a sensor, which detects whether or not the target is located at a certain *observed area* $A \subset X$. For the cardinality $m = |A|$ of the observed area, it follows that $1 \leq m \leq n$. In addition, let $\mathcal{X} = 2^X$ be a *power set* of the set X , that is, \mathcal{X} is a set that contains all possible subsets of X . Then, any observed area A is an element of the set \mathcal{X} . Since the size of the available observed areas depends on the characteristics of the sensors used, in practical tasks a certain subset of the power set is relevant. The restricted observed areas will be considered in later sections.

Let A^t be an observed area that is chosen and checked by the searcher at time t , $t = 0, 1, 2, \dots$. In the case of errorless detection, which corresponds to detection with a definite range law (see Figure 2.3 and further remarks), the *observation result* z^t is equal to ‘1’ or ‘true’ if, at the considered time t , the target location $x^t \in A^t$ and is equal to ‘0’ or ‘false’ if $x^t \notin A^t$. Thus for errorless detection it is assumed that $z^t = \mathbf{1}(A^t, x^t)$, where $\mathbf{1}$ is the *indicator function* such that $\mathbf{1}(A, x) = 1$ if $x \in A$ and $\mathbf{1}(A, x) = 0$ otherwise. For erroneous detection, it is assumed that, as mentioned above, the observation result $z^t = z(A^t, x^t)$ is determined by a certain *observation function* $z : \mathcal{X} \times X \rightarrow [0, 1]$ that specifies a result of observation of the observed area $A \in \mathcal{X}$, while the target is located at $x \in X$. If, according to the assumption (Equation 2.2), the target cannot be erroneously detected at point $x \in X$ if it is not located at that point, then the observation result of imperfect detection is determined by detection probability function ψ^t (see Section 2.1.2) as $z^t = \psi^t(A^t) = \sum_{x \in A^t} \psi^t(x)$, and for the observation result it follows that $0 \leq z^t \leq 1$.

In terms of group testing and corresponding decision making, the search process can be defined as follows. At time $t = 0, 1, 2, \dots$, the target chooses location $x^t \in X$. The searcher chooses an observed area $A^t \in \mathcal{X}$ and obtains observation result $z^t = z(A^t, x^t)$. According to the obtained result z^t and the *search history*, which includes direct information regarding previous observations [27] (or is represented by a certain learning process [36]), the searcher makes a decision regarding the choice of the next observed area $A^{t+1} \in \mathcal{X}$ to be selected. If the target is found, then the search process terminates. The dynamics of the system is shown in Figure 2.15.

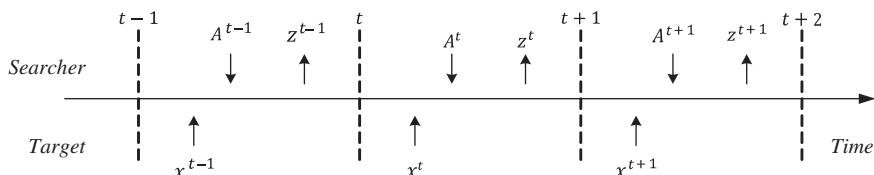


Figure 2.15 Dynamics of the group-testing search.

According to this search process, at each time $t = 0, 1, 2, \dots$, the searcher has to solve two main problems:

- Selecting the observed area $A^{t+1} \in \mathcal{X}$ given previous selections and their corresponding observation results.
- Determining either to stop or to continue the search.

The problem of selecting the next observed area $A^{t+1} \in \mathcal{X}$ is often addressed by relying on the probability measures, such as the location probabilities, observed probabilities, and estimated probabilities (see Section 2.1.1) defined over the locations set X . In the worst case, while the information regarding the target location is not available, the initial location probabilities can be represented by a uniform distribution according to the maximum entropy principle. In contrast, the problem of search termination is based on additional assumptions that depend on the considered task and sometimes on cost considerations [37].

In general, conditions of search termination are determined by the values of certain control variables that are combined into a *control set* [38, 39] and are governed by a *control function* c . It is assumed that, given an observed area A^t , the observation result z^t , and the search history, control function c obtain the values $c^t = \text{terminate}$ or $c^t = \text{continue}$. If the value of the control function does not depend on the search history, or if the search history is represented by the result of imperfect observations, then the control function is specified as $c : \mathcal{X} \times [0, 1] \rightarrow \{\text{terminate}, \text{continue}\}$ and its values are denoted by $c^t = c(A^t, z^t)$.

These concepts allow the group-testing search to be presented in the form of a general search procedure. Assume that the values $c^t, t = 0, 1, 2, \dots$, of the control function c do not depend on the search history, so that for a given time t they are determined by the observed area A^t and obtained observation result z^t . Then the group-testing search is presented by the following procedure.

Procedure 2.1 (group-testing search). Given locations set X , its power set \mathcal{X} , the observation function z , and the control function c , such that for any $A^t \in \mathcal{X}, x^t \in X, t = 0, 1, 2, \dots$, it follows that $z^t = z(A^t, x^t) \in [0, 1]$ and $c^t = c(A^t, z^t) \in \{\text{terminate}, \text{continue}\}$, do:

1. Set $t = 0$.
2. Choose an observed area $A^t \in \mathcal{X}$; usually $A^{t=0} = X$.
3. Observe area A^t : Set $z^t = z(A^t, x^t)$.
4. Specify control: Set $c^t = c(A^t, z^t)$.
5. While $c^t = \text{continue}$ do:

- 5.1. Increase t : Set $t = t + 1$.

Target turn:

- 5.2. Choose a point $x^t \in X$.

Searcher turn:

- 5.3. Choose an observed area $A^t \in \mathcal{X}$.

5.4. Observe area A^t : Set $z^t = z(A^t, x^t)$.

5.5. Specify control: Set $c^t = c(A^t, z^t)$.

6. Return A^t . ■

Note that the group-testing search procedure follows the actions of the procedure of local search that was introduced in Section 1.4 with additional information regarding the target's action. In the presented form, the group-testing search procedure describes a real-time search process that is also called the *pursuit process*. The goal of the searcher is to find or catch the target in minimal time. If the target is informed about the searcher's actions and its goal is to escape from the searcher, then the procedure corresponds to the *search game* or *pursuit-evasion game* [40–43].

Let us consider the values of control function c in particular. As indicated above, in case of perfect detection, the observation result is specified by an indicator function $\mathbf{1}$ so that $z^t = \mathbf{1}(A^t, x^t) \in \{0, 1\}$ for any time $t = 0, 1, 2, \dots$. At the initial step of the search, it is usually assumed that the searcher observes the complete locations set X , that is, $A^{t=0} = X$ (see Line 2 of Procedure 2.1) to verify that there is a target to be searched for in the set. If the target is not located at any point $x \in X$, then $z^{t=0} = \mathbf{1}(X, x) = 0$ and the corresponding control is $c^{t=0} = c(X, 0) = \text{terminate}$. In practice, however, the size of the observed areas is determined by the sensor characteristics and often restricted so that $|A| < |X|$. Thus, in most applications it is assumed that the target is indeed located at some point of X and the case when $\mathbf{1}(X, x) = 0$ is avoided. In the next steps of the search, termination of the condition can be obtained by different ways that follow the considered search task. In most cases, for perfect detection such a condition is specified as follows: if for an observed area A^t chosen at time t it follows that $|A^t| \leq m^*$, where $0 < m^* < n$ is a predefined size of the observed area, and $z^t = \mathbf{1}(A^t, x^t) = 1$, then $c^t = c(A^t, 1) = \text{terminate}$. For imperfect detection with $z^t = \mathbf{1}(A^t, x^t) \in [0, 1]$, similar conditions are checked with respect to some threshold value z^* , $0 \leq z^* \leq 1$. For example, for an initial search step, it is specified that $c^{t=0} = c(X, z^{t=0}) = \text{terminate}$ when $z^{t=0} = z(X, x) < z^*$, while for further steps it is determined that $c^t = c(A^t, z^t) = \text{terminate}$ when $z^t = z(A^t, x^t) \geq z^*$.

Hence, for a group-testing search with *perfect detection*, the control function c is defined by the following conditions:

$$c^t = c(A^t, z^t) = \begin{cases} \text{terminate} & \text{if } A^t = X \text{ and } z^t(A^t, x^t) = 0, \\ \text{terminate} & \text{if } |A^t| \leq m^* \text{ and } z^t(A^t, x^t) = 1, \\ \text{continue} & \text{otherwise.} \end{cases}$$

Similarly, in the case of *imperfect detection*, the control function c is defined by the use of the threshold value z^* :

$$c^t = c(A^t, z^t) = \begin{cases} \text{terminate} & \text{if } A^t = X \text{ and } z^t(A^t, x^t) < z^*, \\ \text{terminate} & \text{if } |A^t| \leq m^* \text{ and } z^t(A^t, x^t) \geq z^*, \\ \text{continue} & \text{otherwise.} \end{cases}$$

The threshold value z^* is often called the *threshold detection probability*. In certain cases, the search can be terminated also after a predefined number of unsuccessful detections or after reaching a predefined overall size of the observed part $\cup_t A^t$ of the set X .

Now let us consider the methods of selection of the observed areas $A^t \in \mathcal{X}$, $t = 0, 1, 2, \dots$. As indicated above, at the initial time $t = 0$ it is usually assumed that $A^{t=0} = X$, $z^{t=0} = 1$, and $c^{t=0} = \text{continue}$. Say at the time $t - 1$ an observed area A^{t-1} was chosen and an observation result z^{t-1} was obtained. Since the search was not terminated at $t - 1$, the control value $c^{t-1} = \text{continue}$. In the case of *offline search planning*, decision making regarding the observed area A^t is conducted according to the predefined *search plan* that determines the variants of choices with respect to the observation result at each time. In the case of *online search planning* or *real-time search*, in contrast, choice of the observed area A^t is based on the search history and is conducted according to some *search policy* that defines the decision-making process on the basis of the available information obtained in the previous search steps. As indicated above, the search history can be presented by a sequence $((A^{t-1}, z^{t-1}), (A^{t-2}, z^{t-2}), \dots, (A^{t=0}, z^{t=0}))$ of previously chosen observed areas and the corresponding observation results or by the results of a certain learning process.

To illustrate this procedure, let us consider a group-testing search with perfect detection, while the target is static and the searcher is free to choose any observed area $A^t \in \mathcal{X}$. The search follows the general group-testing procedure. The search plan is specified by a binary search tree $\mathcal{T} = (S, E)$, where S is a set of vertexes associated with selected search areas and E is a set of edges associated with the observation results. Thus the root node and each internal node of the tree are associated with a *test*, that is, a search in a particular subset of X . The result of the test splits this subset into two new subsets such that for at least one of them an observation of ‘0’ or ‘1’ can be obtained. The leaves of the tree correspond to the single-point subsets of X where the target can be located.

Example 2.8. Let $X = \{x_1, x_2, x_3, x_4\}$ and let a binary search tree \mathcal{T} with corresponding observed areas have the form shown in Figure 2.16.

In the figure, the first test splits the set $X = \{x_1, x_2, x_3, x_4\}$ into two disjoint and exhaustive subsets $A_1 = \{x_1, x_4\}$ and $A_2 = \{x_2, x_3\}$ of X , that is, $A_1 \cap A_2 = \emptyset$ and $A_1 \cup A_2 = X$. For each of these subsets the observation result ‘0’ or ‘1’ can be obtained. The next two tests are related to subsets A_1 and A_2 , correspondingly. The first of these tests results in the subsets $A_{11} = \{x_1\}$ and $A_{12} = \{x_4\}$, and the second in the subsets $A_{21} = \{x_2\}$ and $A_{22} = \{x_3\}$. Since each of these subsets includes only one point, they correspond to the leaves of the tree.

Assume that the target is located at the point $x = x_2$. Then, $z^{t=0} = 1$ and $c^{t=0} = \text{continue}$. According to the search tree shown in Figure 2.16, the searcher splits the set X into two observed areas A_1 and A_2 and obtains observation results $z(A_1, x) = 0$ and $z(A_2, x) = 1$. Thus $z^{t=1} = z(A_2, x) = 1$. Since A_2 is not a leaf, $c^{t=1} = \text{continue}$ and the searcher applies the test to the set A_2 . This set is split into the subsets A_{21} and A_{22} so

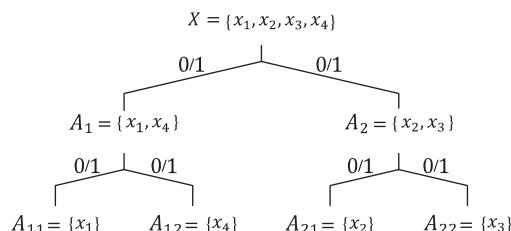


Figure 2.16 Example of binary search tree for static target search.

that $z(A_{21}, x) = 1$ and $z(A_{22}, x) = 0$. Thus $z^{t=2} = z(A_{21}, x) = 1$. Since the set A_{21} is a leaf, the control is $c^{t=2} = \text{terminate}$. The search terminates and the target is found in the single-point set $A_{21} = \{x_2\}$. Notice again that the considered search is applicable for a static target and unrestricted observed areas. ■

In the scenario of search with certain detection and static target considered in Example 2.8, each test splits the set into two disjoint subsets, so the observation of one of them unambiguously predicts the observation result of the other subset and the results of all future tests over such a subset. Following this remark, the group-testing algorithm that does not allow tests with predictable results is called *reasonable*, and for such algorithms the following facts hold [27].

Let $X = \{x_1, x_2, \dots, x_n\}$ be a set and let $\mathcal{T}[X]$ be a binary search tree that represents a reasonable group-testing algorithm over X . For each point $x \in X$, denote by $l(x)$ a number of edges from the root node, which is associated with the set X , to the leaf that is associated with the single-point set $\{x\}$, which includes the indicated point x . Then, for any reasonable group-testing algorithm it follows [27] that

$$\sum_{i=1}^n 2^{-l(x_i)} = 1. \quad (2.25)$$

Denote by $D(\mathcal{T}[X]) = \max_{x \in X} \{l(x)\}$ the depth of the binary search tree $\mathcal{T}[X]$. Then, for the search with certain detection, the goal of the searcher is to find a binary search tree $\mathcal{T}^*[X]$ such that its depth is minimized over all available binary search trees over X , that is, $D(\mathcal{T}^*[X]) = \min_{\mathcal{T}[X]} \max_{x \in X} \{l(x)\}$, while for the depth $D(\mathcal{T}^*[X])$ it follows [27] that

$$D(\mathcal{T}^*[X]) \geq \lceil \log(n) \rceil, \quad (2.26)$$

where $\lceil \cdot \rceil$ stand for the ceiling of the real number.

Note that the equality (2.25) applies to particular cases (e.g., for dyadic distributions) of the well-known Kraft inequality from information theory [35], while the right part of Equation 2.26 is an upper bound of the entropy of the set X . This observation invites implementations of information theory methods in group-testing search planning. In the next sections we consider the information theory approach in particular.

A group-testing algorithm that implements the above scenarios of search for a single static target is known as a *binary splitting algorithm* [27, 44, 45]. Given a locations set $X = \{x_1, x_2, \dots, x_n\}$, denote by $X^t \subseteq X$ a locations set that is considered at time t and assume that $X^{t=0} = X$. Then, in terms of Procedure 2.1, the binary splitting algorithm is outlined as follows.

Algorithm 2.4 (binary splitting). Given locations set X do:

1. If $\mathbf{1}(X, x) = 0$ then: Return \emptyset .
2. Set $t = 0$.
3. Set $X^t = X$.
4. While $|X^t| > 1$ do:

4.1 Split the set X^t into two observed areas A_1^t and A_2^t such that:

- a. $A_1^t \cap A_2^t = \emptyset$ and $A_1^t \cup A_2^t = X^t$,
- b. $||A_1^t| - |A_2^t|| \leq 1$.

4.2 Set $z_1^t = \mathbf{1}(A_1^t, x)$ and $z_2^t = \mathbf{1}(A_2^t, x)$.

4.3 If $z_1^t = 1$ then:

4.3.1. Set $X^{t+1} = A_1^t$.

Else:

4.3.2. Set $X^{t+1} = A_2^t$.

4.4. Set $t = t + 1$.

5. Return X^t . ■

Algorithm 2.4 is probably the most common algorithm of search for a single static target and, in the case of equivalent probabilities of the target location and non-restricted observed areas, it provides a suitable group-testing search scenario. However, if there are certain restrictions on the observed areas, for example, on their size and connectedness, or restrictions regarding their selection, for example, connectedness of the observed part $\bigcup_{\tau=0}^t A^\tau$ of the set X for any $t = 0, 1, 2, \dots$, then general algorithms are not applicable and the search planning is conducted by using specific approaches. Note that in the search and screening problem that was considered in Section 2.1, both requirements have to be implemented.

Let us now note a group-testing search for a moving target. Recall the relation between Algorithm 2.4 and its binary search tree $T[X]$. In the case of static target search, the subsets that correspond to the vertexes of each level of the tree are disjoint, while their union results in the complete set X . The systems of sets with the indicated property are called *set partition systems* [34]. Moreover, the subsets that correspond to the successors of each vertex form a set partition system for the set that corresponds to this vertex. For example, for the tree shown in Figure 2.16 there are two set partition systems: A_1 and A_2 at the first level; and A_{11}, A_{12}, A_{21} , and A_{22} at the second level. The subsets A_{11} and A_{12} form a set partition system for the set A_1 , and the subsets A_{21} and A_{22} form a set partition system for the set A_2 . In the case of a moving target search (MTS), in contrast, these properties do not hold: the subsets, which are located at the lower level of the search tree, do not form a set partition system of the corresponding set of the upper level, so the subsets that are located on the lower levels of the tree do not form a set partition system of the initial set X . To illustrate such a difference between the search trees, let us consider the same set X as in the previous example.

Example 2.9 Let $X = \{x_1, x_2, x_3, x_4\}$, and assume that at time $t = 0$ the target is located at the point $x^{t=0} = x_2$. In contrast to Example 2.8, assume that the target can now move to the neighboring point; that is, if $x^{t=0} = x_2$, then at time $t = 1$ the possible location of the target is $x^{t=1} \in \{x_1, x_3\}$, and so on. Moreover, assume that the target behaves according to Brownian motion, so it must move at each time step. The search tree in this case is illustrated in Figure 2.17. Note that this tree is not unique and is given for illustrative purposes only.

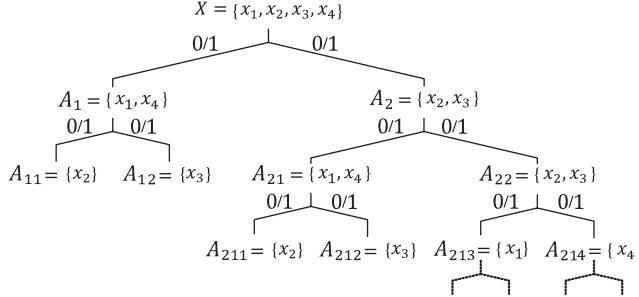


Figure 2.17 Example of binary search tree for a moving target search.

Following the presented tree, the searcher splits the set X into two observed areas A_1 and A_2 , and obtains observation results $z(A_1, x)$ and $z(A_2, x)$. Such a splitting is similar to the static target search scenario, yet the observation results are different. If the target moves from the initial point $x^{t=0} = x_2$ to the point $x^{t=1} = x_1$, then $z(A_1, x) = 1$, but if it moves to the point $x^{t=1} = x_3$, then $z(A_2, x) = 1$. Assume, first, that $z(A_1, x) = 1$ and $z(A_2, x) = 0$. Then the target is located either at the point x_1 or at the point x_4 , and at the next time $t = 2$ it can move to point x_2 or to point x_3 . Thus, the searcher chooses one of the observation areas $A_{11} = \{x_2\}$ and $A_{12} = \{x_3\}$ so that for the chosen area it is found that $z^{t=2} = 1$. Since both areas correspond to the leaves of the tree, the obtained control is $c^{t=2} = \text{terminate}$. The search terminates and the target is found in the single-point set $A_{11} = \{x_2\}$ or $A_{12} = \{x_3\}$. Now assume that, in contrast, $z(A_1, x) = 0$ and $z(A_2, x) = 1$. Then the target is located either at the point x_2 or at the point x_3 . Note that if the target move follows Brownian motion, then if $x^{t=0} = x_2$, the scenario $x^{t=1} = x_2$ is not allowed, but since the searcher is not informed about the initial target location this point has to be included in the search tree. According to the searcher's available information, at the next time $t = 2$ from point x_2 , the target can move either to the point x_1 or to the point x_3 , while from point x_3 it can move to point x_2 or to point x_4 . Following the search tree, the searcher proceeds to search areas $A_{21} = \{x_1, x_4\}$ and $A_{22} = \{x_2, x_3\}$. If $z(A_{21}, x) = 1$ and $z(A_{21}, x) = 0$, then the actions of the searcher are the same as in the top left branch of the tree, considered above. Assume that $z(A_{21}, x) = 0$ and $z(A_{21}, x) = 1$. Then, at time $t = 3$ the target can move to any point $x^{t=3} \in \{x_1, x_2, x_3, x_4\}$, while the chosen search areas are $A_{213} = \{x_1\}$ and $A_{214} = \{x_4\}$ and both correspond to the leaves. Hence, if $z(A_{213}, x) = 1$ or $z(A_{214}, x) = 1$, then $c^{t=3} = \text{terminate}$ and the target is found in one of the chosen areas; otherwise the control is set to $c^{t=3} = \text{continue}$ and the searcher has to conduct the next observations. ■

The group-testing search procedure and corresponding scenarios of search for static and moving targets represent a general approach to offline search planning and to the online search scheme. In both cases, the observed areas are not restricted and no additional information regarding the target locations has been used. However, in most applications it is assumed that, similar to the search and screening approach (see Section 2.1), certain probabilities of target detection, as well as its transition probabilities, can be defined and used during the search. In the next section, we consider tasks that implement such an approach for the group-testing search.

2.2.2 Combinatorial group-testing search for static targets

Let us consider the search that follows the group-testing approach for detecting defective members in large populations [24]. As indicated in Section 2.2, this approach was developed to address the need for a procedure for testing blood samples for the presence of an antigen. Similar group-testing techniques are applicable for any group-testing search for a *single static target* or a *number of static targets*, while the available observed areas are unrestricted. The procedures are not optimal in comparison to information theory procedures and are presented to give suitable intuition regarding combinatorial methods of group-testing search and of search for an unknown number of targets.

Let, as indicated above, $X = \{x_1, x_2, \dots, x_n\}$ be a set of possible target locations and assume that the target is located at some point of X , that is, $\mathbf{1}(X, x) = 1$. In contrast to previous considerations, let us allow that more than one target can be located at X , while *the strict number of targets is unavailable to the searcher*.

Assume that the detection is errorless, that is, as indicated above, for any time $t = 0, 1, 2, \dots$ it follows that $\psi^t(x) = \Pr[\text{target is detected at } x | \text{target is located at } x] = 1$, and define a probability p of detecting the target at the randomly chosen point $x \in X$, that is, $p = \Pr[\text{target is detected at } x | x \text{ is chosen randomly}]$. Such a probability depends on a priori knowledge regarding the locations set X and on the number of targets. Corresponding to the search and screening methods (see Section 2.1), this probability can be defined by equivalent location probabilities $p'(x_i)$ at a fixed time $p = p'(x_i) = 1/n$, $i = 1, \dots, n$. Such location probabilities represent the worst case, in which a priori information regarding the target location is not available. Such an assumption follows from the maximum entropy principle [35]. In addition, denote by $q = 1 - p$ the probability of not detecting the target at a randomly chosen point.

Let $\mathcal{X} = 2^X$ be the set of all possible subsets of X that are considered as observed areas. We focus on areas $A \in \mathcal{X}$ of some fixed size $|A| = m$, $1 \leq m \leq n$. The Dorfman group-testing procedure includes three main actions, outlined as follows [24].

Procedure 2.2 (Dorfman group-testing search):

1. Split the set X into the areas $A_j \in \mathcal{X}$, $|A_j| = m$, $1 \leq j \leq n/m$.
2. Observe each area A_j and obtain observation result $z_j = \mathbf{1}(A_j, x)$.
3. For each area A_j such that $z_j = 1$, observe points $x_{jk} \in A_j$, $1 \leq k \leq m$, individually and obtain an observation result $z_{jk} = \mathbf{1}(\{x_{jk}\}, x)$.
4. Return all points for which $z_{jk} = 1$. ■

In the last step, for each observed area A_j one obtains exactly m observations of the single-point sets $\{x_i\}$, $x_i \in A_j$. The expected total number of observations $E_D(z|n, m)$ is defined as follows [24]. The probability that the target will not be detected in a randomly chosen observed area A of size m is $Q_m^A = q^m = (1 - p)^m$ and the corresponding probability of finding the target in the area A is $P_m^A = 1 - Q_m^A = 1 - (1 - p)^m$. Then

$$E_D(z|n, m) = n/m + m(n/m) P_m^A = n/m + n(1 - (1 - p)^m), \quad (2.27)$$

where the first addendum represents a number of observations of the areas $A \in \mathcal{X}$, $|A| = m$, and the second addendum represents an expected number of points in the observed areas

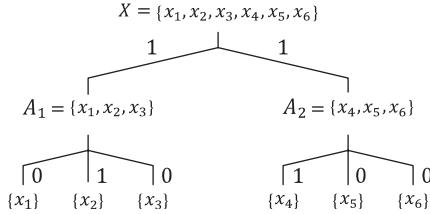


Figure 2.18 Example of the Dorfman group-testing search tree.

A that have to be observed individually. The actions of the Dorfman group-testing search are illustrated by the following example.

Example 2.10 Let $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and assume that there are two targets located at X : one target is located at point x_2 and the other at point x_4 . Let the size of the observed areas be $m = 3$, and let the searcher start by splitting the set X into two disjoint areas $A_1 = \{x_1, x_2, x_3\}$ and $A_2 = \{x_4, x_5, x_6\}$. Following the Dorfman group-testing procedure, the searcher obtains observation results $z(A_1, x) = 1$ and $z(A_2, x) = 1$. Then, the search continues by observing the points $x \in A_1$ and $x \in A_2$ individually, and, accordingly, obtaining observation results $z(\{x_2\}, x) = 1$ and $z(\{x_4\}, x) = 1$, while for the remaining points the observation results are zero. The search tree for the considered scenario is shown in Figure 2.18.

The number of observations $E_D(z|n, m)$ that is required by the procedure for the search with $n = 6$ and $m = 3$ is $E_D(z|6, 3) = 8$. ■

Note that in the Dorfman procedure the size m of the observed areas $A \in \mathcal{X}$ and their content are predefined and do not change during the search. Sterrett [25, 26] modified the Dorfman procedure and suggested a group-testing procedure in which the size of the chosen observed areas and the included points depend on the observation results. The Sterrett group-testing procedure uses the same set \mathcal{X} of possible observed areas $A \subset X$. As indicated above, denote by $X^t \subseteq X$ and by \mathcal{X}^t the corresponding power set of X^t , and assume that $X^{t=0} = X$ and $\mathcal{X}^{t=0} = \mathcal{X}$. The size of the sets X^t is denoted by $n(t) = |X^t|$ and the size of the corresponding observed areas $A \subset X^t$ is denoted by $m(t) = |A|$. Then the Sterrett procedure is outlined as follows [25, 26].

Procedure 2.3 (Sterrett group-testing search):

1. Set $t = 0$.
2. Split the set X^t into the areas $A_j \in \mathcal{X}^t$, $|A_j| = m(t)$, $1 \leq j \leq \lceil n(t)/m(t) \rceil$.
3. Observe each area A_j and obtain observation result $z_j = \mathbf{1}(A_j, x)$.
4. For each search area A_j such that $z_j = 1$ do:
 - 4.1 Observe points $x_k \in A_j$ sequentially, $1 \leq k \leq m$, up to the first point, for which the observation results $z_{jk} = \mathbf{1}(\{x_{jk}\}, x) = 1$ are obtained.
 - 4.2 Add the remaining points to the next set X^{t+1} .

5. Set $t = t + 1$.
6. If $|X^t| > 0$ then continue search starting from Line 2.
7. Return all points for which $z_{jk} = 1$. ■

The expected number of observations required by the Sterrett procedure is specified as follows. Denote by $p(m, v)$ the probability that the observed area A of size m includes v targets and by $E_S(z|m, v)$ the expected number of observations required to find v targets in the set of size m . Then the expected number of observations required by Procedure 2.3 is given by the following equation [25, 26]:

$$E_S(z|n, m) = (n/m) \sum_{v=0}^m p(m, v) E_S(z|m, v), \quad (2.28a)$$

where

$$E_S(z|m, v) = (m+1) \frac{v}{v+1} + \tau + 1 - 2 \frac{v}{m}. \quad (2.28b)$$

Note that if $v = 0$, then $E_S(z|m, 0) = 1$, corresponding to the fact that there is enough to conduct a single observation to verify that there is no target in the observed area A . The actions of the Sterrett group-testing procedure are illustrated by the following example.

Example 2.11 Let, as in the previous example, $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ with two targets located at points x_2 and x_4 . Let, as indicated above, the size of the observed areas be $m = 3$. Again, the searcher starts by splitting the set X into two subareas $A_1 = \{x_1, x_2, x_3\}$ and $A_2 = \{x_4, x_5, x_6\}$. Since both $z(A_1, x) = 1$ and $z(A_2, x) = 1$, then, following the Sterrett group-testing procedure, the searcher starts observing the points $x \in A_1$ and $x \in A_2$ individually, until the first points, x_2 and x_4 , are found with an observation result of ‘1.’ The remaining points are united in the new observed area $\{x_3, x_5, x_6\}$. Since $z(\{x_3, x_5, x_6\}, x) = 0$ the search terminates. The searcher’s choices are illustrated in Figure 2.19.

Note that in this case the obtained graph of the searcher’s observed areas is not a tree. The number of observations $E_S(z|n, m)$ required by the procedure for the considered search with $n = 6$ and $m = 3$ is $E_S(z|6, 3) = 6$. ■

Finally, let us implement the Dorfman and Sterrett procedures in search for a single target $v = 1$, following the assumption that the location probabilities are equivalent.

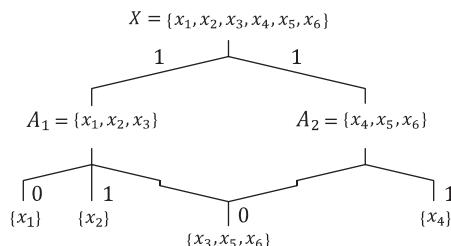


Figure 2.19 Example of the Sterrett group-testing search graph.

In the Dorfman procedure, the probability of detecting the target at a randomly chosen point $x \in X$ is $p = 1/n$. Thus, by using Equation 2.27 we obtain the following expected number of observations:

$$E_D(z|n, m) = \frac{n}{m} + n(1 - (1 - p)^m) = \frac{n}{m} + n\left(1 - \left(\frac{n-1}{n}\right)^m\right).$$

In particular, for Example 2.10, where $n = 4$ and $m = 2$, we obtain

$$E_D(z|4, 2) = \frac{4}{2} + 4\left(1 - \left(\frac{4-1}{4}\right)^2\right) = 3.75.$$

In the Sterrett procedure, according to the assumption regarding location probabilities, the probability of finding exactly one target in the observed area A , $|A| = m$, is $p(m, 1) = 1/m$. Thus, the probability of not finding the target is $p(m, 0) = 1 - 1/m$, and the probability of finding more than one target is $p(m, v > 1) = 0$.

As indicated above, $E_S(z|m, 0) = 1$. Since $p(m, v > 1) = 0$, all addendums in Equation 2.28a that include this value are zero, and we need only the value $E_S(z|m, 1)$. According to Equation 2.28b, we obtain

$$E_S(z|m, v) = (m+1) \frac{v}{v+1} + v+1 - 2\frac{v}{m} = \frac{m^2 + 5m - 4}{2m}.$$

Substitution of this value and probabilities $p(m, 0)$ and $p(m, 1)$ into Equation 2.28a gives the following expected number of observations:

$$\begin{aligned} E_S(z|n, m) &= \frac{n}{m} (p(m, 0) E_S(z|m, 0) + p(m, 1) E_S(z|m, 1)) \\ &= \frac{n}{m} \left(\left(1 - \frac{1}{m}\right) + \frac{1}{m} \frac{m^2 + 5m - 4}{2m} \right) = \frac{n}{2m^3} (3m^2 + 3m - 4). \end{aligned}$$

Similar to the previous calculation in Example 2.11, $n = 4$ and $m = 2$; thus

$$E_S(z|4, 2) = \frac{4}{2 \times 2^3} (3 \times 2^2 + 3 \times 2 - 4) = 3.50.$$

For small locations sets X , the Sterrett procedure is not necessarily better than the Dorfman procedure. In fact, for $n = 6$ and $m = 2$ the expected number of observations in the Dorfman procedure is $E_D(z|6, 2) = 4.83$, while in the Sterrett procedure $E_S(z|6, 2) = 5.25$. However, for relatively large locations sets, the Sterrett procedure demonstrates better results than the Dorfman procedure. For example, $E_D(z|10, 3) = 6.04$, while $E_S(z|10, 3) = 5.93$, and $E_D(z|100, 10) = 19.56$, while $E_S(z|100, 10) = 16.30$.

Note again that in the Dorfman and Sterrett procedures the searcher is not informed about the number of the targets. Hence, the search continues after finding any number of targets, and the expected number of observations in both procedures is far from the corresponding lower bound as specified by the inequality in Equation 2.26.

The considered group-testing procedures implement *online search for a static target*, in which the size of the observed areas is not strictly determined (that is why both methods of group-testing search are indicated as procedures rather than algorithms). Let us consider Hwang's simple algorithm [27] which specifies these values and governs a combinatorial

group-testing search. This algorithm is known as the *generalized binary splitting algorithm*. It follows the general group-testing search in Procedure 2.1 (see Section 2.2.1) with omitted target's turn and implements an idea of the Sterrett procedure, Procedure 2.3. However, note that in contrast to the considered Dorfman and Sterrett procedures, execution of Hwang's algorithm requires knowledge of the number τ of targets that are located in the set X .

Let, as above, $X = \{x_1, x_2, \dots, x_n\}$ be the allocations set and let v , $1 \leq v \leq n$, be the number of targets located at X . Then, $X^t \subseteq X$ stands for a set that is considered at time t and $X^{t=0} = X$. Hwang's generalized binary splitting algorithm implements a search after a number v of static targets and is outlined as follows.

Algorithm 2.5 (generalized binary splitting) [27]. Given locations set X and a number of targets $1 \leq v \leq n$ do:

1. If $\mathbf{1}(X, x) = 0$ then: Return.
2. Set $t = 0$.
3. Set $X^t = X$.
4. Initialize resulting set of points: Set $Res = \emptyset$.
5. While $|X^t| > 2v - 2$ and $v > 0$ do:
 - 5.1 Set $h = \lfloor \log((|X^t| - v + 1)/v) \rfloor$, where $\lfloor \cdot \rfloor$ stands for the floor of a real number.
 - 5.2 Split the set X^t into two observed areas A_1^t and A_2^t such that:
 - a. $A_1^t \cap A_2^t = \emptyset$ and $A_1^t \cup A_2^t = X^t$,
 - b. $|A_1^t| = 2^h$; thus $|A_2^t| = |X^t| - 2^h$.
 - 5.3 Set $z_1^t = \mathbf{1}(A_1^t, x)$.
 - 5.4 If $z_1^t = 1$ then:
 - 5.4.1 Set $\{x^t\} = \text{Algorithm 2.4 } (A_1^t)$ and save observed areas $A \subset A_1^t$, for which Algorithm 2.4 obtains observation results $\mathbf{1}(A, x) = 0$.
 - 5.4.2 Set $X^{t+1} = A_1^t \setminus \{x^t\} \setminus (\cup A)$, where $A \subset A_1^t$ have been saved at Line 5.4.1.
 - 5.4.3 Set $v = v - 1$.
 - 5.4.4 Set $Res = Res \cup \{x^t\}$.
Else:
 - 5.4.5 Set $X^{t+1} = A_2^t$.
 - 5.5 Set $t = t + 1$.
6. For each $i = 1, \dots, |X^t|$ do (observe points $x \in X^t$ sequentially):
 - 6.1 Set $z_i^t = \mathbf{1}(\{x_i\}, x)$.
 - 6.2 If $z_i^t = 1$ then: Set $Res = Res \cup \{x_i\}$.
7. Return Res . ■

This algorithm generalizes Algorithm 2.4 and is reduced to it in the case of search for a single target, that is, while $v = 0$. Denote by $E_H(z|n, v)$ an expected number of observations required by Algorithm 2.5 for finding v targets in the set of size n . The value $E_H(z|n, v)$ is specified by the following theorem.

Theorem 2.6 [27]. *Given a locations set X of size $n = |X|$ and a number of targets v , $1 \leq v \leq n$, denote $h = \lfloor \log((n - v + 1)/v) \rfloor$. Then the depth of the binary search tree that corresponds to the action of Algorithm 2.5 is defined as follows:*

$$E_H(z|n, v) = \begin{cases} n & \text{if } n \leq 2v - 2, \\ (h+2)v + \epsilon - 1 & \text{otherwise,} \end{cases}$$

where integer number ϵ , $0 \leq \epsilon < v$, is determined by the equation $n - v + 1 = 2^h v + 2^h \epsilon + \theta$, where θ is an integer number such that $0 \leq \theta < 2^h$.

Proof. Let us consider the first condition $n \leq 2v - 2$. In this case, $E_H(z|n, v) = n$ follows directly from sequential observations of n points as defined in Line 6.

The second condition includes two particular cases and a general induction over a number of targets and size of the locations set. At first, let us consider two particular cases.

1. If $2v - 1 \leq n \leq 3v - 2$, then $1 \leq (n - v + 1)/v \leq 2 - 1/v$. Thus $0 \leq \log((n - v + 1)/v) \leq 1$ and $h = \lfloor \log((n - v + 1)/v) \rfloor = 0$. Hence, according to the choice of the observed areas in Line 5.2, the algorithm conducts sequential observations. According to the statement of the theorem, for $h = 0$ it follows that $\theta = 0$ and so $\epsilon = n - 2v + 1$. Substitution of these values into the second condition results in the following number of observations:

$$E_H(z|n, v) = (h+2)v + \epsilon - 1 = 2v + n - 2v + 1 - 1 = n.$$

2. If $v = 1$, then direct substitution of this value into the corresponding lines of the algorithm shows that it is reduced to the binary splitting. According to the condition $0 \leq \epsilon < v$, for $v = 1$ it follows that $\epsilon = 0$. Thus, the number of observations is as follows:

$$E_H(z|n, 1) = (h+2)v + \epsilon - 1 = h + 1 = \lceil \log(n) \rceil + 1 = \lceil \log(n) \rceil,$$

corresponding to Equation 2.26.

Now let us apply an induction for $v > 1$ and $n > 3v - 2$. According to Lines 5.1–5.2 and further choice, the number of observations is

$$E_H(z|n, v) = \max \{1 + E_H(z|n - 2^h, v), 1 + h + E_H(z|n - 1, v - 1)\}.$$

Hence, it is required to consider the terms of the max operator.

Let $m = n - v + 1$ and consider the number $E_H(z|n - 2^h, v)$. For $n' = n - 2^h$, we obtain $m' = n' - v + 1 = m - 2^h = 2^h v + 2^h \epsilon + \theta - 2^h$, which for different values of ϵ and θ can be written as follows:

- if $\epsilon \geq 1$ then $m' = 2^h v + 2^h(\epsilon - 1) + \theta$;
- if $\epsilon = 0$ and $\theta < 2^{h-1}$ then

$$m' = 2^h v + \theta - 2^h = 2^{h-1} v + 2^{h-1} v + \theta - 2 \times 2^{h-1} = 2^{h-1} v + 2^{h-1}(v - 2) + \theta;$$

- if $\epsilon = 0$ and $\theta \geq 2^{h-1}$ then

$$\begin{aligned} m' &= 2^h v + \theta - 2^h = 2^{h-1} v + 2^{h-1} v + \theta - 2 \times 2^{h-1} \\ &= 2^{h-1} v + 2^{h-1} v + \theta - 2^{h-1} - 2^{h-1} \\ &= 2^{h-1} v + 2^{h-1}(v - 1) + (\theta - 2^{h-1}). \end{aligned}$$

Hence,

$$E_H(z|n - 2^h, v) = \begin{cases} (h+2)v + (\epsilon - 1) - 1, & \epsilon \geq 1, \\ (h+1)v + (\epsilon - 2) - 1, & \epsilon = 0 \text{ and } \theta < 2^{h-1}, \\ (h+1)v + (\epsilon - 1) - 1, & \epsilon = 0 \text{ and } \theta \geq 2^{h-1}, \end{cases}$$

and the addition of unity results in the following value:

$$1 + E_H(z|n - 2^h, v) = \begin{cases} (h+2)v + \epsilon - 2, & \epsilon = 0 \text{ and } \theta < 2^{h-1}, \\ (h+2)v + \epsilon - 1, & \text{otherwise.} \end{cases}$$

Let us consider the second term, $E_H(z|n - 1, v - 1)$. As above, for $n' = n - 1$ and $v' = v - 1$ we obtain $m' = n' - v' + 1 = (n - 1) - (v - 1) - 1 = m$. Then:

- if $\epsilon \leq v - 3$ then

$$m' = 2^h v + 2^h \epsilon + \theta = 2^h v - 2^h + 2^h \epsilon + 2^h + \theta = 2^h(v - 1) + 2^h(\epsilon + 1) + \theta;$$

- if $\epsilon = v - 2$ then

$$m' = 2^h v + 2^h \epsilon + \theta = 2^h v + 2^h(v - 2) + \theta = 2^{h+1}(v - 1) + \theta; \text{ and}$$

- if $\epsilon = v - 1$ then

$$m' = 2^h v + 2^h \epsilon + \theta = 2^h v + 2^h(v - 1) + \theta = 2^{h+1}(v - 1) + 2^h + \theta.$$

Hence,

$$E_H(z|n - 1, v - 1) = \begin{cases} (h+2)(v - 1) + (\epsilon + 1) - 1, & \epsilon \leq v - 3, \\ (h+3)(v - 1) - 1, & v - 2 \leq \epsilon \leq v - 1, \end{cases}$$

and the addition of $(1 + h)$ results in the following value:

$$1 + h + E_H(z|n - 1, v - 1) = \begin{cases} (h+2)v + \epsilon - 2, & \epsilon = v - 1, \\ (h+2)v + \epsilon - 1, & \text{otherwise.} \end{cases}$$

Finally, note that if $\epsilon \neq 0$ and $\epsilon \neq v - 1$, then both the obtained values are equal, that is, $1 + E_H(z|n - 2^h, v) = 1 + h + E_H(z|n - 1, v - 1) = (h + 2)v + \epsilon - 1$, while for a number of targets $v \geq 2$, the cases $\epsilon = 0$ and $\epsilon = v - 1$ are mutually exclusive. Hence,

$$\begin{aligned} E_H(z|n, v) &= \max \{1 + E_H(z|n - 2^h, v), 1 + h + E_H(z|n - 1, v - 1)\} \\ &= (h + 2)v + \epsilon - 1 \end{aligned}$$

as required by the statement of the theorem. ■

From Theorem 2.6 it follows that the relation between the tree depth $E_H(z|n, v)$ and a lower bound $\left\lceil \log \binom{n}{v} \right\rceil$ of the number of search steps is specified by the following inequality [27]:

$$E_H(z|n, v) - \left\lceil \log \binom{n}{v} \right\rceil \leq v - 1, \text{ while } v \geq 2.$$

In fact, if $m = n - v + 1$ then

$$\binom{n}{v} = \binom{m + v - 1}{v} > \frac{m^v}{v!} = \frac{(2^h v + 2^h \epsilon + \theta)^v}{v!} \geq \frac{(2^h v (1 + \epsilon/v))^v}{v!} \geq 2^{(h+1)v+\epsilon-1}.$$

Hence,

$$\left\lceil \log \binom{n}{v} \right\rceil > (h + 2)v + \epsilon - 1,$$

and the indicated inequality holds.

Note that in contrast to search for a single static target, the expected number of observations $E_H(z|n, v)$ in general is not equivalent to the depth $D(\mathcal{T}[X]|n, v)$ of the corresponding binary search tree. The illustration of this difference along with consideration of the execution of Algorithm 2.4 over a small locations set X are presented in the following example.

Example 2.12 Let, as above, $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and let $v = 2$ targets be located at the points x_2 and x_4 . At the initial time $t = 0$, it follows that $v = 2 > 0$ and $|X^{t=0}| = |X| = 6 > 2v - 2 = 2$. Thus the group-testing search according to Lines 5.1–5.5 of Algorithm 2.5 is applied. For $|X^{t=0}| = 6$ and $v = 2$ we have

$$h = \left\lceil \log \left(\frac{|X'| - v + 1}{v} \right) \right\rceil = \left\lceil \log \left(\frac{6 - 2 + 1}{2} \right) \right\rceil = \left\lceil \log \left(\frac{5}{2} \right) \right\rceil = 1.$$

Hence, the size of the area $A_1^{t=0}$ is $2^h = 2^1 = 2$ and the size of the complementary area $A_2^{t=0}$ is $|X^{t=0}| - 2^h = 6 - 2 = 4$. Assume that $A_1^{t=0} = \{x_1, x_2\}$ and $A_2^{t=0} = \{x_3, x_4, x_5, x_6\}$. Since one of the targets is located at point x_2 , the observation result is $z_1^{t=0} = \mathbf{1}(A_1^{t=0}, x) = 1$, and a binary splitting according to Algorithm 2.4 is applied to the area $A_1^{t=0}$.

After finding the target at point x_2 , the number of targets v decreases to $v = 1$. From that point, the algorithm is applied to the area $A_2^{t=0}$, which is considered as $X^{t=1} = \{x_3, x_4, x_5, x_6\}$. As above, for $|X^{t=1}| = 4$ and $v = 1$ it holds that

$$h = \left\lceil \log \left(\frac{|X'| - v + 1}{v} \right) \right\rceil = \left\lceil \log \left(\frac{4 - 1 + 1}{2} \right) \right\rceil = \left\lceil \log \left(\frac{4}{2} \right) \right\rceil = 1.$$

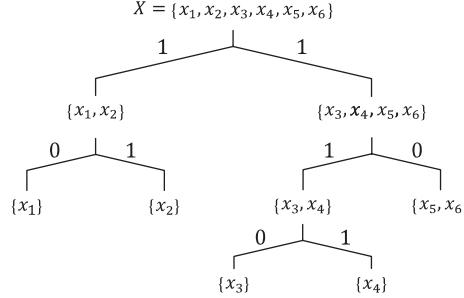


Figure 2.20 Example of binary search tree for Hwang's algorithm.

Thus, the size of the area $A_1^{t=1}$ is $2^h = 2^1 = 2$ and the size of the complementary area $A_2^{t=1}$ is $|X^{t=1}| - 2^h = 4 - 2 = 2$.

Let $A_1^{t=1} = \{x_3, x_4\}$ and $A_2^{t=1} = \{x_5, x_6\}$. Hence, $z_1^{t=1} = \mathbf{1}(A_1^{t=1}, x) = 1$ and over the area $A_1^{t=1}$ a binary splitting is applied. After finding the target at point x_4 , the number of targets v decreases to $v = 0$ and the search terminates. The search tree for this scenario is shown in Figure 2.20.

Note that the depth $D(\mathcal{T}[X]|n, v)$ of the tree is $D(\mathcal{T}[X]|6, 2) = 4$, while the number of observations is

$$\begin{aligned} E_H(z|n, v) &= E_H(z|6, 2) = \left(\left\lceil \log \left(\frac{6-2+1}{2} \right) + 2 \right\rceil \right) 2 + \epsilon - 1 \\ &= 6 + \epsilon - 1 = 5 + \epsilon, \quad 0 \leq \epsilon < 2. \end{aligned}$$

In this scenario, $\epsilon = 0$ and the number of observations is $E_H(z|6, 2) = 5$. ■

In the group-testing procedures, the search both for a single and for multiple targets is conducted under the assumption that detection is errorless, that is, the observation $z(A, x)$ of any observed area $A \subset X$ results in ‘1’ if $x \in A$ and ‘0’ if $x \notin A$. Now let us consider the procedures that allow erroneous observations.

2.2.3 Search with unknown number of targets and erroneous observations

Given a locations set X and a set $\mathcal{X} = 2^X$ of all possible subsets of X , assume that the searcher chooses an observed area $A \in \mathcal{X}$ and that the erroneous observations are such that $z(A, x) \in [0, 1]$ is allowed. Then, there are two possible observation errors: false negative observations, which correspond to the erroneous decision that the target is not in the observed area A while, in fact, the target is located at this point, that is, $z(A, x) < 1$ if $x \in A$; and false positive observations, which represent the erroneous decision that the target is in the observed area A while, in fact, the target is not located at this point, that is, $z(A, x) > 0$ if $x \notin A$. As indicated above, the case of false negative observations is defined by using detection probability function ψ^t (see Section 2.1) as $z^t = z(A^t, x) = \psi^t(A^t) = \sum_{x \in A^t} \psi^t(x)$, while $\psi^t(x) = \Pr\{\text{target is detected at } x | \text{target is located at } x\}$, $t = 0, 1, 2, \dots$. The corresponding group-testing procedure with false negative tests was suggested by Graff and Roeloffs [46] on the basis of Sobel and Groll's algorithm [47]. Later, Gupta and Malina

[29] built a scenario that implements false positive observations; however, such a scenario disagrees with the assumption in Equation 2.2 which omits the appearance of false targets in the search and screening approach. Below, we consider the algorithms by Sobel and Groll, and Graff and Roeloffs, in the context of the problem of search for a number of static targets located in the finite set.

Let us start with Sobel and Groll's basic algorithm [47], in which it is assumed that the observations are errorless, while the number of targets is unknown. In the context of the terms considered above, such an assumption implies that, similar to the search for a moving target, the set partition systems for the locations set X cannot be obtained. As in the Dorfman and Sterrett procedures (Procedures 2.2 and 2.3), denote by p the probability of detecting the target at a randomly chosen point $x \in X$, and by $q = 1 - p$ the corresponding non-detection probability. As above, let v stand for the number of targets, but, in contrast to the previous consideration, v is considered as a random variable.

Let $A \in \mathcal{X}$, $A \subset X$, be the observed area. We say that the observed area A is *empty*, that is, it does not contain the targets, if

$$\Pr\{v = 0 | \text{observed area is } A\} = 1, \quad (2.29a)$$

and that the observed area A is *filled* if

$$\Pr\{v = 0 | \text{observed area is } A\} = 0. \quad (2.29b)$$

If, in addition, the number of targets v in the area A satisfies the binomial distribution

$$\Pr\{v = y | \text{observed area is } A\} = \binom{m}{y} p^y q^{m-y} / (1 - q^m), \quad y = 1, 2, \dots, m, \quad (2.30)$$

then the filled area is called the *binomial area* [47].

Sobel and Groll's algorithm is based on the following lemma. For some event D , denote by $z(A, x|D)$ a result of observation of the area A with respect to the event D . Assume that $z(A, x|v \geq y) = 1$ and $z(A, x|v < y) = 0$, where $y, 1 \leq y \leq m = |A|$, is some fixed number of targets.

Let $A_0 \subset X$ be an area and denote by $A_j \subset X$ not necessarily mutually disjoint areas such that $A_0 \cap A_j = \emptyset$ for any $j = 1, 2, \dots, k$. Denote by v_0 and v_j random numbers of targets located in the areas A_0 and A_j , respectively, $j = 1, 2, \dots, k$.

Lemma 2.2 [47]. *Assume that A_0 and $A_0 \cup A_j$, $j = 1, 2, \dots, k$, are a priori binomial areas, and let both $z(A_0, x|v_0 \geq y_0) = 1$, $y_0 \geq \max_j \{y_j\}$, and $z(A_0 \cup A_j, x|v_0 + v_j \geq y_j) = 1$, $j = 1, 2, \dots, k$. Then, a posterior distribution on the areas A_j , $j = 1, 2, \dots, k$, is also binomial.*

Proof. Let us fix some number of targets $y \geq 1$. Then, for each area A_j we are interested in the a posteriori probability that $v_j \leq y$, $j = 1, 2, \dots, k$, given the indicated observation results. Let us consider this probability $P = \Pr\{v_j \leq y | v_0 + v_i \geq y_i, v_0 \geq y_0\}$, $i = 1, 2, \dots, k$. Since $y_0 \geq \max_j \{y_j\}$, the condition $v_0 + v_i \geq y_i$ is a direct consequence of the last condition $v_0 \geq y_0$ that results in $P = \Pr\{v_j \leq y | v_0 \geq y_0\}$. From the assumption that $A_0 \cap A_j = \emptyset$, $j = 1, 2, \dots, k$, it follows that the events $\{v_j \leq y\}$ and $\{v_0 \geq y_0\}$ are

independent. Hence $P = \Pr\{\nu_j \leq y\}$; that is, the posterior distribution on the areas A_j , $j = 1, 2, \dots, k$, does not depend on the observation results for A_0 and $A_0 \cup A_j$, and is equivalent to the a priori binomial distributions. ■

In other words [27], Lemma 2.2 states that if for some area A both $A' \subset A$ and $A'' \subset A'$ are observed as filled, then the points from the areas $A \setminus A'$ and $A' \setminus A''$ can be mixed without losing information.

According to Lemma 2.2, the results of the observations do not change the distribution of the number of targets in the remaining areas that allows an application of the same procedures at each step of the search. In Sobel and Groll's algorithm, this property is utilized for recursively calculating the size of the area that has to be observed at the next step of the search. The size of the next area is obtained as a solution of the minimization problem on the expected number of observations. The recursive equations are defined as follows [47].

Let X be a locations set of size n and let, as above, q be a constant a priori probability of not detecting any of the targets at the points in the locations set X before starting the search, so $p = 1 - q$. Assume that the area $A \subset X$ of size m has been observed as filled, that is, an observation result over this area is $z(A, x) = 1$, and that the area $X \setminus A$ of size $n - m = |X \setminus A|$ is binomial. Denote by $E_{SG}(z|m, n)$ the expected number of observations that are required to find all locations that are occupied by the targets in the area A , $|A| = m$, that is, a nested set of the set X , $|X| = n$. In addition, let $E_{SG}^0(z|n) = E_{SG}(z|0, n)$. Then, the values of the expected number of observations are specified by the following recursive equations:

$$\begin{aligned} E_{SG}^0(z|n) &= 1 + \min_{1 \leq y \leq n} \left\{ q^y \times E_{SG}^0(z|n - y) + (1 - q^y) \times E_{SG}(z|y, n) \right\}, \\ E_{SG}^0(z|0) &= 0 \end{aligned} \quad (2.31a)$$

$$E_{SG}(z|m, n) = 1 + \min_{1 \leq y \leq m-1} \left\{ \frac{q^y - q^m}{1 - q^m} \times E_{SG}(z|m - y, n - y) + \frac{1 - q^y}{1 - q^m} \times E_{SG}(z|y, n) \right\},$$

$$E_{SG}(z|1, n) = E_{SG}(z|n - 1). \quad (2.31b)$$

The integer number y that, given the sizes m and n solves the recursions (Equations 2.31), specifies the size of the area that has to be observed at the next step.

Let us define a function $next_size(m, n, q)$ that, according to the recursions (Equations 2.31), returns a size y of the next observed area for the arguments m , n , and q :

$next_size(m, n, q)$:

1. If $m \leq 0$ then:

1.1. Calculate $E_{SG}^0(z|n)$ according to (Equation 2.31a).
Else

1.2. Calculate $E_{SG}(z|m, n)$ according to (Equation 2.31b).

2. Return size y .

Then, according to the example presented by Sobel and Groll [47] and the outline provided by Du and Hwang [27], the algorithm of group-testing search with errorless observations, which is also known as the R_1 procedure, includes the following actions.

Algorithm 2.6 (Sobel and Groll group-testing search) [27, 47]. Given locations set X and probability q do:

1. Set $t = 0$.
2. Initialize resulting set of points: Set $Res = \emptyset$.
3. Set $X^t = X$ and $n = |X^t|$.
4. Set $A^t = \emptyset$ and $m = |A^t| = 0$.
5. Do:
 - 5.1. Set $y = next_size(m, n, q)$.
 - 5.2. Choose an observed area $A^{t+1} \subset X^t$ such that $|A^{t+1}| = y$.
 - 5.3. Set $z^{t+1} = \mathbf{1}(A^{t+1}, x)$.
 - 5.4. If $z^{t+1} = 0$ then:
 - 5.4.1. If $y = n$ then:
 - a. Break.
 - Else:
 - b. Set $X^{t+1} = X^t \setminus A^{t+1}$ and $n = |X^{t+1}|$.
 - c. Set $m = m - y$ and $n = n - y$.
 - Else:
 - 5.4.2. If $y = 1$ then:
 - a. Set $Res = Res \cup A^{t+1}$.
 - b. Set $X^{t+1} = A^{t+1}$ and $n = |X^{t+1}|$.
 - c. Set $A^{t+1} = \emptyset$ and $m = |A^{t+1}| = 0$.
 - Else:
 - d. Set $m = y$ and remain previous value of n .
- 5.5. Set $t = t + 1$.
- While $n \geq 1$.
6. Return Res .

■

The actions of the algorithm are illustrated by the following example.

Example 2.13 Assume that locations set X includes $n = |X| = 6$ points and that the probability of not detecting any one of the targets is $q = 0.8$. Then, according to Algorithm 2.6, the search tree is built as follows. Since the value of the probability $q = 0.8$ is constant, we do not indicate it in the arguments list.

Let us consider the first steps of the algorithm. According to the initial steps (Lines 3–4), the function $next_size(m, n, q)$ is applied with the arguments $m = 0$ and $n = 6$, and results in the size $y = 3$. Hence, according to Lines 5.4.1b–c, the next search area has a size

$|A^{t+1}| = y = 3$. If an observation of this area results in 0, that is, $\mathbf{1}(A^{t+1}, x) = 0$, then the search area A^{t+1} does not include any target and can be removed from further consideration. Hence, at the next step the size of the observed area is obtained by application of the function $next_size(m, n, q)$ with the following arguments: $m = 0$ and $n = n - y = 6 - 3 = 3$. If, in contrast, it results in 1, that is, $\mathbf{1}(A^{t+1}, x) = 1$, then the search area A^{t+1} includes at least one target and the points of the area have to be included in further detections. According to Line 5.4.2d, the arguments of the function $next_size(m, n, q)$ are as follows: $m = y = 3$ and $n = 6$.

Let us concentrate on the last case, in which $\mathbf{1}(A^{t+1}, x) = 1$ and the arguments $m = 3$ and $n = 6$. The function $next_size(3, 6, q)$ results in the size $y = 1$, thus the area A^{t+2} is such that $|A^{t+2}| = y = 1$ has to be observed. This area includes a single point, so the observation result exactly indicates whether there is a target. Assume that there is, so $\mathbf{1}(A^{t+2}, x) = 1$. Hence, according to Lines 5.4.2, the area A^{t+2} is included in the resulting set Res and excluded from the next consideration. The arguments for the function $next_size(m, n, q)$ are $m = 0$ and $n = n - 1 = 6 - 1 = 5$. On the contrary, if $\mathbf{1}(A^{t+1}, x) = 0$, the arguments are specified as $m = 3 - y = 3 - 1 = 2$ and $n = 6 - y = 6 - 1 = 5$. The starting levels of the tree are illustrated in Figure 2.21, where the function $next_size(m, n, q)$ for given values of m and n and a returned value of y are denoted by $NS(m, n) \rightarrow y$.

The tree demonstrates how to assign a size to the observed area that has to be chosen at the next step given an observation result. Let us consider the first executions of the algorithm over the locations set $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, $n = |X| = 6$, while, as in previous examples, the targets are located at the points x_2 and x_4 . The search starts with $X^0 = X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and $next_size(0, 6, q)$ results in the size $y = 3$. Assume that the chosen area of size $y = 3$ is $A^1 = \{x_1, x_2, x_3\}$. Then, since one of the targets is located at point x_2 , an observation result is $\mathbf{1}(A^1, x) = 1$. Hence, the size of the next observed area is calculated as $next_size(3, 6, q)$ and its value is $y = 1$. Since $\mathbf{1}(A^1, x) = 1$, the observed area A^2 such that $|A^2| = 1$ is chosen from the set $X^1 = X^0$.

Assume that $A^2 = \{x_1\}$. Then $\mathbf{1}(A^2, x) = 0$, hence, the next set is $X^2 = X^1 \setminus A^2 = \{x_2, x_3, x_4, x_5, x_6\}$, $n = |X^2| = 5$, and the size of the next observed area A^3 is specified by $next_size(2, 5, q)$, that is, $y = 1$. Assume that the chosen observed area of this size is

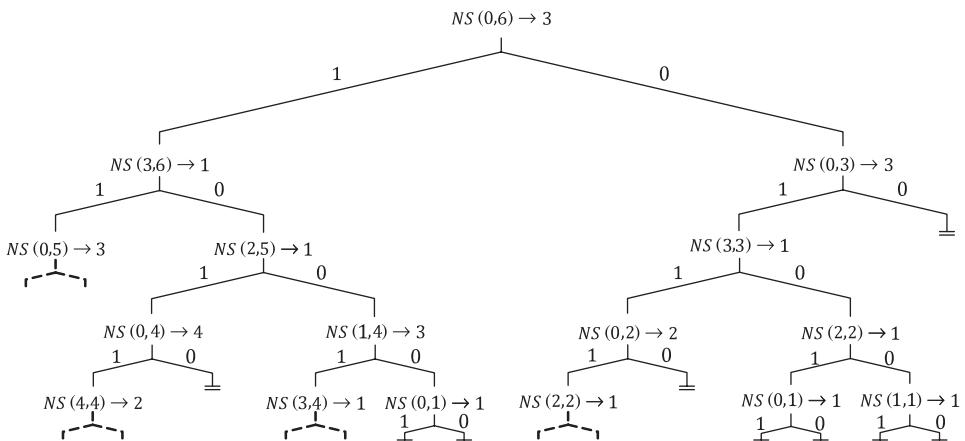


Figure 2.21 Example of the search tree for Sobel and Groll's algorithm.

$A^3 = \{x_2\}$. Then, an observation result is $\mathbf{1}(A^3, x) = 1$ and point x_2 is added to the resulting set of points, in which the targets are located, and is removed from the considered locations set. Hence, $X^3 = X^2 \setminus A^3 = \{x_3, x_4, x_5, x_6\}$. Since $\mathbf{1}(A^3, x) = 1$ and $|X^3| = 4$, the size of the next observed area is calculated as $\text{next_size}(0, 4, q)$, that is, the search continues similarly to its starting points. ■

In Sobel and Groll's Algorithm 2.6 (R_1 procedure) it is assumed that the filled area, that is, the area in which the targets are located, is binomial. Similar results regarding the observed areas such that the number of targets v in the filled area A satisfies the uniform distribution

$$\Pr \{v = y \mid \text{observed area is } A\} = 1/(m - y + 1), \quad y = 1, 2, \dots, m,$$

and corresponding group-testing procedures (the two-set R_2 procedure and conditional R_C procedure) were suggested by Rodriguez *et al.* [48]. For further modifications of Sobel and Groll's algorithm and its combinatorial versions see, for example, [27, 49–51].

Sobel and Groll's algorithm with perfect observations forms a basis for Graff and Roeloffs' algorithm [46], in which erroneous observations are allowed. Let us consider this algorithm in particular.

Let the empty and filled areas be defined according to the probabilities in Equation 2.29 and assume that the filled areas are binomial as specified by Equation 2.30 with a priori probability q of not detecting any target in the locations set and $p = 1 - q$. Thus, Lemma 2.2 is also applicable, and recursive calculations of size y of the area that has to be observed at the next search step are allowed.

Assume that, in contrast to perfect observations, in some cases an observation gives an erroneous result such that $z(A, x) = 0$, where $x \in A$; that is, some observations do not detect targets that are located in the observed area. Let the probability of such an erroneous observation be

$$\Pr \{\text{observation result of area } A \text{ is } 0 \mid \text{observed area } A \text{ is filled}\} = a,$$

for $0 < a \leq 1$. According to the assumption in Equation 2.2 for the non-existence of false targets, it is specified that

$$\Pr \{\text{observation result of area } A \text{ is } 1 \mid \text{observed area } A \text{ is empty}\} = 0.$$

In addition, assume that the cost of observation is defined and that a constant value

$$c = \text{cost of observation error}/\text{cost of observation}$$

specifies a relative cost.

In contrast to the previous Algorithm 2.6, in Graff and Roeloffs' algorithm the recursions are defined for the expected cost of observations, and, in addition to the size y , they specify a number r of required repetitive observations of the observed area. The role of the number r is as follows. If each of r repetitive observations of the area A results in $z(A, x) = 0$, then the area A is empty, but if at least one of r repetitive observations of the area A results in $z(A, x) = 1$, then the area A is filled, that is, it includes at least one target.

Assume that at time t , $t = 1, 2, \dots$, area A has been chosen and observed as filled, that is, at least one of r repetitive observations of the area A resulted in $z(A, x) = 1$. Let us define the following values:

$$\mathcal{U} = \prod_{\tau=1}^t (q^{y_\tau} + (1 - q^{y_\tau}) a^{r_\tau}), \quad (2.32a)$$

$$\mathcal{V} = \sum_{\tau=1}^t y_\tau, \quad (2.32b)$$

where y_τ is the size of the observed area and r_τ the number of repetitive observations that were specified at time τ , $\tau = 1, 2, \dots, t$.

Denote by $E_{GR}(C|m, n, \mathcal{U}, \mathcal{V}, \ell)$ the expected cost of observations that are required to find all locations that are occupied by the targets in the locations set X , $|X| = n$, while the size of the already observed filled area is $A \subset X$, $|A| = m$, and ℓ is a parameter that obtains values of 1, $m - 1$, or m corresponding to the state of the search process. Possible states of the search by using Graff and Roeloffs' algorithm [46] are defined according to the sizes of two types of the observed areas:

- *binomial areas*, for which the probability of finding the target is specified by a binomial distribution according to Equation 2.30;
- *uncertain areas*, for which it is impossible to define whether they are filled or binomial.

Then, the three following states of search process are distinguished:

- **State 1.** An area that is known as binomial is non-empty, while the areas that are known as filled and uncertain are empty.
- **State 2.** An area that is known as filled is non-empty, while the area that is known as uncertain is empty.
- **State 3.** An area that is known as filled is empty, while the area that is known as uncertain is non-empty.

For these states, the values of the expected cost $E_{GR}(C|m, n, \mathcal{U}, \mathcal{V}, \ell)$ are calculated by using the following arguments:

- **State 1.** $E_{GR}(C|m, n, \mathcal{U} = 1, \mathcal{V} = 0, \ell = 1)$
- **State 2.** $E_{GR}(C|m, n, \mathcal{U} = 1, \mathcal{V} = 0, \ell = m - 1)$
- **State 3.** $E_{GR}(C|m, n, \mathcal{U}, \mathcal{V}, \ell = m)$.

In addition, for any state it is specified that if $m = 0$ then the expected cost does not depend on ℓ and the statistics \mathcal{U} and \mathcal{V} . In this special case its value is denoted by $E_{GR}^0(C|n) = E_{GR}(C|0, n, \mathcal{U}, \mathcal{V}, \ell)$.

Recall that q denotes the a priori probability of not detecting any target in the locations set, a stands for the probability of erroneous observation, and c is the relative cost of observation. Note that these three values are assumed to be known constants. Size y of the observed area and a number r of required repetitive observations, in contrast, are considered as decision variables that obtain integer values. The values $E_{GR}^0(C|n)$

and $E_{GR}(C|m, n, u, v, \ell)$ of the expected cost are specified by the following recursive equations:

$$\begin{aligned} E_{GR}^0(C|n) = & \\ & \min_{r \geq 1, \quad 1 \leq y \leq n} \left\{ q^y \left(r + E_{GR}^0(C|n-y) \right) + \right. \\ & \left. + (1-q^y) \left(\frac{1-a^r}{1-a} + (1-a^r) E_{GR}(C|y, n, 1, 0, \ell) \right) + a^r \left(E_{GR}^0(C|n-y) + \frac{cy(1-q)}{1-q^y} \right) \right\}. \\ E_{GR}^0(C|0) = 0, \end{aligned} \quad (2.33a)$$

$$\begin{aligned} E_{GR}(C|m, n, u, v, \ell) = & \\ & \min_{r \geq 1, \quad 1 \leq y \leq \ell} \left\{ \frac{q^y u - q^{m+v}}{u - q^{m+v}} (r + E_{GR}(C|m-y, n-y, u(q^y + (1-q^y)a^r), v+y, \ell)) \right. \\ & + \frac{(1-q^y) u}{u - q^{m+v}} \left(\frac{1-a^y}{1-a} + (1-a^r) E_{GR}(C|y, n, 1, 0, \ell) + a^r \right) \\ & \left. \left(E_{GR}^0(C|n-y) + \frac{cy(1-q)}{1-q^y} \right) \right\}. \\ E_{GR}(C|1, n, 1, 0, \ell) = E_{GR}^0(C|n-1). \end{aligned} \quad (2.33b)$$

Integer numbers y and r that solve the recursions (Equations 2.33) represent the size of the area that has to be observed at the next step and a number of repetitive observations of this area, correspondingly. Note again that $\ell \in \{1, m-1, m\}$ in the recursions (Equations 2.33) is specified according to the state of the search.

Similar to the above, let us define the function $next_size_and_reobs(m, n, u, v, \ell, a, c, q)$ that follows the recursions (Equations 2.33) according to its arguments and returns size y and the number of ‘re-observations’ r :

$next_size_and_reobs(m, n, u, v, \ell, a, c, q)$:

1. If $m \leq 0$ then:

1.1. Calculate $E_{GR}^0(C|n)$ according to Equation 2.33a.
Else

1.2. Calculate $E_{GR}(C|m, n, u, v, \ell)$ according to Equation 2.33b.

2. Return size y and re-observations number r .

Since in Graff and Roeloffs’ algorithm [46] an observation result is obtained according to r re-observations, let us define the function $observation_result(A, r)$. According to the role of the number r indicated above, if each of r repetitive observations of the area A results in $z(A, x) = 0$, then the function returns ‘0,’ and if at least one of r repetitive observations of the area A results in $z(A, x) = 1$, it returns ‘1.’ In addition, let us define two arrays $ArrY$ and $ArrR$ that store, if needed, the values of y and r , and that are used for calculating statistics u and v according to Equation 2.32. The functions $calculate_u(ArrY, ArrR)$ and $calculate_v(ArrY)$ obtain these statistics and return their values. Then, according to the description by Graff and Roeloffs [46], the group-testing search algorithm with re-observations is outlined as follows.

Algorithm 2.7 (Graff and Roeloffs' group-testing search with uncertain observations) [46]. Given locations set X , probability q , probability of erroneous observation a , and relative cost c do:

1. Set $t = 0$.
2. Initialize resulting set of points: Set $Res = \emptyset$.
3. Initialize arrays $ArrY$ and $ArrR$ as empty.
4. Set $X^t = X$ and $A^t = \emptyset$.
5. Do:
 - State 1
 - 5.1. Set $n = |X^t|$ and $m = |A^t|$.
 - 5.2. Set $\mu = 1$ and $\nu = 0$.
 - 5.3. Set $\ell = 1$.
 - 5.4. Set $(y, r) = next_size_and_reobs(m, n, \mu, \nu, \ell, a, c, q)$.
 - 5.5. Choose an observed area $A^{t+1} \subset X^t$ such that $|A^{t+1}| = y$.
 - 5.6. Set $z^{t+1} = observation_result(A^{t+1}, r)$.
 - 5.7. If $z^{t+1} = 0$ then:
 - 5.7.1. Add y to $ArrY$ and r to $ArrR$.
 - 5.7.2. If $y = n$ then:
 - a. Break.
 - Else:
 - b. Set $X^{t+1} = X^t \setminus A^{t+1}$.
 - c. Continue.
 - Else:
 - 5.7.3. Set $ArrY$ and $ArrR$ as empty.
 - 5.7.4. Add y to $ArrY$ and r to $ArrR$.
 - 5.7.5. If $y = 1$ then:
 - a. Set $Res = Res \cup A^{t+1}$.
 - b. Set $X^{t+1} = X^t \setminus A^{t+1}$ and $A^{t+1} = \emptyset$.
 - c. Continue.
 - Else:
 - d. Go to State 2.
- State 2
- 5.8. Set $B^t = A^{t+1}$.

- 5.9. Set $n = |X^t|$ and $m = |B^t|$.
 - 5.10. Set $u = 1$ and $v = 0$.
 - 5.11. Set $\ell = m - 1$.
 - 5.12. Set $(y, r) = \text{next_size_and_reobs}(m, n, u, v, \ell, a, c, q)$.
 - 5.13. Choose an observed area $B^{t+1} \subset B^t$ such that $|B^{t+1}| = y$.
 - 5.14. Set $z^{t+1} = \text{observation_result}(B^{t+1}, r)$.
 - 5.15. If $z^{t+1} = 0$ then:
 - 5.15.1. Add y to ArrY and r to ArrR .
 - 5.15.2. If $y = m$ then:
 - a. Set $X^{t+1} = X^t \setminus B^{t+1}$ and $A^{t+1} = \emptyset$.
 - b. Continue.
 - Else:
 - c. Set $X^{t+1} = X^t \setminus B^{t+1}$.
 - d. Go to State 3.
 - Else:
 - 5.15.3. Set ArrY and ArrR as empty.
 - 5.15.4. Add y to ArrY and r to ArrR .
 - 5.15.5. If $y = 1$ then:
 - a. Set $\text{Res} = \text{Res} \cup B^{t+1}$.
 - b. Set $X^{t+1} = X^t \setminus B^{t+1}$ and $A^{t+1} = \emptyset$.
 - c. Continue.
 - Else:
 - d. Set $A^{t+1} = B^{t+1}$.
 - e. Go to State 2.
- State3
- 5.16. Set $C^t = A^{t+1} \setminus B^{t+1}$.
 - 5.17. Set $n = |X^{t+1}|$ and $m = |C^t|$.
 - 5.18. Set $u = \text{calculate_u}(\text{ArrY}, \text{ArrR})$ and $v = \text{calculate_v}(\text{ArrY})$.
 - 5.19. Set $\ell = m$.
 - 5.20. Set $(y, r) = \text{next_size_and_reobs}(m, n, u, v, \ell, a, c, q)$.
 - 5.21. Choose an observed area $C^{t+1} \subset C^t$ such that $|C^{t+1}| = y$.
 - 5.22. Set $z^{t+1} = \text{observation_result}(C^{t+1}, r)$.

5.23. If $z^{t+1} = 0$ then:

5.23.1. Add y to $ArrY$ and r to $ArrR$.

5.23.2. Set $X^{t+1} = X^t \setminus C^{t+1}$ and $B^{t+1} = C^{t+1}$.

5.23.3. Go to State 3.

Else:

5.23.4. Set $ArrY$ and $ArrR$ as empty.

5.23.5. Add y to $ArrY$ and r to $ArrR$.

5.23.6. If $y = 1$ then:

a. Set $Res = Res \cup B^{t+1}$.

b. Set $X^{t+1} = X^t \setminus B^{t+1}$ and $A^{t+1} = \emptyset$.

c. Continue.

Else:

d. Set $A^{t+1} = C^{t+1}$.

e. Go to State 2.

5.24. Set $t = t + 1$.

While $|X^{t+1}| > 0$.

6. Return Res . ■

Note that the recursions (Equations 2.33), which are used by the function *next_size_and_reobs(...)*, converge very slowly for low probabilities a of erroneous observations and the algorithm can be implemented for small locations sets. The actions of the algorithm are illustrated by the following example.

Example 2.14 In order to obtain the complete tree, let us consider actions of Algorithm 2.7 over a small locations set such that, as in Examples 2.8 and 2.9, locations set X includes $n = |X| = 4$ points. Assume the probability of not detecting any of the targets is $q = 0.8$, the probability of erroneous observation is $a = 0.3$, and the relative cost of observation is $c = 10$. As above, since a , c , and q are constants, they are not included in the list of arguments.

The search tree created by Algorithm 2.7 is rather similar to the search tree created by Algorithm 2.6 (see Example 2.13). The first levels of the tree are shown in Figure 2.22.

In the figure, states 1, 2, and 3 are denoted by ST1, ST2, and ST3, correspondingly. Function *next_size_and_reobs*($m, n, u, v, \ell, a, c, q$) for the values of m and n and a returning pair of the values of y and r are denoted by $NSR(m, n) \rightarrow (y, r)$. Values of u and v in states 1 and 2 are $u = 1$ and $v = 0$, while for state 3 these values are indicated in the figure for each call of the function.

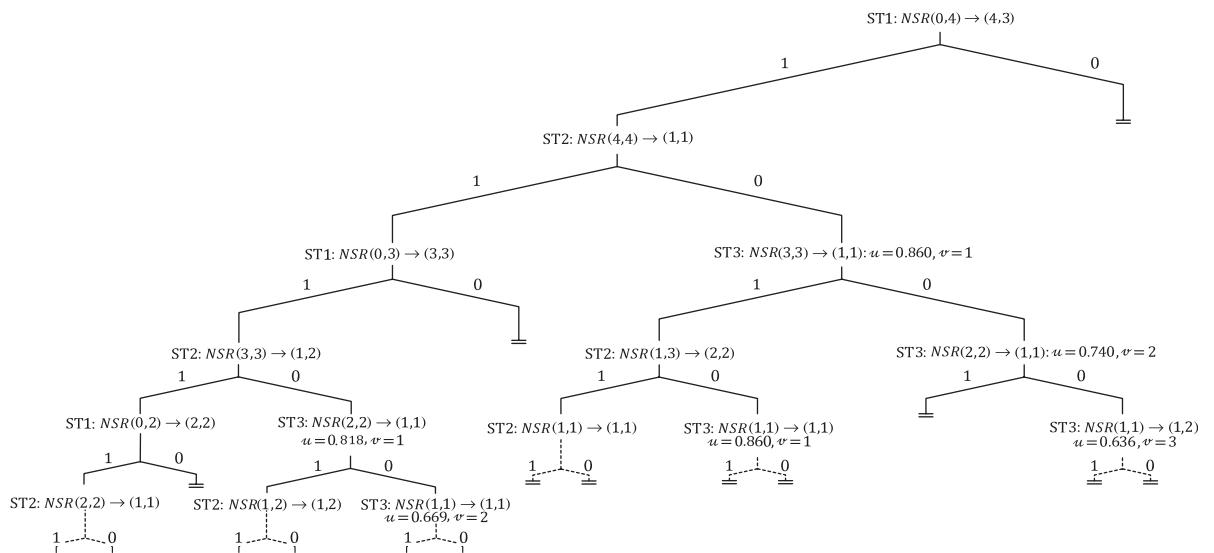


Figure 2.22 Example of the search tree for Graff and Roeloffs' algorithm.

As in Example 2.13, the tree demonstrates how to assign a size for the observed area that has to be chosen at the next step given an observation result. In addition, the tree shows the states of the search process. Let us illustrate a search that follows the constructed tree.

Assume that in the locations set $X = \{x_1, x_2, x_3, x_4\}$, $n = |X| = 4$, the targets are located at points x_2 and x_4 . As above, we assume that the observed areas are chosen according to the regular order of the points:

- **State 1.** The search starts in State 1 with the set $X^0 = X = \{x_1, x_2, x_3, x_4\}$ and the function $\text{next_size_and_reobs}(0, 4, 1, 0, \ell, a, c, q)$ results in the size $y = 4$ and re-observations number $r = 3$. Hence, the chosen observed area $A^1 = \{x_1, x_2, x_3, x_4\}$. If each of three observations result in 0, then the search terminates. Assume that this is not the case and that the observation result is 1, so the search continues.
- **State 2.** The process passes to State 2, and the size y of the next observed area and the number r of re-observations are calculated by $\text{next_size_and_reobs}(4, 4, 1, 0, \ell, a, c, q)$. The function results in the size $y = 1$ and re-observations number $r = 1$. Hence, the next observed area B^1 has size $|B^1| = 1$, and has to be chosen from the observed area $A^1 = \{x_1, x_2, x_3, x_4\}$. Assume that this area is $B^1 = \{x_1\}$. Since this point does not include the target, let us assume that the single observation of B^1 results in 0.
- **State 3.** Since the observation in State 2 resulted in 0, the search passes to State 3. Up to this stage, there was one observation that resulted in 1 (in State 1), and after this observation the size of the chosen observed area was $y = 1$ and the re-observations number was $r = 1$ (in State 2). Hence, u and v calculated for the values $y = 1$ and $r = 1$ are $u = 0.860$ and $v = 1$. The size y of the next observed area and the number r of re-observations are calculated by $\text{next_size_and_reobs}(3, 3, 0.860, 1, \ell, a, c, q)$, which gives $y = 1$ and $r = 1$. Hence, the next observed area C^1 has size $|C^1| = 1$, and has to be chosen from the set $C^0 = A^1 \setminus B^1 \setminus \{x_1\} = \{x_2, x_3, x_4\}$ and observed one time. Let the chosen area be $C^1 = \{x_2\}$. Since one of the targets is located at this point, assume that the observation result is 1. Then the point x_2 is excluded from the search process and added to the resulting set $Res = Res \cup \{x_2\}$.
- **State 2.** Since the observation in State 3 resulted in 1, the search passes to State 2, while the observed area after State 3 is $A^2 = \{x_1, x_3, x_4\}$, $n = |A^2| = 3$, and area $B^1 = \{x_1\}$, which includes only one point x_1 , $m = |B^1| = 1$, was already observed with a zero result. Since the process is in State 2, $u = 1$ and $v = 0$, and the size y and the number r are calculated by $\text{next_size_and_reobs}(1, 3, 1, 0, \ell, a, c, q)$. The obtained results are $y = 2$ and $r = 2$. Assume that the chosen area is $B^2 = \{x_1, x_3\}$. Since both points do not contain the targets, suppose that two required observations result in 0. Then the process again passes to State 3.

In State 3 the search continues similarly to the procedure described above, up to finding the target at the last point x_4 . ■

Graff and Roeloffs' algorithm allows false negative observations, while an observation of the area that includes the target gives a zero result. As indicated above, Gupta and Malina [29] considered a procedure in which false positive observations, that is, an observation of an empty area gives a non-zero result, are allowed. However, such a scenario disagrees with the assumption (Equation 2.2) of non-existence of false targets. For modifications of Graff and Roeloffs' algorithm and similar group-testing schemes see, for example, [52–54].

The combinatorial procedures above present scenarios of search for static targets, while the number of targets is unknown, erroneous observations are allowed, and the size of the observed areas is not restricted. However, the combinatorial scenarios do not consider those cases in which the location probabilities of the target are available. Below, we consider some basic information theory methods of search for static targets that utilize such probabilities. Further development of the information theory algorithms of search will be presented in later chapters.

2.2.4 Basic information theory search with known location probabilities

As indicated above, the algorithms of combinatorial group-testing search are applicable to search for multiple static targets, while the strict number of targets is not available to the searcher, and allow uncertain observations. Nevertheless, such algorithms do not take into account possible information regarding target locations that, in the case of static target search, are specified by the target location probabilities. In this section, we present two basic algorithms of group-testing search for a single static target that utilize such probabilities.

Let us return to the general description of group-testing search (see Section 2.2.1) and assume that the search over locations set $X = \{x_1, x_2, \dots, x_n\}$ is based on certain observations; that is, for any observed area $A^t \subset X$, which is chosen at time t , $t = 0, 1, 2, \dots$, the observation result is $z^t = \mathbf{1}(A^t, x^t) \in \{0, 1\}$. In addition, assume that, similar to the search and screening methods (see Section 2.1), for any time t , $t = 0, 1, 2, \dots$, location probabilities $p^t(x_i)$, $i = 1, \dots, n$, can be defined such that for any t it follows that $0 \leq p^t(x_i) \leq 1$ and $\sum_{i=1}^n p^t(x_i) = 1$. In the worst case, it can be specified that $p^{t=0}(x_i) = 1/n$ for each $i = 1, \dots, n$. Since we consider search for a static target, let us omit index t and denote $p_i = p(x_i) = p^{t=0}(x_i)$. As in previous sections, the locations set X with the defined location probabilities is called the *sample space*, while in general statements we will refer to it as some finite set.

Note that, in this section, in contrast to the above procedures, we consider search for a single static target such that the group-testing search is represented by a binary search tree $T[X]$. Recall that for such a search, the areas which correspond to the vertexes on each level of the tree $T[X]$ form a set partition system [34] for the respective area that corresponds to the node of the higher level of the tree. Let v be a vertex of the binary search tree $T[X]$ and let v_1 and v_2 be descending vertexes of v . Assume that vertex v is associated with an observed area A and vertexes v_1 and v_2 are associated with some observed areas A_1 and A_2 , correspondingly. Then, if for any vertex v and its associated area A the areas A_1 and A_2 of the descendants v_1 and v_2 of v form a set partition system of A , we say that the tree $T[X]$ is a *binary partitioning tree*. In the tree $T[X]$, we denote by $l(x_i)$ a number edges from the root node, which is associated with the set X , to the leaf which is associated with the single-point set $\{x_i\}$, $x_i \in X$, $i = 1, \dots, n$. Thus the average number of edges from the root to the leaves is defined by the equation

$$L = \sum_{i=1}^n p_i l(x_i). \quad (2.34)$$

The simplest procedure for constructing a binary partitioning tree $T[X]$ is similar to Algorithm 2.4 for binary splitting. This procedure is formulated as follows.

Procedure 2.4 (division by half). Given the sample space $X = \{x_1, x_2, \dots, x_n\}$ with the probabilities p_i , $i = 1, \dots, n$, process the following actions:

1. Start with the sample space X and consider it as an observed area A .
2. At each step, divide all observed areas obtained at that step into two disjoint areas A' and A'' such that the difference $|p(A') - p(A'')|$ between their probabilities $p(A') = \sum_{x \in A'} p(x)$ and $p(A'') = \sum_{x \in A''} p(x)$ is minimal (ties are broken randomly).
3. Terminate when the single-point areas $\{x_i\}$, $i = 1, \dots, n$, are obtained.

■

The binary search tree that is constructed by Procedure 2.4 is illustrated by the following example.

Example 2.15 Let the sample space be $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and assume that the location probabilities $p_i = p(x_i)$, $i = 1, \dots, n$, are defined as follows:

x_i	x_1	x_2	x_3	x_4	x_5	x_6
p_i	0.2	0.05	0.1	0.4	0.2	0.05

According to Procedure 2.4, the tree is built starting from the complete sample space X so that at each node, except leaves, the corresponding area is divided into two parts with equal probabilities. The resulting tree is shown in Figure 2.23.

The probabilities of the elements are given in the figure, while the index presents the number of the point according to the original order in the sample space X .

According to Equation 2.34, the average number L of edges from the root to the leaves is $L = 0.2 \times 3 + 0.05 \times 3 + 0.1 \times 3 + 0.4 \times 2 + 0.2 \times 2 + 0.05 \times 3 = 2.4$, while the depth of the tree is $D(\mathcal{T}[X]) = \max_{1 \leq i \leq n} \{l(x_i)\} = 3$. ■

Note that Procedure 2.4 does not present a search process, while an implementation of the rule given in Line 2 of the binary splitting Algorithm 2.4 results in a similar algorithm of ‘division by half’ search.

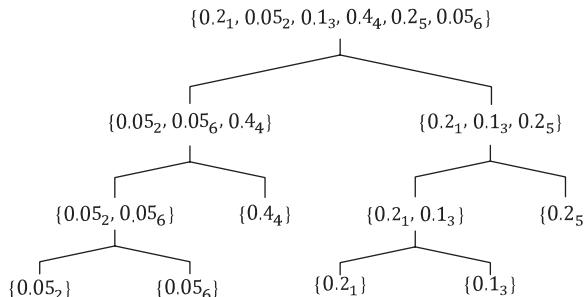


Figure 2.23 Example of ‘division by half’ search tree.

Algorithm 2.8 ('division by half' search). Given sample space X with location probabilities p_i , $i = 1, \dots, n$, do:

1. If $\mathbf{1}(X, x) = 0$ then: Return \emptyset .
2. Set $t = 0$.
3. Set $X^t = X$.
4. While $|X^t| > 1$ do:
 - 4.1 Split the space X^t into two observed areas A_1^t and A_2^t such that:
 - a. $A_1^t \cap A_2^t = \emptyset$ and $A_1^t \cup A_2^t = X^t$,
 - b. $|p(A_1^t) - p(A_2^t)| \rightarrow \min$, $p(A_1^t) = \sum_{x \in A_1^t} p(x)$, and $p(A_2^t) = \sum_{x \in A_2^t} p(x)$.
 - 4.2 Set $z_1^t = \mathbf{1}(A_1^t, x)$ and $z_2^t = \mathbf{1}(A_2^t, x)$.
 - 4.3 If $z_1^t = 1$ then:
 - 4.3.1 Set $X^{t+1} = A_1^t$.
 - Else:
 - 4.3.2 Set $X^{t+1} = A_2^t$.
 - 4.4 Set $t = t + 1$.
5. Return X^t . ■

The actions of Algorithm 2.8 are similar to those of the binary splitting Algorithm 2.4, while the underlying Procedure 2.4 is the best known procedure for constructing binary search trees for a search over a sample space with defined location probabilities. However, the resulting trees do not guarantee a search procedure with a minimal number of expected observations for finding the target. To obtain methods for creating partitioning trees that provide such a minimum, additional considerations are required. Since the lemmas that are presented below are general and not necessarily focused on the search methods, we will refer to X as some finite set.

Lemma 2.3 (Kraft inequality) [34, 35]. *Let X be a finite set and $T[X]$ be a binary partitioning tree over X . Then, the numbers $l(x_i)$ of edges from the root node (associated with X) to the leaves (associated with $\{x_i\}$) satisfy the following inequality:*

$$\sum_{i=1}^n 2^{-l(x_i)} \leq 1. \quad (2.35)$$

Proof. (sketch). Let $l_{\max} = D(T[X]) = \max_{1 \leq i \leq n} \{l(x_i)\}$ be the depth of the tree $T[X]$, that is, a maximal number of edges from the root to a leaf, and let T_{\max} be a maximal binary tree of depth l_{\max} , that is, a tree such that for each vertex except leaves there are exactly two descendant vertexes. An example of a binary tree of $l_{\max} = 3$ and the corresponding maximal binary tree are shown in Figure 2.24.

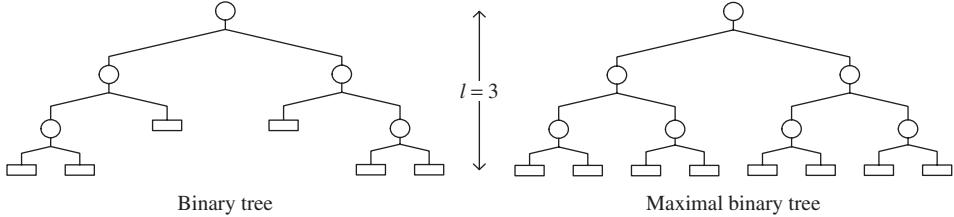


Figure 2.24 Example of binary tree and maximal binary tree.

In the maximal tree T_{\max} there are exactly $2^{l_{\max}}$ leaves, and for any leaf $\{x_i\}$ of the tree $T[X]$ such that the number of edges to this leaf is $l(x_i)$, in the tree T_{\max} there are $2^{l_{\max}-l(x_i)}$ leaves. Since tree $T[X]$ is the partitioning tree, the areas that are assigned to the nodes of each level, including leaves, are disjoint and a sequence of the areas that correspond to each branch starting from the root up to a leaf is unique. Hence, for the tree $T[X]$ it follows that $\sum_{i=1}^n 2^{l_{\max}-l(x_i)} \leq 2^{l_{\max}}$, which gives the required inequality (Equation 2.35).

To examine the reverse statement that given a set of numbers of edges $l(x_i)$, $i = 1, \dots, n$, that satisfy the Kraft inequality (Equation 2.35) one can build a binary partitioning tree, let us consider an example of such a tree. Let $X = \{x_1, x_2, \dots, x_6\}$ and assume that $l(x_1) = l(x_2) = l(x_5) = l(x_6) = 3$, while $l(x_3) = l(x_4) = 2$. A straightforward calculation shows that

$$\sum_{i=1}^n 2^{-l(x_i)} = 4 \cdot \frac{1}{2^3} + 2 \cdot 4 \cdot \frac{1}{2^2} = \frac{1}{2} + \frac{1}{2} = 1,$$

that is, for the indicated numbers of edges $l(x_i)$, the Kraft inequality (Equation 2.35) holds. The binary tree that includes branches with these numbers of edges is the tree shown on the left side of Figure 2.24. ■

As indicated above, the proven inequality (Equation 2.35) generalizes Equation 2.25 and holds for any reasonable group-testing algorithm. In addition, note that if instead of a binary tree the Q -ary partitioning tree is applied, then the inequality in Equation 2.35 obtains the form $\sum_{i=1}^n Q^{-l(x_i)} \leq 1$.

This inequality is known as the *Kraft inequality* since it was formulated and proved in Kraft's MSc thesis [55] in 1949. However, as indicated by Mandelbrot [56], the first formulation of the inequality appears in Szilard's paper [57], which was published in 1929.

Using the Kraft inequality (Equation 2.35), let us consider the problem of partitioning such that the average number $L = \sum_{i=1}^n p_i l(x_i)$ of edge numbers $l(x_i)$ reaches its minimum, $0 \leq p_i \leq 1$, $\sum_{i=1}^n p_i = 1$. The minimization problem is formulated as follows [35]:

$$L = \sum_{i=1}^n p_i l(x_i) \rightarrow \min, \quad (2.36)$$

$$\text{subject to the Kraft inequality : } \sum_{i=1}^n 2^{-l(x_i)} \leq 1 \text{ and } \sum_{i=1}^n p_i = 1. \quad (2.37)$$

To consider this minimization problem, let us neglect the integer constraint on the edge numbers $l(x_i)$, $i = 1, \dots, n$, and treat them as some positive real numbers. In addition, let

us assume that in Equation 2.37 an equality holds. Then, we have to minimize a Lagrangian

$$\mathcal{L} = \sum_{i=1}^n p_i l(x_i) + \lambda \sum_{i=1}^n 2^{-l(x_i)},$$

where $0 \leq \lambda < \infty$ is the Lagrange multiplier.

From the requirement $\partial \mathcal{L} / \partial l(x_i) = p_i - \lambda 2^{-l(x_i)} \ln 2 = 0$ it follows that $2^{-l(x_i)} = p_i / (\lambda \ln 2)$. Substitution of the obtained equivalence into the constraint (Equation 2.37), in which the equality holds, results in

$$\frac{1}{\lambda \ln 2} \sum_{i=1}^n p_i = 1$$

while $\sum_{i=1}^n p_i = 1$. Thus, $\lambda = 1 / \ln 2$, and from the equivalence $2^{-l(x_i)} = p_i / (\lambda \ln 2)$ it follows that $p_i = 2^{-l(x_i)}$. From the last expression we obtain that the non-integer numbers $l^*(x_i)$, $i = 1, \dots, n$, that solve the minimization problem (2.36–2.37) are

$$l^*(x_i) = -\log p_i.$$

If $\log p_i$ are integer numbers, then the values $l^*(x_i)$ represent the number of edges that provide a minimum of the average number

$$L^* = \sum_{i=1}^n p_i l^*(x_i) = -\sum_{i=1}^n p_i \log p_i,$$

while if the numbers of $\log p_i$ are non-integer, then $l(x_i) \neq -\log p_i$ and numbers $l(x_i)$ have to be close to the optimal values $-\log p_i$. Thus, in other words, the entropy over the search space probability provides a lower bound on the number of expected search steps.

Let us show that the value $L^* = -\sum_{i=1}^n p_i \log p_i$ provides a general minimal average over all possible edge numbers $l(x_i)$, $i = 1, \dots, n$.

Lemma 2.4 [34, 35]. *Given a set $X = \{x_1, x_2, \dots, x_n\}$, for any probabilities p_i , $i = 1, \dots, n$, over X and binary partitioning tree $T[X]$ it follows that*

$$L = \sum_{i=1}^n p_i l(x_i) \geq L^* = -\sum_{i=1}^n p_i \log p_i, \quad (2.38)$$

while the equality holds if and only if $p_i = 2^{-l(x_i)}$.

Proof. Let us consider the difference between L and L^* . We have

$$L - L^* = \sum_{i=1}^n p_i l(x_i) + \sum_{i=1}^n p_i \log p_i = -\sum_{i=1}^n p_i \log 2^{-l(x_i)} + \sum_{i=1}^n p_i \log p_i.$$

Let us specify $r_i = 2^{-l(x_i)} / \sum_{i=1}^n 2^{-l(x_i)}$. Then

$$\begin{aligned} L - L^* &= \sum_{i=1}^n p_i \log \frac{p_i}{r_i} - \sum_{i=1}^n p_i \log \sum_{i=1}^n 2^{-l(x_i)} \\ &= \sum_{i=1}^n p_i \log \frac{p_i}{r_i} + \sum_{i=1}^n p_i \log \frac{1}{\sum_{i=1}^n 2^{-l(x_i)}}. \end{aligned}$$

According to the Kraft inequality (2.35), $\sum_{i=1}^n 2^{-l(x_i)} \leq 1$. Hence, for the second addendum the inequality

$$\sum_{i=1}^n p_i \log \frac{1}{\sum_{i=1}^n 2^{-l(x_i)}} \geq 0$$

holds.

Let us consider the first addendum $\sum_{i=1}^n p_i \log(p_i/r_i)$. The values of r_i are positive and $\sum_{i=1}^n r_i = 1$. Thus, they can be considered as some probabilities that, in general, differ from p_i , $i = 1, \dots, n$. To demonstrate that $\sum_{i=1}^n p_i \log(p_i/r_i) \geq 0$, let us consider the value

$$-\sum_{i=1}^n p_i \log \frac{p_i}{r_i} = \sum_{i=1}^n p_i \log \frac{r_i}{p_i} = \frac{1}{\ln 2} \sum_{i=1}^n p_i \ln \frac{r_i}{p_i} \leq 0.$$

Since $1/\ln 2 > 0$, we need to show that $\sum_{i=1}^n p_i \log(p_i/r_i) \geq 0$. For any real number $y \geq 0$ function $\ln(1+y)$ is convex and $\ln(1+y) \leq y$. Hence, it follows that

$$\begin{aligned} \sum_{i=1}^n p_i \ln \frac{r_i}{p_i} &= \sum_{i=1}^n p_i \ln \left(\frac{p_i + r_i - p_i}{p_i} \right) = \sum_{i=1}^n p_i \ln \left(1 + \frac{r_i - p_i}{p_i} \right) \\ &\leq \sum_{i=1}^n p_i \frac{r_i - p_i}{p_i} = \sum_{i=1}^n (r_i - p_i) = \sum_{i=1}^n r_i - \sum_{i=1}^n p_i = 1 - 1 = 0. \end{aligned}$$

Thus, $-\sum_{i=1}^n p_i \log(p_i/r_i) \leq 0$ and for the first addendum an inequality $\sum_{i=1}^n p_i \log(p_i/r_i) \geq 0$ also holds. Thus, since both addendums are non-negative, $L - L^* \geq 0$ holds.

Let us consider this inequality with the probabilities $p_i = 2^{-l(x_i)}$. In this case, $\sum_{i=1}^n 2^{-l(x_i)} = 1$, so in the first addendum $r_i = 2^{-l(x_i)} / \sum_{j=1}^n 2^{-l(x_j)} = 2^{-l(x_i)} = p_i$ and $\sum_{i=1}^n p_i \log(p_i/r_i) = \sum_{i=1}^n p_i \log 1 = 0$, and in the second addendum

$$\sum_{i=1}^n p_i \log \frac{1}{\sum_{i=1}^n 2^{-l(x_i)}} = \sum_{i=1}^n p_i \log 1 = 0,$$

as well. Thus, for the probabilities $p_i = 2^{-l(x_i)}$, an equivalence $L = L^*$ holds. ■

Note that if the proven inequality (2.38) is implemented over the equal probabilities $p_i = 1/n$, $i = 1, \dots, n$, then it obtains the form

$$L \geq -\sum_{i=1}^n p_i \log p_i = -\sum_{i=1}^n \frac{1}{n} \log \frac{1}{n} = \log n,$$

which, under the requirement of integer L , provides the inequality (2.26) which bounds the depth of the binary search tree.

Finally, let us obtain the possible upper bound of the average number $L = \sum_{i=1}^n p_i l(x_i)$.

Lemma 2.5 [34, 35]. *There exists a binary partitioning tree $T[X]$ such that for the average number of edges it follows that*

$$L = \sum_{i=1}^n p_i l(x_i) < L^* + 1 = -\sum_{i=1}^n p_i \log p_i + 1.$$

Proof. If the probabilities p_i , $i = 1, \dots, n$, are such that the numbers $-\log p_i$ are integer, then $l(x_i) = -\log p_i$ and $L = \sum_{i=1}^n p_i l(x_i) = L^* < L^* + 1$.

Assume that $-\log p_i$ are non-integer so that $l(x_i) \neq -\log p_i$. Let us choose $l(x_i) = \lceil -\log p_i \rceil$, where $\lceil \cdot \rceil$ stands for a ceiling of the real number. Then

$$l(x_i) < -\log p_i + 1, \quad i = 1, \dots, n.$$

Multiplication of this inequality by the corresponding probability p_i gives

$$p_i l(x_i) < -p_i \log p_i + p_i, \quad i = 1, \dots, n.$$

Finally, summing both parts of the last inequality over i , we obtain

$$\sum_{i=1}^n p_i l(x_i) < -\sum_{i=1}^n p_i \log p_i + 1,$$

as required. ■

Similar to Lemma 2.2, Lemmas 2.3 and 2.4 still correct for Q -ary partitioning tree with corresponding changes in the base of the logarithm and specifying $p_i = Q^{-l(x_i)}$.

Note that these trees are partitioning. Hence, the edge numbers $l(x_i)$ are unambiguously associated with the leaves $\{x_i\}$ of the trees, and the probabilities p_i can be associated with points x_i , so we can write $p_i = p(x_i)$. Then, from the lemmas above it follows that, given a set $X = \{x_1, x_2, \dots, x_n\}$ and probabilities $p_i = p(x_i)$, $i = 1, \dots, n$, there exists a binary partitioning tree $T[X]$ such that for the average number $L = \sum_{i=1}^n p_i l(x_i)$ it follows that

$$-\sum_{i=1}^n p_i \log p_i \leq \sum_{i=1}^n p_i l(x_i) < -\sum_{i=1}^n p_i \log p_i + 1, \quad (2.39)$$

where the lower bound is reached while $p_i = 2^{-l(x_i)}$, or, essentially the same, while $-\log p_i$ are integer numbers.

Let us consider the algorithm that results in the optimal binary partitioning tree. In the field of information theory, such an algorithm was suggested by Huffman [58] in 1952 as an algorithm for constructing minimal codes. The same algorithm in the field of group-testing was independently suggested by Zimmerman [59] in 1959, who proved its optimality using general considerations. Below, we present the Zimmerman procedure and then demonstrate the correspondence between the obtained binary partitioning tree and the Huffman binary code of minimal average length.

In contrast to the above procedures that start from the complete sample space X and follow down to the points $x \in X$ by specifying the sizes of areas, which have to be observed at the next steps of search, the considered algorithm starts with individual points $x \in X$ that are combined up to the complete space X . Hence, in contrast to Procedure 2.4, which constructs the search tree down from the root top level to the leaves level, the Zimmerman algorithm constructs the search tree by pairwise combination of the vertexes from the leaves'

bottom level up to the root level. The algorithm acts iteratively and the combination rule is specified by the following theorem.

Theorem 2.7 [59]. (optimal binary combination). *Given the sample space $X = \{x_1, x_2, \dots, x_n\}$ with probabilities p_i , $i = 1, \dots, n$, an iterative procedure for creating an optimal binary partitioning tree $\mathcal{T}[X]$ is specified by the following actions:*

1. Start from the leaves associated with the single-point areas $\{x_i\}$, $i = 1, \dots, n$.
2. At each step, combine two nodes that are associated with the areas A' and A'' that have the two smallest probabilities $p(A') = \sum_{x \in A'} p(x)$ and $p(A'') = \sum_{x \in A''} p(x)$ (ties are broken randomly).
3. Terminate when the combination of the complete space X is obtained.

Proof. We need to prove that the application of the specified combination rule results in the optimal binary partitioning tree. Let $X = \{x_1, x_2, \dots, x_n\}$ be a sample space with probabilities $p_i = p(x_i)$, $i = 1, \dots, n$. Let us arrange the space X in ascending order of the probabilities so that

$$p(x_1) \leq p(x_2) \leq \dots \leq p(x_n).$$

Since in the binary partitioning tree $\mathcal{T}[X]$ the number of edges from the root X to the leaves $\{x_i\}$ is $l(x_i) = \lceil -\log p_i \rceil$, $i = 1, \dots, n$, in descending order of the probabilities it follows that the edge numbers $l(x_i)$ are descending,

$$l(x_1) \geq l(x_2) \geq \dots \geq l(x_n),$$

that is, the number of edges from the root to the leaf with smaller probability is greater than the number of edges to the leaf with larger probability. The sample space that is arranged according to this rule is called the *n-set*.

First, let us demonstrate that the combination rule specified by the theorem can be applied recursively. Assume that the rule is applied to the *n-set*. Then, the obtained set has the same overall probability and can be ordered so that the $(n - 1)$ -set is obtained. Application of the rule $(n - 1)$ times results in the 1-set that includes a single element – the original sample space X .

Now let us demonstrate that recursive application of the specified combination rule results in the optimal binary tree. Denote by $E(D|\mathcal{T}[n])$ an expected depth of the binary tree that is built over the *n-set*, $E(D|\mathcal{T}[1]) = 0$. If the *n-set* corresponds to the original sample space $X = \{x_1, x_2, \dots, x_n\}$ with n points, which represent the leaves of the tree, then $E(D|\mathcal{T}[n]) = L = \sum_{i=1}^n p_i l(x_i)$, while for the sets of the observed areas that correspond to the intermediate nodes this equality does not hold. Since each combination reduces exactly one edge for each combined element, we have $E(D|\mathcal{T}[n]) = E(D|\mathcal{T}[n - 1]) + (p_i + p_j)$, where p_i and p_j are the probabilities of the combined elements. It is clear that the greater values of these probabilities result in a greater value of $E(D|\mathcal{T}[n])$, while a proportional change in p_i and p_j without changing their sum does not change the value of $E(D|\mathcal{T}[n])$.

Since $E(D|\mathcal{T}[1]) = 0$, the specified combination rule provides minimal expected depth $E(D|\mathcal{T}[n])$ for any 1-set. In addition, since there is a single combination for the 2-set, the statement is also true for any 2-set.

Assume that the combination rule, which is specified by the theorem, provides minimal expected depth $E(D|\mathcal{T}[m])$ for the m -set, where $m = 1, \dots, n - 1$, and let us show that the rule provides minimal expected depth for the n -set. In any n -set, where $p_1 \leq p_2 \leq \dots \leq p_n$, there are three possible first combinations of different elements i and j , $i \neq j$:

1. Combination of the elements i and j such that both $i > 2$ and $j > 2$.
2. Combination of the elements i and j such that either $i < 3$ or $j < 3$.
3. Combination of the elements i and j such that both $i < 3$ and $j < 3$.

The last possibility represents the combination rule, which is specified by the theorem, and we need to demonstrate that this combination is at least as good as the first ones. Let us compare possibilities 1 and 2 to possibility 3:

1. Combination of the elements i and j , while both $i > 2$ and $j > 2$, results in the $(n - 1)$ -set with probabilities $p_1, p_2, \dots, (p_i + p_j), \dots, p_n$. According to the assumption, in the next step elements 1 and 2 are combined, and the resulting $(n - 2)$ -set has probabilities $p_3, (p_1 + p_2), \dots, (p_i + p_j), \dots, p_n$.

If, instead, at first elements 1 and 2 are combined, then at the next step elements i and j can be combined to result in the same $(n - 2)$ -set. Hence,

$$E(D|\mathcal{T}[n - 1]) + (p_1 + p_2) \leq E(D|\mathcal{T}[n - 1]) + (p_i + p_j),$$

$$E(D|\mathcal{T}[n - 1]) + 2(p_1 + p_2) + (p_i + p_j) \leq E(D|\mathcal{T}[n - 1]) + 2(p_i + p_j) + (p_1 + p_2)$$

and the combination of elements 1 and 2 is at least as good as the combination of elements i and j , both $i > 2$ and $j > 2$.

2. Let us consider two possible cases: combination of elements 1 and $j > 3$ and combination of elements 1 and 3.

- 2.1 Combination of elements $i = 1$ and $j > 3$ results in the $(n - 1)$ -set with probabilities $p_2, p_3, \dots, (p_1 + p_j), \dots, p_n$. According to the assumption, the next combination results in the $(n - 2)$ -set with probabilities $p_k, \dots, (p_2 + p_3), \dots, (p_1 + p_j), \dots, p_n$, where, if $j > 4$ then $k = 4$, and $k > 4$ otherwise.

If, instead, at first elements 1 and 2 are combined, then the resulting $(n - 1)$ -set corresponds to the probabilities $p_3, \dots, (p_1 + p_2), \dots, p_n$. In the next step, elements 3 and j can be combined to result in the $(n - 2)$ -set with probabilities $p_k, \dots, (p_1 + p_2), \dots, (p_3 + p_j), \dots, p_n$. As indicated above, the obtained sums $(p_1 + p_2)$ and $(p_3 + p_j)$ cannot give the expected depth $E(D|\mathcal{T}[n])$ greater than the sums $(p_2 + p_3)$ and $(p_1 + p_j)$, that is,

$$\begin{aligned} E(D|\mathcal{T}[n - 2]) + 2(p_1 + p_2) + (p_3 + p_j) &\leq E(D|\mathcal{T}[n - 2]) + 2(p_1 + p_j) \\ &\quad + (p_2 + p_3) \end{aligned}$$

Hence, the combination of elements 1 and 2 is at least as good as the combination of elements $i = 1$ and $j > 3$.

2.2 The other combination in this case is a combination of elements $i = 1$ and $j = 3$. This can result in two possible $(n - 1)$ -sets:

- $(p_1 + p_3) > p_4$ and the $(n - 1)$ -set probabilities are $p_2, p_4, \dots, (p_1 + p_3), \dots, p_n$;
- $(p_1 + p_3) < p_4$ and the $(n - 1)$ -set probabilities are $p_2, (p_1 + p_3), p_4, \dots, p_n$.

The first case is similar to the above case 2.1. Let us consider the second case. If, instead of combining elements 1 and 3, elements 1 and 2 are combined, then for the sums obtained it follows that $(p_1 + p_2) \leq (p_1 + p_3)$. Hence, the expected depth is

$$E(D|\mathcal{T}[n - 1]) + (p_1 + p_2) \leq E(D|\mathcal{T}[n - 1]) + (p_1 + p_3),$$

and the combination of elements 1 and 2 is at least as good as the combination of elements 1 and 3. ■

The procedure that constructs an optimal search tree as specified by Theorem 2.7 is illustrated by the following example.

Example 2.16 As in the previous example, let sample space X include six points with location probabilities $p_i = p(x_i)$, $i = 1, \dots, n$, defined similarly as above:

x_i	x_1	x_2	x_3	x_4	x_5	x_6
p_i	0.2	0.05	0.1	0.4	0.2	0.05

According to Theorem 2.7, the tree is constructed bottom-up, from the leaves to the root, and the nodes, including leaves, are combined according to the smallest probabilities. The resulting tree is shown in Figure 2.25.

As in the previous figure, the indices near the probabilities represent the numbers of the points in sample space X .

According to Equation 2.34, the average number L^* of edge numbers from the root to the leaves in the obtained optimal binary tree is $L^* = 0.2 \cdot 2 + 0.05 \cdot 4 + 0.1 \cdot 3 + 0.4 \cdot 2 +$

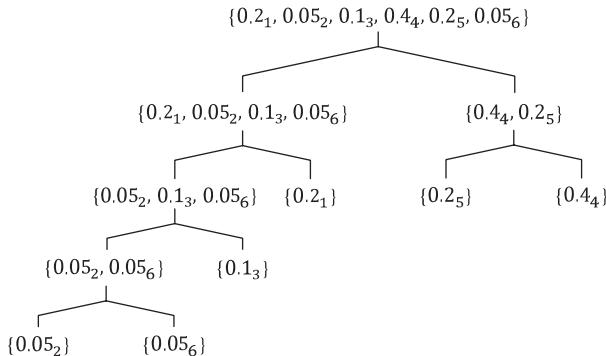


Figure 2.25 Example of optimal binary search tree.

$0.2 \cdot 2 + 0.05 \cdot 4 = 2.3$, which is less than the average number L obtained previously in Example 2.15. However, the depth of the optimal tree is $D(\mathcal{T}[X]) = \max_{1 \leq i \leq n} \{l(x_i)\} = 4$, which is greater than for the tree in Example 2.15. ■

Similar to Procedure 2.4, which constructs a search tree for the ‘division by half’ algorithm, Theorem 2.7 determines the method of constructing an optimal binary search tree. However, in contrast to Procedure 2.4 and the corresponding Algorithm 2.8, in which the tree can be constructed along with the search process, the search using Theorem 2.7 requires two stages. In the first stage, following Theorem 2.7 an optimal binary search tree is constructed starting from the leaves (bottom level) up to the root (upper level), and, in the second stage, the search follows the constructed tree. Since the key idea of the algorithm is to apply an optimal binary tree, which was independently found by Huffman in information theory [58] and by Zimmermann in group-testing search [59], we interchangeably refer to this algorithm as a Huffman or Zimmerman search.

In the Huffman–Zimmerman search algorithm, we use the following functions:

- function $\text{root}(\mathcal{T})$ returns a root node for a given tree \mathcal{T} ;
- function $\text{area}(node)$ returns an area $A \subset X$ that is associated with the $node$; and
- functions $\text{right_successor}(node)$ and $\text{left_successor}(node)$ return right and left successors of the $node$, correspondingly.

In addition, by a *leaf* we denote a node that has no successors. A result of a certain observation of the area $A \subset X$ is denoted, as above, by an indicator $\mathbf{1}(A, x) \in \{0, 1\}$. The search algorithm using the optimal binary search tree is outlined as follows.

Algorithm 2.9 (Huffman–Zimmerman search). Given sample space X with location probabilities p_i , $i = 1, \dots, n$, do:

1. If $\mathbf{1}(X, x) = 0$ then: Return \emptyset .
- Stage 1
2. Construct optimal binary tree $\mathcal{T}[X]$ by using Theorem 2.7.
- Stage 2
3. Set $node = \text{root}(\mathcal{T}[X])$.
4. While $node \neq \text{leaf}$ do:
 - 4.1 Set $\text{observed_node} = \text{right_successor}(node)$.
 - 4.2 Set $A = \text{area}(\text{observed_node})$.
 - 4.3 If $\mathbf{1}(A, x) = 1$ then:
 - 4.3.1 Set $node = \text{observed_node}$.
 - 4.3.2 Else:
 - 4.3.2.1 Set $node = \text{left_successor}(node)$.
5. Return $\text{area}(node)$.

■

This algorithm follows a general procedure of search following a binary search tree. The first line preserves the algorithm from acting over the sample space that does not include any target. If an observation of the complete sample results in 1, then, in Stage 1, the algorithm constructs an optimal binary search tree. In the circle that starts at the fourth line, the algorithm follows the tree from the root and observes areas that are associated with the nodes down to the resulting leaf. Since at each step the area that is associated with the node is observed, the algorithm chooses one of the successors of the node, the right one for certainty (see Line 4.1), and if the associated area includes the target, the algorithm continues with this node (see Line 4.3.1). If, in contrast, this node does not include the target, then the algorithm passes to its sibling, that is, the left node. Since the tree is binary, the previous zero observation result unambiguously shows that the target is in the area associated with this node. Hence, additional observation is not needed and the algorithm continues with this node (see Line 4.3.2).

As indicated above, there is a strong connection between the search using the binary partitioning tree and the methods of optimal coding developed in information theory. To clarify this connection, let us return to the optimal binary search tree shown in Figure 2.25. Each node of the tree corresponds to a subset of the sample space X as represented by the indices of the points, which appear near the probabilities of the points. In addition, as usual for decision trees (see, e.g., the search trees of Sobel and Groll's algorithm, Figure 2.21, and Graff and Roeloffs' algorithm, Figure 2.22), let the numbers '1' and '0' correspond to the branches of the tree; for example, by '1' we denote the 'left' branches and by '0' the 'right' branches. The resulting tree has the same structure as the optimal search tree (see Figure 2.25), and, with the notation for the branches by the numbers '1' and '0,' it represents the *Huffman coding procedure* over the sample space X and is called the *Huffman coding tree*. The procedure for constructing the Huffman tree was presented in 1952 by Huffman [58] and it follows the same steps as the Zimmerman procedure [59], which is presented by Theorem 2.7. The Huffman coding tree is shown in Figure 2.26.

According to the tree, a sequence of ones and zeros corresponds to each leaf, which determines a path from the root to the leaf. Such a sequence is called the *codeword* of the point, which is assigned to the leaf of the tree, and the set of codewords for all the leaves is called the *code* and denoted by $\mathcal{C}[X]$. For the considered tree the correspondence between the points $x_i \in X$ with their probabilities and the codewords $w_i = w(x_i)$ is as follows:

x_i	x_1	x_2	x_3	x_4	x_5	x_6
p_i	0.2	0.05	0.1	0.4	0.2	0.05
w_i	10	1111	110	00	01	1110

Since there is a unique path from the root to each leaf, the obtained codewords w_i unambiguously represent the corresponding points. Moreover, there is no codeword that can be considered as the beginning of another codeword. The code $\mathcal{C}[X]$ with such a property is called the *prefix* or *instantaneous code*.

Now let us clarify the relation between the Huffman code and the inequality in Equation 2.39. Recall that in the partitioning tree $\mathcal{T}[X]$ over the sample space $X = \{x_1, x_2, \dots, x_n\}$ values $l(x_i)$ stand for the number of edges from the root to the leaves $\{x_i\}$, $i = 1, \dots, n$. Since in the Huffman code each codeword $w(x_i)$ includes the ones and zeros assigned

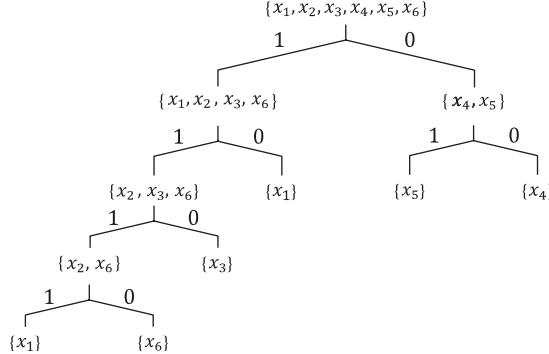


Figure 2.26 Huffman coding tree corresponding to the optimal binary search tree.

to the edges of the tree from the root to the leaf, values $l(x_i)$ are exactly the *lengths of the codewords* $w(x_i)$, and the average number $L = \sum_{i=1}^n p_i l(x_i)$ gives the *expected code-length*.

Consider the lengths $l(x)$ of the codewords $w(x)$, $x \in X$. In the Huffman code over the sample space X (see Figure 2.26), the points with small probabilities correspond to the long codewords, and the points with large probabilities to the short codewords. More precisely, in parallel with Lemmas 2.4 and 2.5, the following statement holds true.

Lemma 2.6 [35, 58]. *Given a finite discrete set $X = \{x_1, x_2, \dots, x_n\}$, for any probability mass function $p : X \rightarrow [0, 1]$ that specifies the probabilities $p_i = p(x_i)$, $i = 1, \dots, n$, of the elements of X there exists a prefix code $\mathcal{C}^*[X]$ with minimal expected code-length $L(\mathcal{C}^*) = \sum_{i=1}^n p_i l^*(x_i)$ such that the following three properties hold:*

1. *If $p_j < p_k$ then $l(x_j) \geq l(x_k)$, $j, k = 1, \dots, n$.*
2. *The two longest codewords have equal length.*
3. *The two longest codewords are siblings, that is, they differ only in the last bit.*
4. *The two longest sibling codewords correspond to two elements of X with the lowest probabilities.*

Proof. Let us consequently consider the outlined properties:

1. Let $\mathcal{C}^*[X]$ be a code with minimal expected code-length, and assume that there is a code $\mathcal{C}[X]$ in which the codewords $w(x_j)$ and $w(x_k)$ from the code $\mathcal{C}^*[X]$ are interchanged. Then the difference between the code-lengths $L(\mathcal{C})$ and $L(\mathcal{C}^*)$ is

$$\begin{aligned} L(\mathcal{C}) - L(\mathcal{C}^*) &= \sum_{i=1}^n p_i l(x_i) - \sum_{i=1}^n p_i l^*(x_i) \\ &= p_j l(x_k) + p_k l(x_j) - p_j l(x_j) - p_k l(x_k) = (p_k - p_j)(l(x_j) - l(x_k)). \end{aligned}$$

According to the requirement $p_k - p_j > 0$, and since $\mathcal{C}^*[X]$ is minimal, it also holds true that $L(\mathcal{C}) - L(\mathcal{C}^*) \geq 0$. Hence, $l(x_j) - l(x_k) \geq 0$ and $l(x_j) > l(x_k)$.

2. Assume, in contrast, that in the minimal code $\mathcal{C}^*[X]$ the two longest codewords have different lengths. Then by preserving the prefix property the last bits can be deleted

from the longer codeword. Such a deletion provides a lower expected code-length that contradicts the assumption that the code $\mathcal{C}^*[X]$ is minimal. Hence, the lengths of the two longest codewords are equal.

3. Assume, in contrast, that in the minimal code $\mathcal{C}^*[X]$ a code of maximal length is without any siblings. Then, similar to the previous property, the last bits from the codeword can be deleted and a lower expected code-length can be obtained, in contradiction with the assumption that the code $\mathcal{C}^*[X]$ is minimal. Hence, each codeword with maximal code-length has a sibling that differs from it only in the last bit.
4. Assume that $w(x_i)$, $i = 1, \dots, 4$, are the longest codewords with equal lengths such that $w(x_1)$ is a sibling of $w(x_2)$ and $w(x_3)$ is a sibling of $w(x_4)$, when $p_2 > p_3$. Then the expected code-length is $L = (p_1 l(x_1) + p_2 l(x_2) + p_3 l(x_3) + p_4 l(x_4) + \sum_{i=5}^n p_5 l(x_5))$. Since the lengths $l(x_i)$, $i = 1, \dots, 4$, are equal, elements x_2 and x_3 can be exchanged, and the expected code-length $L = (p_1 l(x_1) + p_2 l(x_3) + p_3 l(x_2) + p_4 l(x_4) + \sum_{i=5}^n p_5 l(x_5))$ is preserved, equal to the original value. Hence, the longest sibling codewords correspond to the elements with the lowest probabilities. ■

The properties that are specified by Lemma 2.6 determine the procedure for constructing an optimal coding tree, so Theorem 2.7 is a corollary of this lemma. In addition, note that properties 2 and 3 presented by the lemma are not necessary for a minimum of the expected code-length. However, each minimal code can be changed, by preserving the expected code-length, so that these properties hold.

Starting in the 1970s [34, 60] the connection between the search and coding methods was intensively applied for the development of the methods of group-testing search and its optimization, and, in addition to the optimal Huffman search (Zimmerman procedure) above, resulted in a number of effective group-testing procedures [28, 30, 61, 62]. Below we consider the search procedure that provides a basis for these methods.

Let us return to the inequality in Equation 2.39 which, in terms of information theory, specifies the bounds for the minimal expected code-length. The lower bound on the code-length is denoted by the entropy of the codeword probabilities

$$H(X) = -\sum_{i=1}^n p_i \log p_i, \quad (2.40)$$

and is called the *Shannon entropy*, or *entropy* for short. In Equation 2.40 it is assumed that $0 \log 0 = 0$, and that the logarithm is taken to base 2. The concept of entropy was introduced in such a form in 1948 by Shannon [63] as a lower bound of the communication channel capacity. It defines an expected number of binary symbols (*bits*) from the alphabet $\{0, 1\}$ that is required to represent random variable x with alphabet X and probabilities $p(x)$ of the symbols $x \in X$. In general, entropy $H(x)$ is considered as a measure of uncertainty of the random variable x . To stress that the entropy is defined for the probabilities $p(x)$, $x \in X$, it is denoted by $H(p_1, p_2, \dots, p_n)$.

The entropy $H(X)$ as a function of the probabilities $p(x)$, $x \in X$, has the following basic properties [63]:

- Entropy $H(X) \equiv H(p_1, p_2, \dots, p_n)$ is a convex function of its arguments.

- Entropy $H(X) \equiv H(p_1, p_2, \dots, p_n) = 0$ if and only if all probabilities p_i , except one, are equal to 0, while the remaining probability is equal to 1.
- Entropy $H(X) \equiv H(p_1, p_2, \dots, p_n)$ reaches its maximum $\log n$ while all probabilities p_i are equal to $1/n$, that is, $\max_{p_1, p_2, \dots, p_n} H(p_1, p_2, \dots, p_n) = H\left(\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n}\right) = \log n$.

The graph of the entropy $H(p, 1 - p)$ function for a two-point set is shown in Figure 2.27.

The basic properties of the entropy are utilized in the group-testing search procedure that is considered below. Additional properties of the entropy will be presented in the next section along with consideration of its implementation in the search procedures.

As indicated at the beginning of this section, in the case of static target search each level of the tree $T[X]$ forms a set partition system for the sample space X , and the areas, which are associated with the successors of some node, form a set partition system for the area, which is associated with this node [34]. A set of the sets that form a set partition system is called a *partition*. In other words, partition of the set X is a set $\alpha = \{A_1, A_2, \dots, A_m\}$ such that $A_i \cap A_j = \emptyset$, $i \neq j$, and $\cup_{i=1}^m A_i = X$. As in Procedure 2.4, which constructs a binary search tree using ‘division by half,’ let us define the probabilities that the target is located in the areas $A \in \alpha$ as follows:

$$p(A) = \sum_{x \in A} p(x).$$

Since the areas A are disjoint subsets of the sample space X , it follows that $0 \leq p(A) \leq 1$ and $\sum_{A \in \alpha} p(A) = \sum_{x \in X} p(x) = p(X) = 1$.

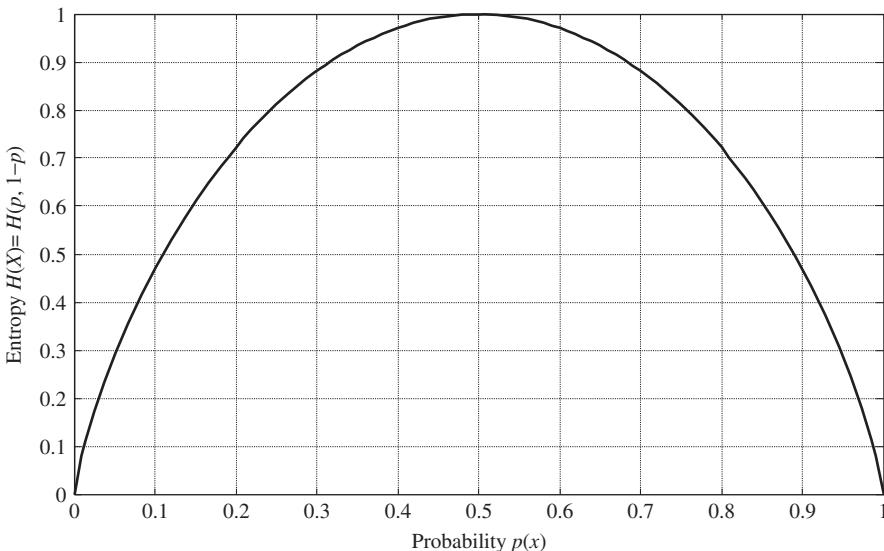


Figure 2.27 Entropy for two-point sample space.

Similar to the entropy $H(X)$ that is defined for the sample space X by Equation 2.40, the *entropy of the partition* α is defined as follows:

$$H(\alpha) = -\sum_{A \in \alpha} p(A) \log p(A), \quad (2.41)$$

where, as above, it is assumed that $p(\emptyset) \log p(\emptyset) = 0 \log 0 = 0$. Since the entropy $H(\alpha)$ of partition α as a function of probabilities has the same form as the entropy $H(X)$ of the sample space X , the basic properties of the entropy $H(X)$ above hold for the entropy $H(\alpha)$.

Let us return to Procedure 2.4, which constructs a binary search tree using ‘division by half,’ and consider the partition that corresponds to each level of the search tree. According to Procedure 2.4, at each step the area that is associated with the node of the tree is divided into two parts such that the probabilities of the parts are as close as possible. In the sense of partitions, this means that at each step, the procedure constructs a partition of the corresponding area such that the entropy of this partition reaches its maximum. In fact, recall that Procedure 2.4 of ‘division by half’ starts from the complete sample space X . Then, at the initial step $t = 0$, $p^0(X) = \sum_{x \in X} p^0(x) = 1$, and, at the first step $t = 1$, division of X into the two areas A_1^1 and A_2^1 such that $|p^0(A_1^1) - p^0(A_2^1)| \rightarrow \min$ leads to $p^0(A_1^1) \rightarrow 1/2$ and $p^0(A_2^1) \rightarrow 1/2$. Hence, the entropy of the partition $\alpha^1 = \{A_1^1, A_2^1\}$ is $H(\alpha^1) \rightarrow H(1/2, 1/2)$, which, according to the properties of the entropy, provides its maximal value. In the second step, each area A_1^1 and A_2^1 is considered as a complete sample space. Hence, the probabilities of the points from A_1^1 and A_2^1 are normalized (according to the Bayes theorem, see, for example, [64]) so that the resulting probabilities are specified as $p^1(x) = p^0(x)/p^0(A_1^1)$ for $x \in A_1^1$, and $p^1(x) = p^0(x)/p^0(A_2^1)$ for $x \in A_2^1$. With the use of these probabilities $p^1(A_1^1) = \sum_{x \in A_1^1} p^1(x) = 1$ and $p^1(A_2^1) = \sum_{x \in A_2^1} p^1(x) = 1$. Now, each of the two areas A_1^1 and A_2^1 is divided into two areas according to the above rule and results in two partitions, partition α_1^2 of the area A_1^1 and partition α_2^2 of the area A_2^1 , such that the entropies $H(\alpha_1^2)$ and $H(\alpha_2^2)$ reach their maximums. In the next step, the obtained areas are considered, and the process continues down to the leaves of the tree. To clarify the correspondence between the search tree and the entropies of the partitions, let us consider the following example.

Example 2.17 Assume that the sample space is $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, and consider the search tree with the probabilities shown in Figure 2.23.

The complete sample space X implies the partition $\alpha^0 = \{\{x_1, x_2, x_3, x_4, x_5, x_6\}, \emptyset\}$ with entropy $H(\alpha^0) = H(1, 0) = 0$. The space X is divided into two areas with equal probabilities that at step $t = 1$ results in the partition $\alpha^1 = \{\{x_2, x_6, x_4\}, \{x_1, x_3, x_5\}\}$ with the maximal entropy $H(\alpha^1) = H(0.5, 0.5) = 1$ (see Figure 2.27).

In the next step $t = 2$, each area $\{x_2, x_6, x_4\}$ and $\{x_1, x_3, x_5\}$ from the partition α^1 is divided into two parts and results in two partitions, $\alpha_1^2 = \{\{x_2, x_6\}, \{x_4\}\}$ and $\alpha_2^2 = \{\{x_1, x_3\}, \{x_5\}\}$. The entropy of the first partition is

$$H(\alpha_1^2) = H\left(\frac{0.05 + 0.05}{0.05 + 0.05 + 0.4}, \frac{0.4}{0.05 + 0.05 + 0.4}\right) = 0.722$$

and the entropy of the second partition is

$$H(\alpha_2^2) = H\left(\frac{0.2 + 0.1}{0.2 + 0.1 + 0.2}, \frac{0.2}{0.2 + 0.1 + 0.2}\right) = 1.058.$$

In both cases, the entropies are calculated for the normalized probabilities of the areas and reach their maximums for the divided areas $\{x_2, x_6, x_4\}$ and $\{x_1, x_3, x_5\}$.

Finally, the areas from the partitions α_1^2 and α_2^2 are divided into two parts, while the leaves $\{x_4\}$ and $\{x_5\}$ obtained already are considered as single-point sample spaces. The resulting partitions at $t = 3$ are: $\alpha_1^3 = \{\{x_2\}, \{x_6\}\}$, $\alpha_2^3 = \{\{x_4\}, \emptyset\}$, $\alpha_3^3 = \{\{x_1\}, \{x_3\}\}$, and $\alpha_4^3 = \{\{x_5\}, \emptyset\}$. The entropies of these partitions, which are calculated for the normalized probabilities, are as follows:

$$\begin{aligned} H(\alpha_1^3) &= H\left(\frac{0.05}{0.05+0.05}, \frac{0.05}{0.05+0.05}\right) = 1, & H(\alpha_2^3) &= H\left(\frac{0.4}{0.4}, 0\right) = 0, \\ H(\alpha_3^3) &= H\left(\frac{0.2}{0.2+0.1}, \frac{0.1}{0.2+0.1}\right) = 0.918, & H(\alpha_4^3) &= H\left(\frac{0.2}{0.2}, 0\right) = 0. \end{aligned}$$

For each level of the tree, the partitions can be combined into partitions of the sample space X as follows:

- $\alpha^0 = \{\{x_1, x_2, x_3, x_4, x_5, x_6\}, \emptyset\}$,
- $\alpha^1 = \{\{x_1, x_3, x_5\}, \{x_2, x_6, x_4\}\}$,
- $\alpha^2 = \alpha_1^2 \cup \alpha_2^2 = \{\{x_1, x_3\}, \{x_2, x_6\}, \{x_4\}, \{x_5\}\}$,
- $\alpha^3 = \alpha_1^3 \cup \alpha_2^3 \cup \alpha_3^3 \cup \alpha_4^3 = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\}, \{x_6\}\}$.

The entropies of the partitions α^0 and α^1 , already obtained, are $H(\alpha^0) = 0$ and $H(\alpha^1) = 1$, and the entropies of the remaining partitions are $H(\alpha^2) = H((0.05 + 0.05), 0.4, (0.2 + 0.1), 0.2) = 1.846$ and $H(\alpha^3) = H(0.2, 0.05, 0.1, 0.4, 0.2, 0.05) = 2.222$. ■

From Example 2.17, it follows that the entropy $H(\alpha')$ of the partitions $\alpha^0, \dots, \alpha^3$ corresponding to Procedure 2.4 increases from the top level to the bottom level, and direct calculation shows that a similar property holds for the partitions that correspond to the optimal procedure specified by Theorem 2.7. Such an observation leads to the group-testing algorithm that, on the one hand, acts similarly to the ‘division by half’ algorithm from the top level to the bottom level of the search tree, and, on the other hand, by use of a certain cost function, it results in a tree that is close to the optimal binary search tree. Such an algorithm was suggested by Hartmann *et al.* [62] and is considered below.

The Generalized Optimal Testing Algorithm (GOTA) was suggested by Hartmann *et al.* [62] as a modification of a method for converting decision tables to information theory algorithms [60]. Further generalization of this approach was later proposed by Goodman and Smyth [61]. Similar to the ‘division by half’ Procedure 2.4, the GOTA starts from the sample space and divides it into parts, while the divisions are restricted and can be conducted by a predefined set of rules. In addition, the GOTA allows possible actions to be taken into account so that the search can be terminated by allocation of the target to some observed area, which includes more than one point. In such a case, the leaves of the search tree, which is created by the GOTA, are allowed to be associated with the non-single-point areas.

Formally, let $X = \{x_1, x_2, \dots, x_n\}$ be a sample space with location probabilities $p_i = p(x_i)$, $i = 1, \dots, n$, and recall that $\mathcal{X} = 2^X$ denotes the set of all possible subsets of X . Denote by $\mathbb{T} : \mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}$ a mapping such that, for any area $A \in \mathcal{X}$, it results in a pair

$\{A_1, A_2\}$ of areas such that $A_1 \cap A_2 = \emptyset$ and $A_1 \cup A_2 = A$, where one of the areas A_1 or A_2 , but indeed not both, can be empty. Assume that for a given search process, a finite set of the mappings \mathbb{T} is defined. Mappings \mathbb{T} represent the rules of divisions mentioned above. In the sense of Procedure 2.4 ('division by half') such mappings specify the division of the area into two paths, while for certain areas such divisions are not available. Similar to the search tree, which is used in Procedure 2.4 (see Example 2.17), starting from the root level, the rules \mathbb{T}_j , $j = 1, 2, \dots$, generate partitions $\alpha^0, \alpha^1, \dots$ of the sample space X that correspond to the levels of the search tree. In addition, assume that there is a finite set of actions defined such that a certain action \mathbb{A}_k , $k = 1, 2, \dots$, can be applied to more than one point x of the sample space X . The applicability of the actions \mathbb{A} generates a *final partition* γ of the sample space X that specifies the areas associated with the leaves of the search tree, and to which the actions can be applied. Correspondence between the division rules \mathbb{T} and actions \mathbb{A} and the points $x \in X$ can be presented in the form of a *decision table*. The structure of such a table and its representation in the form of partitions are illustrated by the following example [62].

Example 2.18 [62]. Assume that the sample space is $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and the location probabilities are the same as before in Examples 2.15 and 2.16. Let, in addition, five division rules $\mathbb{T}_1, \dots, \mathbb{T}_5$ be defined such that:

- \mathbb{T}_1 distinguishes between the points x_1, x_2, x_3, x_6 and the points x_4, x_5 ;
- \mathbb{T}_2 distinguishes between the points x_1, x_4, x_5, x_6 and the points x_2, x_3 ;
- \mathbb{T}_3 distinguishes between the points x_1, x_3, x_4, x_5 and the points x_2, x_6 ;
- \mathbb{T}_4 distinguishes between the points x_1, x_3 and the points x_2, x_4, x_5, x_6 ; and
- \mathbb{T}_5 distinguishes between the points x_1, x_4 and the points x_2, x_3, x_4, x_6 .

Finally, assume that there are four actions $\mathbb{A}_1, \dots, \mathbb{A}_4$ such that:

- \mathbb{A}_1 is applicable to the points x_4 and x_5 ;
- \mathbb{A}_2 is applicable to the point x_3 ;
- \mathbb{A}_3 is applicable to the points x_1 and x_2 ; and
- \mathbb{A}_4 is applicable to the point x_6 .

The decision table that represents the sample space and division rules and actions is as follows:

Locations	x_i	x_1	x_2	x_3	x_4	x_5	x_6
Probabilities	p_i	0.2	0.05	0.1	0.4	0.2	0.05
Actions	\mathbb{A}_j	\mathbb{A}_3	\mathbb{A}_3	\mathbb{A}_2	\mathbb{A}_1	\mathbb{A}_1	\mathbb{A}_4
Division rules	\mathbb{T}_1	0	0	0	1	1	0
	\mathbb{T}_2	1	0	0	1	1	1
	\mathbb{T}_3	0	1	0	0	0	1
	\mathbb{T}_4	0	1	0	1	1	1
	\mathbb{T}_5	0	1	1	0	1	1

According to the table, an application of the rule \mathbb{T}_1 to the sample space X results in

$$\mathbb{T}_1(X) = \{\{x_1, x_2, x_3, x_6\}, \{x_4, x_5\}\},$$

while its application to the other areas, for example, $\{x_1, x_2, x_4\}$ and $\{x_1, x_2\}$, results in $\mathbb{T}_1(\{x_1, x_2, x_4\}) = \{\{x_1, x_2\}, \{x_4\}\}$ and $\mathbb{T}_1(\{x_1, x_2\}) = \{\{x_1, x_2\}, \emptyset\}$. Similarly, an application of the rule \mathbb{T}_2 to the sample space X gives

$$\mathbb{T}_2(X) = (\{x_1, x_4, x_5, x_6\}, \{x_2, x_3\}),$$

while, for example, $\mathbb{T}_2(\{x_2, x_4, x_5\}) = \{\{x_4, x_5\}, \{x_2\}\}$ and $\mathbb{T}_2(\{x_2, x_3\}) = \{\{x_2, x_3\}, \emptyset\}$. In the same manner, for the remaining three rules it follows that

$$\mathbb{T}_3(X) = \{\{x_1, x_3, x_4, x_5\}, \{x_2, x_6\}\},$$

$$\mathbb{T}_4(X) = \{\{x_1, x_3\}, \{x_2, x_4, x_5, x_6\}\},$$

$$\mathbb{T}_5(X) = \{\{x_1, x_4\}, \{x_2, x_3, x_5, x_6\}\}.$$

Finally, from the applicability of the actions $\mathbb{A}_1, \dots, \mathbb{A}_4$ it follows that the sample space X is divided into the following areas:

$$\mathbb{A}_1 : \{x_4, x_5\}, \mathbb{A}_2 : \{x_3\}, \mathbb{A}_3 : \{x_1, x_2\}, \mathbb{A}_4 : \{x_6\}$$

Since these areas are disjoint, while their union results in the complete sample space X , the areas can be combined into the partition

$$\gamma = \{\{x_1, x_2\}, \{x_3\}, \{x_4, x_5\}, \{x_6\}\}.$$

In the sense of a decision table, partition γ represents a minimal division of the sample space, for which the indicated actions can be applied. Below, we will use this example to consider the search tree that is created by the GOTA. ■

As illustrated by Example 2.18, an application of the division rules $\mathbb{T} : \mathcal{X} \rightarrow \mathcal{X} \times \mathcal{X}$ and consideration of the applicability of the actions \mathbb{A} is represented by the partitions of the sample space X . Let us define the GOTA that constructs such partitions in a similar top-down manner to that conducted by the ‘division by half’ Procedure 2.4 (see Example 2.17).

The actions of the GOTA are based on the following value. Let $\alpha = \{A | A \subset X\}$ and $\beta = \{B | B \subset X\}$ be two partitions of the sample space X . Recall that for α and β it follows that $\cup_{A \in \alpha} A = X$, where $A_i \cap A_j = \emptyset$ if $i \neq j$, and $\cup_{B \in \beta} B = X$, where $B_i \cap B_j = \emptyset$ if $i \neq j$. The *conditional entropy of partition β given partition α* is defined as follows [65]:

$$H(\beta|\alpha) = -\sum_{A \in \alpha} \sum_{B \in \beta} p(B, A) \log p(B|A), \quad (2.42)$$

where $p(B, A) = p(B \cap A)$ and $p(B|A) = p(B \cap A)/p(A)$. As indicated above, it is assumed that $0 \log 0 = 0$.

Assume that over a sample space there is some binary search tree $\mathcal{T}[X]$ of depth $D(\mathcal{T}[X])$. Let $\langle \alpha^0, \alpha^1, \dots \rangle$ be a sequence of partitions of X that correspond to the levels of

the tree, where $\alpha^0 = \{X, \emptyset\}$ is the partition that corresponds to the root level, and let γ be the final partition. The *conditional information gain* between levels t and $t + 1$ of the tree $\mathcal{T}[X]$ given a final partition γ is defined as follows:

$$I(\alpha^t, \alpha^{t+1} | \gamma) = H(\gamma | \alpha^t) - H(\gamma | \alpha^{t+1}), \quad (2.43)$$

where $t = 0, 1, \dots, (D(\mathcal{T}[X]) - 1)$.

The termination condition of the GOTA is based on the *refinement relation* ' \prec ' which is defined as follows. Let $\alpha = \{A | A \subset X\}$ and $\beta = \{B | B \subset X\}$ be two partitions of the sample space X . It is said that partition α *refines* partition β , written as $\beta \prec \alpha$, if for every set $A \in \alpha$ there exists a set $B \in \beta$ such that $A \subset B$. By direct calculation using Equation 2.42 it is easy to show that $\beta \prec \alpha$ if and only if $H(\beta | \alpha) = 0$. Then the GOTA terminates if partition α^t , which is obtained by using the available division rules \mathbb{T}_j , $j = 1, 2, \dots$, refines the final partition γ , which is specified by the actions \mathbb{A}_k , $k = 1, 2, \dots$

Finally, assume that for any pair (α, β) of partitions of the sample space X , a strictly positive *cost* value $c(\alpha, \beta) > 0$ is defined.

Let us define a function that, for any partition β of the sample space X and set of division rules \mathbb{T}_j , $j = 1, \dots, |\beta|$, results in the partition α such that $\beta \prec \alpha$ and $|\beta| + 1 \leq |\alpha| \leq 2 \cdot |\beta|$, where the size of the trivial partition $\beta = \{X, \emptyset\}$ is $|\beta| = 1$:

next_partition_GOTA($\beta, \{\mathbb{T}_j | j = 1, \dots, |\beta|\}$):

1. Set $\alpha = \emptyset$.
2. For $j = 1, \dots, |\beta|$ do:
 - 2.1 Set $\alpha_j = \mathbb{T}_j(B_j)$, $B_j \in \beta$: Creating partition of the area B_j using the rule \mathbb{T}_j .
 - 2.2 Set $\alpha = \alpha \cup \alpha_j$: Updating resulting partition α with α_j .
3. Return α .

In addition to the functions over the tree that are used in Algorithm 2.9, we apply the following functions defined for the tree and the areas:

- function *create_node()* that creates a *node*, and functions *create_right_successor(node)* and *create_left_successor(node)* that create sibling successors of the *node*;
- function *associate(node, area)* that associates a *node* to the *area* $A \subset X$;
- function *get_node(area)* that returns a node which is associated with the *area*.

Then, the GOTA, which acts in a top-down manner and creates a set partition system that corresponds to the binary search tree $\mathcal{T}[X]$, is defined as follows.

Algorithm 2.10 (Generalized Optimal Testing Algorithm) [62]. Given a sample space $X = \{x_1, x_2, \dots, x_n\}$ with location probabilities p_i , $i = 1, \dots, n$, and finite sets of available division rules \mathbb{T}_j , $j = 1, 2, \dots, l$, and actions \mathbb{A}_k , $k = 1, 2, \dots, m$, and a strictly positive cost function c , do:

Stage 1: Define final partition γ according to the applicability of the actions \mathbb{A}_k .

1. Set $\gamma = \emptyset$.

2. For each action \mathbb{A}_k , $k = 1, 2, \dots, m$, do:

2.1 Specify A as a set of points $x \in X$, to which action \mathbb{A}_k is applicable.

2.2 Include A in γ : Set $\gamma = \gamma \cup \{A\}$.

Stage 2: Create search tree $\mathcal{T}[X]$ according to the division rules \mathbb{T}_j .

1. Set $t = 0$ and Set $A^t = X$.

2. Set $root = create_node()$ and $associate(root, A^t)$.

3. Set $\alpha^t = \{A^t, \emptyset\}$.

4. While $H(\gamma | \alpha^t) \neq 0$ do:

Create next partition α^{t+1}

6.1 Choose a set of $|\alpha^t|$ division rules \mathcal{T}_j such that for the next partition

$$\alpha^{t+1} = next_partition_GOTA(\alpha^t, \{\mathcal{T}_j\})$$

it follows that $I(\alpha^t, \alpha^{t+1} | \gamma) / c(\alpha^t, \alpha^{t+1})$ reaches its maximum.

Create next level of the search tree $\mathcal{T}[X]$

6.2 For $j = 1, \dots, |\alpha^t|$ do:

6.2.1 Select $A_j^t \in \alpha^t$.

6.2.2 If $|A_j^t| > 1$ then:

a. Set $node = get_node(A_j^t)$.

b. Set $left = create_left_successor(node)$.

c. Set $right = create_right_successor(node)$.

d. Set $\alpha_j^{t+1} = \mathbb{T}_j(A_j^t)$; obtained partition is $\alpha_j^{t+1} = \{A_{j1}^{t+1}, A_{j2}^{t+1}\}$.

e. Select $A_{j1}^{t+1} \in \alpha_j^{t+1}$ and $associate(left, A_{j1}^{t+1})$.

f. Select $A_{j2}^{t+1} \in \alpha_j^{t+1}$ and $associate(right, A_{j2}^{t+1})$.

6.3 Set $t = t + 1$.

5. Return $\mathcal{T}[X]$. ■

First, Algorithm 2.10 creates a final partition γ according to the applicability of the given actions, and then it creates a search tree $\mathcal{T}[X]$ by using the available division rules. Similar to the ‘division by half’ algorithm (see Algorithm 2.8), Algorithm 2.10 starts from the complete sample space X and terminates with the partition α^t that refines the given finite partition γ . The sample space X is associated with the root of the search tree $\mathcal{T}[X]$, and the areas from the partition α^t are associated with its leaves. The areas that are included in the partitions, which are obtained at the intermediate stages of the algorithm, are associated with the nodes of the tree.

At each stage, the decision regarding the next tree level (see Line 6.1 of the algorithm) is obtained on the basis of the created partition, and the criterion for decision making is defined

by the relation $I(\alpha^t, \alpha^{t+1}|\gamma)/c(\alpha^t, \alpha^{t+1})$ of relative information gain (see Equation 2.43) to the strictly positive cost that meets the requirement of Equation 2.44. On the basis of this decision, the corresponding level of the tree is created (see Lines 6.2). In order to clarify the actions of the algorithm, the presented outline of the stages of decision making and of creating tree level are given separately, but in the implementations they can be conducted simultaneously. The actions of the algorithm are clarified by the following example which continues Example 2.18.

Example 2.19 Let sample space $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and location probabilities $p_i = p(x_i)$, $i = 1, \dots, 6$, be defined as before in Example 2.18. Assume that available division rules T_1, \dots, T_5 and actions A_1, \dots, A_4 also follow the decision table presented in Example 2.18. In addition, for any $t = 0, 1, \dots$ let the cost value $c(\alpha^t, \alpha^{t+1})$ be defined as a sum of probabilities $p(x_i)$ over the points x_i that appear at the $(t + 1)$ th level of the tree. Then, the actions of Algorithm 2.10 are as follows.

In the first stage, the algorithm creates final partition γ according to the applicability of the actions A_1, \dots, A_4 . As indicated in Example 2.18, such a partition is

$$\gamma = \{\{x_1, x_2\}, \{x_3\}, \{x_4, x_5\}, \{x_6\}\}.$$

In the second stage, the algorithm creates a search tree $\mathcal{T}[X]$. Let us consider an initial step $t = 0$. At this step, the root node of the tree is created and sample space X is associated with this node. Partition α^0 that corresponds to this zero level of the tree $\mathcal{T}[X]$ is

$$\alpha^0 = \{X, \emptyset\} = \{\{x_1, x_2, x_3, x_4, x_5, x_6\}, \emptyset\}.$$

The entropy of this partition is $H(\alpha^0) = H(1, 0) = 0$, while the conditional entropy $H(\gamma|\alpha^0)$ of the final partition γ , given partition α^0 calculated according to Equation 2.42, is as follows:

$$\begin{aligned} H(\gamma|\alpha^0) &= -\sum_{A \in \alpha^0} \sum_{B \in \gamma} p(B \cap A) \log \frac{p(B \cap A)}{p(A)} \\ &= -\sum_{B \in \gamma} p(B) \log p(B) - \sum_{B \in \gamma} 0 \log 0 \\ &= H(\gamma) = -0.25 \log 0.25 - 0.1 \log 0.1 - 0.6 \log 0.6 - 0.05 \log 0.05 \\ &= 1.490 \end{aligned}$$

Let us consider partitions α_j^1 , $j = 1, \dots, 5$, that are the candidates for partition α^1 . These partitions are created by the division rules T_1, \dots, T_5 by their application to the sample space X . Corresponding to the rules T_1, \dots, T_5 , we have (see Example 2.18)

$$\begin{aligned} \alpha_1^1 &= \{\{x_1, x_2, x_3, x_6\}, \{x_4, x_5\}\}, & \alpha_2^1 &= \{\{x_1, x_4, x_5, x_6\}, \{x_2, x_3\}\}, \\ \alpha_3^1 &= \{\{x_1, x_3, x_4, x_5\}, \{x_2, x_6\}\}, & \alpha_4^1 &= \{\{x_1, x_3\}, \{x_2, x_4, x_5, x_6\}\}, \\ \alpha_5^1 &= \{\{x_1, x_4\}, \{x_2, x_3, x_5, x_6\}\}. \end{aligned}$$

The entropies of these partitions are

$$H(\alpha_1^1) = H(0.4, 0.6) = 0.971, \quad H(\alpha_2^1) = H(0.85, 0.15) = 0.610,$$

$$H(\alpha_3^1) = H(0.9, 0.1) = 0.469, \quad H(\alpha_4^1) = H(0.3, 0.7) = 0.881,$$

$$H(\alpha_5^1) = H(0.6, 0.4) = 0.971,$$

while the conditional entropies $H(\gamma|\alpha_j^1)$, $j = 1, \dots, 5$, of the final partition γ given partition α_j calculated according to Equation 2.42 are

$$H(\gamma|\alpha_1^1) = 0.520, \quad H(\gamma|\alpha_2^1) = 1.061,$$

$$H(\gamma|\alpha_3^1) = 1.202, \quad H(\gamma|\alpha_4^1) = 0.790,$$

$$H(\gamma|\alpha_5^1) = 1.101.$$

In this step, for each partition α_j^1 , $j = 1, \dots, 5$, the defined cost value $c(\alpha^0, \alpha_j^1) = \sum_{A \in \alpha_j^1} \sum_{x \in A} p(x)$ includes the probabilities of all points of the sample space; hence $c(\alpha^0, \alpha_j^1) = 1$ for each $j = 1, \dots, 5$.

For the cost value $c(\alpha^0, \alpha_j^1) = 1$, $j = 1, \dots, 5$, the maximum of the relation $\frac{I(\alpha^0, \alpha_j^1|\gamma)}{c(\alpha^0, \alpha_j^1)} = \frac{H(\gamma|\alpha^0) - H(\gamma|\alpha_j^1)}{c(\alpha^0, \alpha_j^1)}$ over the candidate partitions α_j^1 is reached for the partition that provides minimal conditional entropy $H(\gamma|\alpha_j^1)$ of the final partition. For the considered candidates, the minimum is reached on the partition α_1^1 , thus the next partition is specified as

$$\alpha^1 = \alpha_1^1 \{\{x_1, x_2, x_3, x_5, x_6\}, \{x_4, x_5\}\},$$

and the division rule that has to be applied to the sample space X to obtain α^1 is \mathbb{T}_1 .

The partition α^1 corresponds to the first level of the tree $\mathcal{T}[X]$. To obtain the partition that corresponds to the second level of the search tree, it is necessary to consider possible pairs of division rules applied to the areas $A_1^1 \in \alpha^1$ and $A_2^1 \in \alpha^1$. Since an application of the rule \mathbb{T}_1 to both areas of the partition α^1 does not change them, this rule can be removed from consideration. The candidate partitions α_j^2 , $j = 2, \dots, 5$, that are generated by the division rules $\mathbb{T}_2, \dots, \mathbb{T}_5$ are as follows:

$$\alpha_2^2 = \{\{x_1, x_6\}, \{x_2, x_3\}, \{x_4, x_5\}\}, \quad \alpha_3^2 = \{\{x_1, x_3\}, \{x_2, x_6\}, \{x_4, x_5\}\},$$

$$\alpha_4^2 = \{\{x_1, x_3\}, \{x_2, x_6\}, \{x_4, x_5\}\}, \quad \alpha_5^2 = \{\{x_1\}, \{x_2\}, \{x_3, x_6\}, \{x_4, x_5\}\}.$$

Note that $\alpha_3 = \alpha_4$.

The entropies of these partitions are

$$H(\alpha_2^2) = H(0.25, 0.15, 0.6) = 1.353, \quad H(\alpha_3^2) = H(0.1, 0.3, 0.6) = 1.300,$$

$$H(\alpha_4^2) = H(0.1, 0.3, 0.6) = 1.300, \quad H(\alpha_5^2) = H(0.2, 0.2, 0.6) = 1.371.$$

while the conditional entropies $H(\gamma|\alpha_j^2)$, $j = 2, \dots, 5$, of the final partition γ given partition α_j^2 are as follows:

$$H(\gamma|\alpha_2^2) = 0.318, \quad H(\gamma|\alpha_3^2) = 0.376,$$

$$H(\gamma|\alpha_4^2) = 0.376, \quad H(\gamma|\alpha_5^2) = 0.300.$$

Since there is no division rule that divides points x_4 and x_5 , and area $\{x_4, x_5\}$ is an element of the final partition γ , this area is associated with a leaf of the tree $\mathcal{T}[X]$. Thus the cost $c(\alpha^1, \alpha_j^2)$ for candidate partitions α_j^2 , $j = 2, \dots, 5$, is a sum of probabilities of all points except x_4 and x_5 , and $c(\alpha^1, \alpha_j^2) = 0.4$ for each candidate partition α_j^2 .

The maximum value of the relation $\frac{I(\alpha^1, \alpha_j^2 | \gamma)}{c(\alpha^1, \alpha_j^2)} = \frac{H(\gamma | \alpha^1) - H(\gamma | \alpha_j^2)}{c(\alpha^1, \alpha_j^2)}$ over the candidate partitions α_j^2 is reached for the partition that provides minimal conditional entropy $H(\gamma | \alpha_j^2)$, namely, partition α_5^2 . Hence, the next partition is specified as

$$\alpha^2 = \alpha_5^2 = \{\{x_1\}, \{x_2, x_3, x_6\}, \{x_4, x_5\}\},$$

and the division rule that has to be applied to the sample space X to obtain α^2 is \mathbb{T}_5 .

The partition α^2 corresponds to the second level of the tree $\mathcal{T}[X]$. By a similar consideration for the third level of the tree, application of the division rule \mathbb{T}_4 to the partition α^2 results in the partition

$$\alpha^3 = \alpha_4^3 = \{\{x_1\}, \{x_3\}, \{x_2, x_6\}, \{x_4, x_5\}\},$$

and application of the division rule \mathbb{T}_2 to the partition α^3 results in the partition

$$\alpha^4 = \alpha_2^4 = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4, x_5\}, \{x_6\}\},$$

which corresponds to the fourth level of the tree.

Since the conditional entropy of the final partition γ given partition α^4 is $H(\gamma | \alpha^4) = 0$, the algorithm terminates. The search tree obtained with the corresponding partitions is shown in Figure 2.28.

On the left side of Figure 2.28, the obtained search tree is presented, and on the right the partitions that correspond to the levels of the tree are listed. The last partition α^4 is a refinement of the final partition $\gamma = \{\{x_1, x_2\}, \{x_3\}, \{x_4, x_5\}, \{x_6\}\}$, in particular the area $\{x_1, x_2\}$, which appears in the final partition γ ; in the partition α^4 is divided into two points, $\{x_1\}$ and $\{x_2\}$. ■

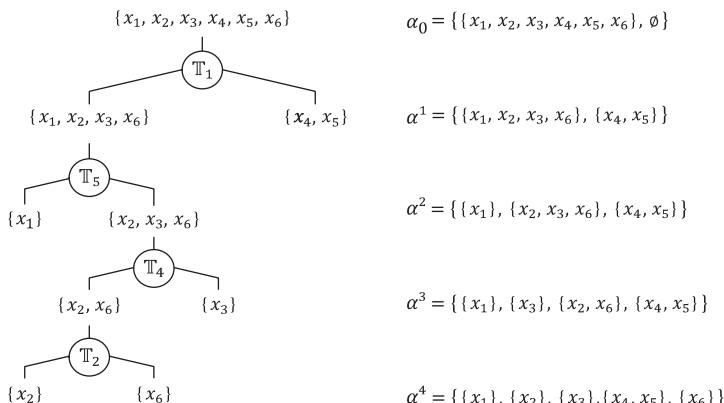


Figure 2.28 Example of the GOTA search tree.

The search tree that is obtained by the GOTA (Algorithm 2.10) depends on the available division rules, which specify partitions α^t , and on the costs $c(\alpha^t, \alpha^{t+1})$. In particular, the GOTA search tree, which is shown in Figure 2.28, is equivalent to the optimal Huffman tree (see Figures 2.25 and 2.26), except for the branch that follows the area $\{x_4, x_5\}$. However, if all possible division rules are available, the same process creates a search tree that would be equivalent to the tree created by the ‘division by half’ Procedure 2.4 (see Figure 2.23).

In general, in the GOTA, the relation between the total cost, which is represented by the costs sum, and the entropies of partitions is specified by the following bounds [62]. Let $\mathcal{T}[X]$ be a binary search tree that is created by the GOTA, and assume that the depth of the tree is $D(\mathcal{T}[X])$. Denote by $C(\mathcal{T}[X]) = \sum_{t=0}^{D(\mathcal{T}[X])-1} c(\alpha^t, \alpha^{t+1})$ the total cost of creating the tree $\mathcal{T}[X]$. Then, if for any partitions α , β , and ξ of X such that $\alpha \prec \xi \prec \beta$, it follows that

$$c(\alpha, \beta) \leq c(\alpha, \xi) + c(\xi, \beta), \quad (2.44)$$

and the total cost $C(\mathcal{T}[X])$ is bounded as follows [62]:

$$\frac{H(X)}{\max_t \{I(\alpha^t, \alpha^{t+1}|\gamma)/c(\alpha^t, \alpha^{t+1})\}} \leq C(\mathcal{T}[X]) \leq \frac{H(X)}{\min_t \{I(\alpha^t, \alpha^{t+1}|\gamma)/c(\alpha^t, \alpha^{t+1}|\gamma)\}}.$$

These bounds are similar to the bounds (Equation 2.39) on the number $L = \sum_{i=1}^n p_i l(x_i)$ in the Huffman–Zimmerman search, which, using the entropy (Equation 2.40), are written as follows:

$$H(X) \leq L < H(X) + 1.$$

However, note that in contrast to the bounds in Equation 2.39, the bounds of the total cost $C(\mathcal{T}[X])$ for the GOTA cannot be used to predict the effectiveness of the algorithm with respect to the given data. The values of $\min_t \{I(\alpha^t, \alpha^{t+1}|\gamma)/c(\alpha^t, \alpha^{t+1})\}$ and $\max_t \{I(\alpha^t, \alpha^{t+1}|\gamma)/c(\alpha^t, \alpha^{t+1})\}$, which are required to calculate the bounds, are obtained by the GOTA (Algorithm 2.10) without implementing the part which creates the search tree (Lines 6.2).

The basic group-testing search algorithms are applicable to the cases of search for a single static target by a single searcher. In the last few decades, within the framework of the information theory approach, the group-testing search has been implemented for the search by multiple searchers and for the search for number of targets. In addition, certain algorithms of search for a moving target were suggested and studied. In the following sections, we will consider such algorithms, in particular a Huffman-type algorithm for the search by a number of searchers, developed by Abrahams [66], and informational algorithms for creating classification trees [67, 68] and for searching [69] that act directly on the space of partitions of the sample space.

2.3 Path planning and search over graphs

A *search over graphs* represents different types of problems and provides the most general description of path-planning processes and optimization problems that appear in the field of AI and require optimal path planning. In such problems, the locations of the searcher and the target are associated with *initial* and *final* vertexes, respectively. The searcher starts at the initial vertex and, following along the edges of the graph, reaches the final vertex,

at which the target is located. The resulting sequence of edges is considered as a search path, and the goal of the searcher is to find the shortest path between the initial and final vertexes. In the case of MTS, the location of the target at each step of search is unknown, while its starting vertex and rule of movement over the graph are available to the searcher.

The studies of algorithms for finding an optimal path between initial and finite vertexes in the graph were initiated in 1956 by Dejkstra, who published in 1959 the first algorithm [70] to follow the greedy *best-first strategy*. A similar algorithm was independently designed in 1957 and published in 1959 by Moore [71]. Furthermore, Walden, in a historical note [72], gives credit to Ford [73] and Bellman [74], who considered the same methods in the context of routing and dataflow in networks. The algorithm acts over weighted graphs and, under an assumption that the weights of edges are additive, results in a suboptimal path between the initial and final vertexes (for a modern description of the algorithm see, e.g., [44]). In a later period, the intensive studies of path-planning and search problems over graphs resulted in different types of algorithms that are implemented for various tasks.

Classical search and path-planning algorithms [75], especially the popular A* algorithm [76], correspond to search for a static target, while newer algorithms [77, 78], which follow the line of A*, were modified to comply with a moving target search. Below, we consider the most useful basic algorithms of path planning and search over graphs both for static and for moving targets.

2.3.1 General BF* and A* algorithms

Let us start with an algorithm for heuristic search on a graph, which applies the *general best-first (GBF) strategies* [75]. Algorithms that follow GBF strategies are called *GBF algorithms*, and are applied to solve the problem of optimal path planning on finite-oriented graphs.

Let $\mathcal{G} = (S, E)$ be an oriented finite graph with a set S of vertexes (or nodes) and a set of edges $E \subset S \times S$. It is also allowed that both vertexes $s \in S$ and edges $e \in E$ are weighted by certain real numbers. In the context of path-planning and search algorithms the vertexes $s \in S$ of the graph \mathcal{G} with assigned weights or *costs* are called *states*. An *optimal path problem* on the graph $\mathcal{G} = (S, E)$ is defined as follows. Let $s_{init} \in S$ be an *initial state* and $s_{fin} \in S$ a *final state*. Then, the optimal path problem is a problem of finding a *path* $a[s_{init}, s_{fin}] = \langle (s_{init} = s^0, s^1), (s^1, s^2), (s^2, s^3), \dots, (s^{t-1}, s^t = s_{fin}) \rangle$ from the state s_{init} to the state s_{fin} such that the cost $C(a[s_{init}, s_{fin}]) \in [0, \infty)$ of this path is minimal over all possible paths between these states. Note that, in spite of the difference in formal definition, the meaning of the path $a[s_{init}, s_{fin}]$ is similar to the meaning of the trajectory $\overrightarrow{a}(t)$, $t = 0, 1, 2, \dots$, as defined for the search-planning problem within the search and screening framework (see Section 2.1.3).

In general, the cost $C(a[s_{init}, s_{fin}])$ is specified only for the paths $a[s_{init}, s_{fin}]$ that start in the initial state s_{init} and finish in the final state s_{fin} [75]. This cost $C(a[s_{init}, s_{fin}])$ is considered as a real cost of the resulting solution and is determined by an arbitrary function assigned to the vertexes and/or edges of the graph \mathcal{G} . For intermediate paths, which are obtained by actions of the GBF algorithms, the cost is defined by a *evaluation function* c that gives a cost $c(s) = c(a[s_{init}, s])$ of the path $a[s_{init}, s]$, $s \in S$, that starts in the initial state s_{init} and leads to the state $s \in S$. In the most general case, the equivalence between $c(s_{fin})$ and $C(a[s_{init}, s_{fin}])$ is not required, that is, it is assumed that $c(s_{fin}) \neq C(a[s_{init}, s_{fin}])$.

however, the monotonic dependence of the evaluation function c on the cost function C is imposed.

The GBF algorithm for solving the optimal path problem acts as follows [75, 79]. Starting from the initial state s_{init} , the algorithm selects a *successor* of the current state that gives a minimal cost. The cost includes both the cost of moving to the successor from the current state and previously calculated costs from the initial state to the current state. The search terminates at a state which has no available successors. If this state is s_{fin} , then the search was successful, otherwise the search failed. The GBF algorithm presents a general framework for finding an optimal path over the successors of the node by greedy methods. For specific applications, different types of cost functions are used, and corresponding to the cost functions, the GBF algorithms are divided into specific families of algorithms.

The best-known family of algorithms that implement best-first strategies are *best-first algorithms*, denoted BF* algorithms, which were suggested by Dechter and Pearl [75, 79], as a generalization of the A* algorithm [76] and similar algorithms of path planning over graphs. We start with BF* algorithms and then consider the A* algorithm [76] and their successors.

Let $\mathcal{G} = (S, E)$ be an oriented graph, and assume that for any path $a[s_{init}, s]$, $s_{init}, s \in S$, in the graph \mathcal{G} an evaluation cost function c is defined. For each state $s \in S$, denote by $c_0(a[s, s_{fin}])$ its initial cost. In the formulation of the BF* algorithm we use linked lists, namely, *OpenList* and *ClosedList*. For both lists, following C-style syntax, we define the following functions:

- function *include*(s) inserts state s into the list;
- function *exclude*(s) removes state s from the list.

That is, $aList.include(s)$ stands for insertion of the state s into $aList$ and $aList.exclude(s)$ stands for removing the state s from $aList$. Similarly, for the list *OpenList* we define:

- function *minimal_state*(), which returns a state of the list such that its cost is minimal, while ties are broken randomly.

In addition, for any state $s \in S$ we implement the following functions:

- function *expand*(s) results in the set of successors of the vertex s in the graph \mathcal{G} ;
- function *set_cost*(s, c) assigns a cost $c \in (-\infty, +\infty)$ to the state s ;
- function *get_cost*(s) returns a cost c that is assigned to the state s .

The BF* algorithm is outlined as follows. Since the evaluation function in the BF* algorithm is not formally defined, we consider this algorithm as a procedure.

Procedure 2.5 (BF* algorithm) [75, 79]. Given graph $\mathcal{G} = (S, E)$, states s_{init} and s_{fin} , $s_{init}, s_{fin} \in S$, cost function c , and initial costs $c_0(s)$ do:

1. Define a list *OpenList* and a list *ClosedList* and set both lists as empty.
2. Define a set *Neighborhood* and set *Neighborhood* = \emptyset .
3. Set $s = s_{init}$.

4. $\text{OpenList}.\text{include}(s)$: Include state $s = s_{\text{init}}$ in a list OpenList .
5. Do:
 - 5.1. If OpenList is empty then: Return and inform that a path from s_{init} to s_{fin} does not exist in the graph \mathcal{G} .
 - 5.2. Set $s = \text{OpenList}.\text{minimal_state}()$.
 - 5.3. $\text{OpenList}.\text{exclude}(s)$.
 - 5.4. $\text{ClosedList}.\text{include}(s)$:
 - 5.5. If $s = s_{\text{fin}}$ then Break: The final state s_{fin} is reached.
 - 5.6. Set $\text{Neighborhood} = \text{expand}(s)$: Include in Neighborhood all successors of the state s .
 - 5.7. For each $s_{\text{next}} \in \text{Neighborhood}$ do:
 - 5.7.1. Set $\text{assigned_cost} = \text{get_cost}(s_{\text{next}})$:
 - 5.7.2. Set $\text{new_cost} = c(s_{\text{next}})$: Calculate the new cost of a path from s_{init} to s_{next} .
 - 5.7.3. If s_{next} is neither in OpenList nor in ClosedList then:
 - a. $\text{set_cost}(s_{\text{next}}, \text{new_cost})$: Assign newly calculated cost to s_{next} .
 - b. $\text{OpenList}.\text{include}(s_{\text{next}})$.

Else: (i.e., the state s_{next} is either in OpenList or in ClosedList)

 - c. If $\text{new_cost} < \text{assigned_cost}$ then:
 $\text{set_cost}(s_{\text{next}}, \text{new_cost})$.
 - d. If s_{next} is in ClosedList then:
 $\text{ClosedList}.\text{exclude}(s_{\text{next}})$.
 $\text{OpenList}.\text{include}(s_{\text{next}})$.

While OpenList is not empty.

6. Return ClosedList . ■

Following the given graph $\mathcal{G} = (S, E)$, the BF* algorithm starts with the initial state s_{init} . At each step, the algorithm chooses a state $s \in S$ with minimal cost and considers the successors of this in the graph \mathcal{G} , which form a neighborhood of state s . For each state $s_{\text{next}} \in N(s)$ from the neighborhood, the algorithm calculates the cost of a path from the state s_{next} to the final state s_{fin} . For the next step, the algorithm chooses a state with minimal cost and continues by considering its neighborhood. Continuing the process, the algorithm results in a path from s_{init} to s_{fin} , if it exists, or informs about the non-existence of the required path with respect to the graph \mathcal{G} and cost function c .

Let us consider the general properties of Procedure 2.5 (BF* algorithm) as presented by Dechter and Perl [75, 79] with respect to finite graphs and under the assumption that there exists at least one required path. Let $\mathcal{G} = (S, E)$ be a finite graph, such that the sets S and E are finite, and let s_{init} and s_{fin} , $s_{\text{init}}, s_{\text{fin}} \in S$, be initial and final states, respectively. Since the graph \mathcal{G} is finite, there are a finite number of paths $a_j = a_j[s_{\text{init}}, s_{\text{fin}}]$, $j = 1, 2, \dots$,

from the state s_{init} to the state s_{fin} . For a state $s \in S$, we write $s \in a_j$ if path a_j passes the state s while going from s_{init} to s_{fin} . In addition, for any state $s \in a_j$, by $c(s|a_j)$ we denote the cost of following the path a_j from s_{init} to s .

The next statements as based on the *order-preserving property* of the evaluation cost function c are defined as follows. Let $a_1[s_{init}, s]$ and $a_2[s_{init}, s]$ be two paths from the initial state s_{init} to state s and let $a_3[s, s']$ be a path that starts at state s and continues up to some state s' . Then, the order-preserving property implies that [75, 79]

$$\begin{aligned} \text{if } c(a_1[s_{init}, s]) &\geq c(a_2[s_{init}, s]) \\ \text{then } c(a_1[s_{init}, s] \circ a_3[s, s']) &\geq c(a_2[s_{init}, s] \circ a_3[s, s']), \end{aligned} \quad (2.45)$$

where $a_i[x, s] \circ a_j[s, y]$ stands for the concatenation of paths a_i and a_j such that the resulting path follows the path a_i starting from state x up to state s , and then starting from s continues with the path a_j up to state y .

Assume that Procedure 2.5 implements a cost function c , for which the order-preserving property (Equation 2.45) holds. Then the following statements hold true.

Lemma 2.7 [75, 79]. *Assume that at the current step of Procedure 2.5 (BF* algorithm) the considered state is s . Let $a_1[s_{init}, s]$ be a path selected up to this step, and assume that $a_2[s_{init}, s]$ is a path to state s , which was considered in the previous steps and in which all states, except s , were already expanded. Then for the paths a_1 and a_2 it follows that*

$$c(s|a_1) \leq c(s|a_2).$$

In particular, if $a_1[s_{init}, s]$ is a path from s_{init} to s , which is a sub-path of the resulting path $a^[s_{init}, s_{fin}]$, that is, $a^*[s_{init}, s_{fin}] = a_1[s_{init}, s] \circ a_3[s, s_{fin}]$, where $a_3[s, s_{fin}]$ is a path that leads from s to s_{fin} , then the inequality*

$$c(s|a_1) \leq c(s|a)$$

holds for any path $a[s_{init}, s]$ from s_{init} to s .

Proof. The first inequality is an immediate result of Lines 5.2, 5.7.3c, and 5.7.3d. Following these lines, at each step Procedure 2.5 chooses a state s , which minimizes the cost $c(s)$, and its successor s_{next} , which also minimizes the cost $c(s_{next})$ of the path that leads to s_{next} . The successor with minimal cost is included in the actions of the procedure at the next steps. Since the order-preserving property of the cost function c guarantees that the states chosen at the next steps cannot increase the cost of the path already obtained, the inequality holds.

The second inequality is a direct consequence of the first inequality and order-preserving property of the cost function c . ■

Denote by $c_j = \max_{s \in a_j} \{c(s|a_j)\}$ a maximal cost of following the path a_j , and by $c^* = \min_j \{c_j\}$ a minimum of the maximal costs c_j over paths a_j , $j = 1, 2, \dots$.

Lemma 2.8 [75, 79]. *In Procedure 2.5 (BF* algorithm), at any step before termination OpenList includes a state s such that for some path $a[s_{init}, s_{fin}]$ it follows that $s \in a$ and $c(s|a) \leq c^*$.*

Proof. Assume that $a_k = a_k[s_{init}, s_{fin}]$ is a path for which the min–max of the cost is reached, that is, $c^* = c_k$. Let $s \in a_k$ be a state from the path a_k such that at some step before termination of Procedure 2.5 it is a minimal state in $OpenList$, while the other states from $OpenList$ are included in the path $a_j = a_j[s_{init}, s_{fin}]$, possibly equal to a_k .

Since $c_k = \max_{s \in a_k} \{c(s|a_k)\}$, it follows that $c(s|a_k) \leq c_k = c^*$. Moreover, since all states in $OpenList$, except s , are from the path a_j , all states from the path a_k , except s , are in $ClosedList$. Hence, according to Lemma 2.7, it follows that $c(s|a_j) \leq c(s|a_k)$, and this implies $c(s|a_j) \leq c^*$, as required. ■

Now let us demonstrate that if Procedure 2.5 (BF* algorithm) terminates, then it results in an optimal path in the sense of min–max cost.

Theorem 2.8 [79]. *Let $a_j = a_j[s_{init}, s_{fin}]$ be a path, with which Procedure 2.5 (BF* algorithm) terminates. Then $c^* = c_j$.*

Proof. In contradiction, assume that $c_j > c^*$, and let $s^* \in a_j$ be a state from the path a_j that provides $c(s^*|a_j) = c_j$. Since Procedure 2.5 terminates with the path a_j , state s^* is chosen at least once for expansion, and at the last step, where an expansion held, state s^* obtains the cost $c(s^*|a_j) > c^*$. However, according to Lemma 2.8, at that step $OpenList$ contains a state s such that $c(s) \leq c^*$, and so $c(s) < c^*$. Hence, s would be expanded before c^* , which contradicts the assumption. ■

Finally, let us demonstrate that, being applied over a finite graph \mathcal{G} , the BF* algorithm (Procedure 2.5) terminates.

Theorem 2.9 [79]. *If there exists a path from the initial state s_{init} and the final state s_{fin} , then Procedure 2.5 (BF* algorithm) terminates with a resulting path.*

Proof. Since graph \mathcal{G} is finite, each path between the initial state s_{init} and final state s_{fin} has a finite length. Assume, in contradiction, that Procedure 2.5 does not terminate. Then, when adding the states to the chosen paths, the evaluated cost c increases according to the order-preserving property (Equation 2.45). Thus, after a certain step there are no states in $OpenList$ such that they have to be expanded. However, according to Lemma 2.8, $OpenList$ includes a state s from the resulting path such that $c(s) \leq c^*$, and it has to be expanded. This contradiction provides termination of the procedure. ■

Note that the assumption that the graph \mathcal{G} is finite is not necessary and the presented statements still hold true for a locally finite graph, in which a set of paths between the initial and final states is infinite but countable [75, 79].

Procedure 2.5 (BF* algorithm) was suggested by Dechter and Pearl as a general template for several algorithms of path planning that differ in their implemented evaluation functions c . To illustrate the actions of Procedure 2.5, let us specify the values of the evaluated costs and define the cumulative evaluation cost function. An illustration is provided in the next example.

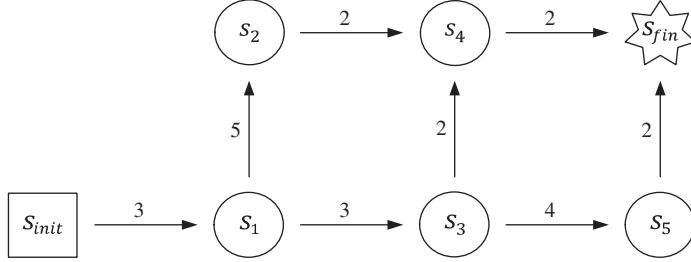


Figure 2.29 Example of the graph of states for the BF* algorithm.

Example 2.20 Let the set S of states includes seven states $S = \{s_{init}, s_1, s_2, s_3, s_4, s_5, s_{fin}\}$ which are connected by the weighted edges $e = (s_j, s_k) \in E$ so that they form an oriented graph $\mathcal{G} = (S, E)$, which is shown in Figure 2.29.

According to the goal of the BF* algorithm (Procedure 2.5), the searcher starts in state s_{init} and has to arrive at state s_{fin} following the path $a[s_{init}, s_{fin}]$ with minimal cost $C(a[s_{init}, s_{fin}])$. Assume that the evaluated costs $c(a[s_{init}, s])$ of the paths $a[s_{init}, s]$ in the graph \mathcal{G} and corresponding evaluated costs $c(s)$ of the states are cumulative. Then Procedure 2.5 acts as follows.

The searcher starts with the state $s^{t=0} = s_{init}$. Expanding this state, the searcher obtains its single successor $s^{t=1} = s_1$, while the evaluated cost $c(s_1)$ of state s_1 , for the cumulative evaluation function, is specified by the weight of the edge (s_{init}, s_1) summed with the weights of the edges of a minimal path from s_1 to s_{fin} , that is, $c(s_1) = c(a[s_{init}, s_1]) = w(s_{init}, s_1) = 3$.

Since there is a single successor s_1 of the initial state s_{init} , the searcher moves to state s_1 . Then, expanding state s_1 , the searcher obtains the set $N(s_1)$ and its successors, $N(s_1) = \{s_2, s_3\}$. The evaluated costs of states s_2 and s_3 are as follows:

$$c(s_2) = c(a[s_{init}, s_2]) = c(a[s_{init}, s_1]) + w(s_1, s_2) = 3 + 5 = 8.$$

$$c(s_3) = c(a[s_{init}, s_3]) = c(a[s_{init}, s_1]) + w(s_1, s_3) = 3 + 3 = 6.$$

According to Procedure 2.5, the searcher chooses a state with minimal evaluated cost. Hence, the searcher chooses the state $s^{t=2} = s_3$. Expanding this state, the searcher obtains its successors: $N(s_3) = \{s_4, s_5\}$. The costs of the states s_4 and s_5 are as follows:

$$c(s_4) = c(a[s_{init}, s_4]) = c(a[s_{init}, s_3]) + w(s_3, s_4) = 6 + 2 = 8.$$

$$c(s_5) = c(a[s_{init}, s_5]) = c(a[s_{init}, s_3]) + w(s_3, s_5) = 6 + 4 = 10.$$

Hence, following the minimal evaluated cost, the chosen state is $s^{t=3} = s_4$.

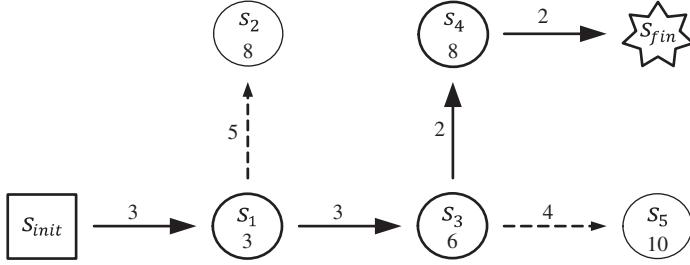
The state s_4 has a single successor that is the final state s_{fin} . Thus, the searcher chooses the states s_{fin} and procedure terminates. The evaluated cost for this state is

$$c(s_{fin}) = c(a[s_{init}, s_{fin}]) = c(a[s_{init}, s_4]) + w(s_4, s_{fin}) = 8 + 2 = 10.$$

The resulting path

$$a[s_{init}, s_{fin}] = \langle (s_{init}, s_1), (s_1, s_3), (s_3, s_4), (s_4, s_{fin}) \rangle$$

with the expanded states and their evaluated costs are shown in Figure 2.30.



$$a[s_{init}, s_{fin}] = \{(s_{init}, s_1), (s_1, s_3), (s_3, s_4), (s_4, s_{fin})\}; C(a[s_{init}, s_{fin}]) = c(s_{fin}) = 10.$$

Figure 2.30 Example of the resulting path obtained by the BF* algorithm with a cumulative evaluation cost function.

According to the assumption regarding the cost, which is a sum of the weights of the edges, the cost $C(a[s_{init}, s_{fin}])$ of the obtained path is equal to the resulting evaluated cost and is minimal. ■

In this example, a cumulative evaluation cost function was implemented so that the cost $C(a[s_{init}, s_{fin}])$ and evaluated cost $c(a[s_{init}, s_{fin}])$ for the resulting path are equivalent. Within the framework of routing problems [80] the evaluation function is defined in a different manner by using distance estimations. Below we consider such a function for the evaluated costs, while similar functions will be considered later in terms of information theory.

As indicated above, the BF* algorithm (Procedure 2.5) was constructed by Dechter and Pearl [75, 79] as a general template for several algorithms of path planning. In particular, it generalizes the Dejkstra algorithm [70], which was suggested in 1959, and the A* algorithm, which was constructed by Hart *et al.* [76] in 1968. Since our further considerations will be based on the A* algorithm, let us consider it in particular.

The A* algorithm acts as follows [76]. Let $\mathcal{G} = (S, E)$ be a finite graph, where S is a set vertexes (or nodes), which correspond to the states, and $E \subset S \times S$ is a set of edges. As above, let $s_{init} \in S$ be the initial state and $s_{fin} \in S$ be the final state. According to the general BF* algorithm, assume that for each state $s \in S$ a finite value $c(s) \in [0, \infty)$ of the evaluation cost function c is defined. For certain implementations of the function c and the values of the costs, we will write c^* , \bar{c}^* , \tilde{c}^* , and similar, assuming that in all cases the same function c is applied. In addition, and in contrast to the general BF* algorithm, in the A* algorithm cost function C , which provides real costs $C(a[s_{init}, s_{fin}])$, and evaluation cost function c are not distinguished.

In the A* algorithm, the values of the *actual evaluation cost function* c^* , $c^*(s) \in [0, \infty)$, over the states $s \in S$ are specified as follows [76]:

$$c^*(s) = \bar{c}^*(a^*[s_{init}, s]) + \tilde{c}^*(a^*[s, s_{fin}]) = \bar{c}^*(s) + \tilde{c}^*(s), \quad (2.46)$$

where $\bar{c}^*(s) = \bar{c}^*(a^*[s_{init}, s]) = c(s|a^*)$ is the *actual evaluated cost of optimal path* $a^*[s_{init}, s]$ from the initial state s_{init} to state s , and $\tilde{c}^*(s) = \tilde{c}^*(a^*[s, s_{fin}])$ is the *actual cost of optimal path* $a^*[s, s_{fin}]$ from state s to the final state s_{fin} .

Starting from the actual cost c^* that characterizes an optimal path, the values of the *evaluation cost function* c over the states $s \in S$ are defined by the following sum [76]:

$$c(s) = \bar{c}(a[s_{init}, s]) + \bar{c}(a^*[s, s_{fin}]) = \bar{c}(s) + \tilde{c}(s), \quad (2.47)$$

where $\bar{c}(s) = \bar{c}(a[s_{init}, s])$ is the evaluated cost of the path from the initial state s_{init} to state s as is found up to the current step of the algorithm, and $\tilde{c}(s) = \tilde{c}(a^*[s, s_{fin}])$ is the *estimated evaluated cost of the optimal path* from state s to the final state s_{fin} .

Note that since the values of the evaluated costs are non-negative, the cost function, which is defined by Equations 2.46 and 2.47, meets the order-preserving property (Equation 2.45). In addition, since a path $a[s, s_{fin}]$ is not necessarily optimal, for the first addendums in Equations 2.46 and 2.47 and each state $s \in S$ it follows that $\bar{c}(s) \geq \bar{c}^*(s)$. However, regarding the second addendums $\tilde{c}(s)$ and $\tilde{c}^*(s)$, some similar relation does not hold automatically and has to be assumed. After formulation of the A* algorithm we will consider the consequences of such an assumption.

Let us formulate the A* algorithm on the basis of the introduced costs. Since the algorithm follows the lines of the general BF* algorithm (Procedure 2.5) while applying a specific function for calculating cost $c(s_{next})$ for the next state s_{next} (see Line 5.7.2), it inherits the main properties of the BF* algorithm. Below we present a brief outline of the A* algorithm, which will be used for reference in the proofs of its properties.

Algorithm 2.11 (A* algorithm) [75, 76, 81]. Given graph $\mathcal{G} = (S, E)$, states s_{init} and s_{fin} , $s_{init}, s_{fin} \in S$, cost function c defined by Equation 2.47, and initial costs $c_0(s)$ do:

1. Start with the initial state s_{init} , mark it as *open* and calculate its cost $c(s_{init})$, and at each step conduct the following actions:
2. If there are no *open* states, then terminate with failure.
3. Select an *open* state s such that its evaluation cost $c(s)$ is minimal over all *open* states. Ties are broken randomly in favor of s_{fin} . Mark selected state s as *closed*.
4. If selected state s is equal to the final state s_{fin} , then terminate with the list of the states that are marked as *closed*.
5. Otherwise, expand s and obtain successors s_{next} of the state s in the graph \mathcal{G} . Mark as *open* all successors of s , which are not marked as *closed*.
6. For each successor s_{next} do:

- 6.1. Estimate $\tilde{c}(s_{next})$ and calculate the cost

$$\bar{c}(s_{next}) = \bar{c}(s) + c(a[s, s_{next}]), \bar{c}(s_{init}) = 0$$

where $\bar{c}(s_{next}) = \bar{c}(s) + c(a[s, s_{next}])$, $\bar{c}(s_{init}) = 0$ and $c(a[s, s_{next}])$ stands for the cost of a path from state s to its successor s_{next} .

- 6.2. If the calculated cost $c(s_{next})$ of the successor s_{next} is less than its previous cost (i.e., $c_0(s_{next})$ at the first trial) and if s_{next} is not marked as *open*, then mark s_{next} as *open*.

7. Continue with Line 2.



Since the A* algorithm (Algorithm 2.11) follows the lines of the BF* algorithm (Procedure 2.5) and the implemented evaluation cost function (Equation 2.47) meets the order-preserving requirement (Equation 2.45), then for the A* algorithm Theorems 2.8 and 2.9 hold. Now let us consider the above assumption regarding the estimated cost $\tilde{c}(s)$ and its relation with the cost $c^*(s)$. In the formulations below we follow the original paper by Hart *et al.* [76] and the later presentation by Nilsson [81]; for other proofs of the optimality of the A* algorithm see [82].

Lemma 2.9 [76, 81]. *If for all states $s \in S$ it follows that $\tilde{c}(s) \leq c^*(s)$, then at any step before termination of the A* algorithm (Algorithm 2.11) and any optimal path $a^*[s_{init}, s_{fin}]$, there exists an open state $s' \in a^*$ such that $c(s') \leq c^*(s')$*

Proof. Since at the considered step the A* algorithm was not terminated, optimal path a^* includes at least one *open* state. Let $s' \in a^*[s_{init}, s_{fin}]$ be the first *open* state in the path $a^*[s_{init}, s_{fin}]$. According to the definition in Equation 2.47, the evaluated cost of the state s' is

$$c(s') = \bar{c}(a[s_{init}, s']) + \tilde{c}(a[s', s_{fin}]) = \bar{c}(s') + \tilde{c}(s').$$

Since s' is in the optimal path a^* and all states in $a^*[s_{init}, s']$ are already marked as *closed*, the evaluated cost $\bar{c}(a[s_{init}, s'])$ of the path $a[s_{init}, s']$ is equal to the actual cost $c^*(a^*[s_{init}, s'])$ of the optimal path $a^*[s_{init}, s']$. Hence, $c(s') = \bar{c}(s') + \tilde{c}(s') = c^*(s') + \tilde{c}(s')$.

According to Equation 2.46, the actual cost of the optimal path that includes the state s' is $c^*(s') = \bar{c}^*(s') + \tilde{c}^*(s')$. The first addends in the definitions of the costs $c(s')$ and $c^*(s')$ are equal, while, according to the assumption in the lemma, for the second addends it holds true that $\tilde{c}(s) \leq \tilde{c}^*(s)$. Thus $c(s') \leq c^*(s')$, as required. ■

Now let us consider termination of the A* algorithm (Algorithm 2.11) and show that it is optimal; that is, in being applied to the graph $\mathcal{G} = (S, E)$, the A* algorithm results in a path from the initial state s_{init} to the final state s_{fin} , if it exists, with minimal cost.

Recall that in the A* algorithm $c(a[s, s_{next}]) = c(e)$, $e \in E$, denotes the evaluated cost of a movement from state s to its successor s_{next} in the graph \mathcal{G} .

Theorem 2.10 [76, 81]. *Assume that for all states $s \in S$ it follows that $\tilde{c}(s) \leq c^*(s)$. If $c(a[s, s_{next}]) \geq \delta > 0$, then the A* algorithm (Algorithm 2.11) terminates and results in the path $a^*[s_{init}, s_{fin}]$, if it exists, with minimal cost.*

Proof. The proof of the theorem follows from the following three claims.

Claim 1. *If Algorithm 2.11 terminates with a result, then at termination it finds the final state s_{fin} .*

Proof. The algorithm terminates either in Line 4 on finding the final state s_{fin} or in Line 2 when there are no states that are marked as *open*. However, according to Lemma 2.9, if a path $a[s_{init}, s_{fin}]$ from the state s_{init} to the state s_{fin} exists, before termination there is a state that is marked as *open*. Thus, if there is a path $a[s_{init}, s_{fin}]$, then the algorithm terminates in Line 2 on finding s_{fin} .

Claim 2. *Algorithm 2.11 terminates, with a result or failure, in a finite number of steps.*

Proof. Assume that the final state s_{fin} can be reached from the initial state s_{init} in a finite number of steps. We need to demonstrate that, in such a case, the algorithm terminates.

Let $c^*(s_{fin}) = \bar{c}^*(a^*[s_{init}, s_{fin}]) + \tilde{c}^*(a[s_{fin}, s_{fin}]) = \bar{c}^*(s_{fin})$ be the cost of an optimal path from the state s_{init} to the state s_{fin} . Since $c(s) = \bar{c}(s) + \tilde{c}(s)$ and all costs are non-negative, $c(s) \geq \bar{c}(s)$. In addition, recall that $\bar{c}(s) \geq \bar{c}^*(s)$ for any state $s \in S$. According to the assumption, $c(a[s, s_{next}]) \geq \delta > 0$ for any edge $(s, s_{next}) \in E$. Then, for the cost of any state s' that is located farther than $l = c^*(s_{fin})/\delta$ steps from the initial state s_{init} , it holds true that $c(s') \geq \bar{c}(s') \geq \bar{c}^*(s') > l\delta = c^*(s_{fin})$.

From Lemma 2.4 it follows that state s' , which is located farther than l steps from the state s_{init} , will not be expanded. In fact, according to the lemma, the optimal path includes an *open* state s'' such that $c(s'') \leq c^*(s_{fin}) < c(s')$, which will be selected instead of s' in Line 3. Thus, the failure of algorithm termination can be caused by re-marking the *closed* states, which are reachable from s_{init} in l steps, as *open* at Line 6.2.

Let $S(l) \subset S$ be a set of states that are reachable from the state s_{init} within l steps and let $|S(l)|$ stand for a number of such states. Since there are a finite number of paths from s_{init} to any state $s \in S(l)$, each state $s \in S(l)$ can be re-marked as *open* at most a finite number of times $k(s, l)$. Let $k = \max_{s \in S(l)} \{k(s, l)\}$ be a maximum number of times that the states $s \in S(l)$ were re-marked as *open*. Then, after at most $k \cdot |S(l)|$ expansions, all states in $S(l)$ will be marked as *closed* and the algorithm will terminate.

Claim 3. *If Algorithm 2.11 terminates with a result, then the resulting path has minimal cost.*

Proof. Assume that the algorithm terminates with the resulting path from the state s_{init} to some state s' , and assume, in contradiction, that the cost of this path is not minimal, that is, $c(s') = \bar{c}(s') > c^*(s')$. However, according to Lemma 2.9, before termination in the optimal path there was an *open* state s such that $c(s) \leq c^*(s') < c(s')$. Thus, at this step before termination, state s would be selected instead of state s' , which contradicts the assumption that the algorithm terminated with s' .

The combination of the proven claims results in the statement of the theorem. ■

The A* algorithm (Algorithm 2.11) acts similarly to the BF* algorithm (Procedure 2.5) considered above. However, to stress the implementation of estimated costs, let us consider the following example.

Example 2.21 Assume that the graph $\mathcal{G} = (S, E)$ of the states and their connections is the same as in Example 2.20. In addition, assume that for each state $s \in S$ an estimated cost $\tilde{c}(s)$ is defined. The graph $\mathcal{G} = (S, E)$ with actual costs $c(a[s, s_{next}])$ of the edges, which are equal to the weights $w(s, s_{next})$ of the edges, and estimated costs $\tilde{c}(s)$ is shown in Figure 2.31. In the figure, the dashed lines with values represent estimated costs, and solid arrows with values represent actual costs of the edges $c(a[s, s_{next}]) = w(s, s_{next})$.

Over the presented graph with estimated costs, the A* algorithm (Algorithm 2.11) acts similarly to the BF* algorithm, and its actions are as follows.

The searcher starts with the initial state $s^{t=0} = s_{init}$ and, expanding it, obtains a single successor $s^{t=1} = s_1$. The evaluated cost $c(s_1)$ of state s_1 is calculated as follows:

$$c(s_1) = \bar{c}(s_{init}) + c(a[s_{init}, s_1]) + \tilde{c}(s_{next}) = 0 + 3 + 6 = 9,$$

where we implied that $\bar{c}(s_{init}) = 0$.

Since s_1 is a single successor of the initial state, the searcher moves to state s_1 . Expansion of state s_1 results in the set $N(s_1) = \{s_2, s_3\}$ of its successors, for which the evaluated costs

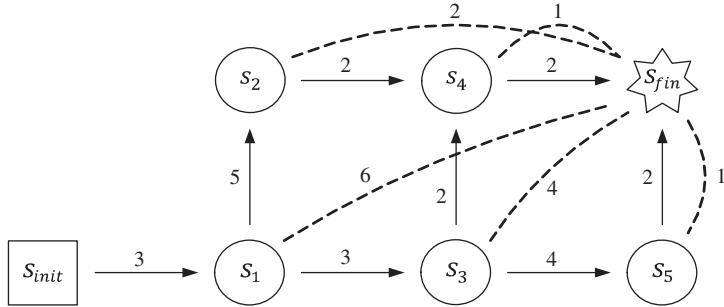


Figure 2.31 Example of the graph of states with estimated costs for the A* algorithm.

are as follows:

$$c(s_2) = \bar{c}(s_1) + c(a[s_1, s_2]) + \tilde{c}(s_2) = 3 + 5 + 2 = 10,$$

$$c(s_3) = \bar{c}(s_1) + c(a[s_1, s_3]) + \tilde{c}(s_3) = 3 + 5 + 2 = 10.$$

Since the obtained evaluated costs are equal, the next state is chosen randomly from s_2 and s_3 . Assume that the chosen state is s_2 . Expansion of this state results in its single successor s_4 . The evaluated cost $c(s_4|a[s_{init}, s_2]) \circ a[s_2, s_4]$ of state s_4 reached from state s_2 is

$$c(s_4|a[s_{init}, s_2]) \circ a[s_2, s_4] = \bar{c}(s_2) + c(a[s_2, s_4]) + \tilde{c}(s_4) = 8 + 2 + 1 = 11.$$

The obtained cost $c(s_4)$ is greater than the previously obtained costs $c(s_2)$ and $c(s_3)$. Hence, at the next step the algorithm returns to the successors $N(s_1) = \{s_2, s_3\}$ of state s_1 and instead of s_2 chooses the not yet expanded state s_3 . Expansion of state s_3 results in the set of successors $N(s_3) = \{s_4, s_5\}$. The evaluated costs $c(s_4|a[s_{init}, s_3]) \circ a[s_3, s_4]$ of state s_4 reached from state s_3 and state s_5 are as follows:

$$c(s_4|a[s_{init}, s_3]) \circ a[s_3, s_4] = \bar{c}(s_3) + c(a[s_3, s_4]) + \tilde{c}(s_4) = 6 + 2 + 1 = 9,$$

$$c(s_5) = \bar{c}(s_3) + c(a[s_3, s_5]) + \tilde{c}(s_5) = 6 + 4 + 1 = 11.$$

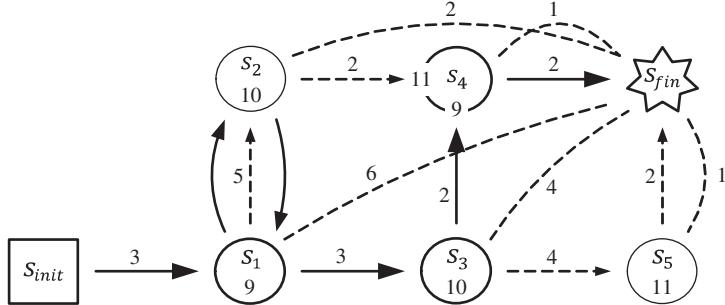
State s_4 can be reached from state s_3 with minimal evaluated cost $c(s_4|a[s_{init}, s_3]) \circ a[s_3, s_4]$. Thus from the state $s^{t=1} = s_1$ the searcher moves to its successor $s^{t=2} = s_3$.

The set of successors of state s_3 obtained already is $N(s_3) = \{s_4, s_5\}$, and for both states expansion results in the final state s_{fin} . The evaluated costs of the final state s_{fin} reached via states s_4 and s_5 are as follows:

$$c(s_{fin}|a[s_{init}, s_4] \circ a[s_4, s_{fin}]) = \bar{c}(s_4) + c(a[s_4, s_{fin}]) + \tilde{c}(s_{fin}) = 8 + 2 + 0 = 10,$$

$$c(s_{fin}|a[s_{init}, s_5] \circ a[s_5, s_{fin}]) = \bar{c}(s_5) + c(a[s_5, s_{fin}]) + \tilde{c}(s_{fin}) = 10 + 2 + 0 = 12,$$

where we implied that $\tilde{c}(s_{fin}) = 0$. Since a minimal evaluated cost is obtained for state s_4 , the searcher moves to this state, $s^{t=3} = s_4$.



$$a[s_{init}, s_{fin}] = \langle (s_{init}, s_1), (s_1, s_3), (s_3, s_4), (s_4, s_{fin}) \rangle; c(s_{fin}) = 10.$$

Figure 2.32 Example of the expanded states and the resulting path obtained by the A^* algorithm.

Further expansion of state s_4 results in the final state s_{fin} . Thus, the searcher chooses the state s_{fin} and the procedure terminates. The evaluated cost $c(s_{fin})$ of the final state is equal to the previously calculated cost $c(s_{fin}|a[s_{init}, s_4] \circ a[s_4, s_{fin}])$, so $c(s_{fin}) = 10$ and it is a minimal cost $c^*(s_{fin})$ of the path $a[s_{init}, s_{fin}]$. The resulting path with the expanded states and their evaluated costs is shown in Figure 2.32.

The path $a[s_{init}, s_{fin}]$, which is obtained by the A^* algorithm, as expected, is similar to the path obtained by the BF^* algorithm. ■

Let us return to the definitions of the costs (Equations 2.46 and 2.47), in which *actual cost* $\tilde{c}^*(s) = \tilde{c}^*(a^*[s, s_{fin}])$ and *estimated cost* $\tilde{c}(s) = \tilde{c}(a^*[s, s_{fin}])$ are implied. As indicated in the formulations of Lemma 2.9 and Theorem 2.10, it is assumed that for the actual cost $\tilde{c}^*(s)$ and estimated cost $\tilde{c}(s)$ of any state $s \in S$ it follows that

$$\tilde{c}(s) \leq \tilde{c}^*(s). \quad (2.48)$$

Since this assumption provides termination and optimality of the A^* algorithm – in the other words, its admissibility – the assumption is widely known as the *admissibility assumption*.

An additional assumption that is useful for certain applications of the A^* algorithm is called the *consistency assumption* and is formulated as follows [76, 81]. Let $c(a[s, s_{next}]) = c(e)$, $e \in E$, where s_{next} is a successor of state s in the graph $G = (S, E)$, be the evaluated cost of an edge. Then, the consistency assumption requires that for all states $s, s_{next} \in S$, for which the cost $c(a[s, s_{next}])$ is defined, it follows that

$$\tilde{c}(s) - \tilde{c}(s_{next}) \leq c(a[s, s_{next}]), \quad (2.49)$$

while the estimated cost function \tilde{c} , which meets the consistency assumption (Equation 2.49), is called *consistent*. By using the full form of the estimated costs, this assumption can be rewritten as $\tilde{c}(a^*[s, s_{fin}]) \leq c(a[s, s_{next}]) + \tilde{c}(a^*[s, s_{next}])$, which shows that it represents a certain kind of triangle inequality regarding the costs. Later, we will use such a property for information theory algorithms of search based on the A^* algorithm.

Let us demonstrate that implementation of consistent estimated cost function \tilde{c} can improve the actions of the A* algorithm. Proofs of the statements that are given below are rather technical. However, in addition to their importance to the A* algorithm, they provide an intuition for considering the informational search algorithms presented in the next sections.

Lemma 2.10 [76, 81]. *If in Algorithm 2.11 (A* algorithm) the estimated cost function \tilde{c} is consistent, then for every closed state $s \in S$ it follows that $\bar{c}(s) = \tilde{c}^*(s)$. In other words, if the A* algorithm expands a state, then an optimal path to this state is found.*

Proof. Let us consider the moment before marking the state s as *closed*, and assume, on the contrary, that $\bar{c}(s) > \tilde{c}^*(s)$. Since, as in the definitions (Equations 2.46 and 2.47) of the evaluated costs, for any state $s \in S$ it follows that $\bar{c}(s) \geq \tilde{c}^*(s)$, the indicated inequality is the only contrary case to the equivalence $\bar{c}(s) = \tilde{c}^*(s)$. In spite of the existence of the optimal path $a^*[s_{init}, s]$, since $\bar{c}(s) > \tilde{c}^*(s)$ it is not found. However, according to Lemma 2.9, there exists an *open* state $s' \in a^*[s_{init}, s]$ such that $\bar{c}(s') = \tilde{c}^*(s')$. If this state s' is exactly the considered state s , then the contradiction obtained in the lemma is proven.

Assume that $s' \neq s$. Then $\bar{c}(s) = \bar{c}(s') + c(a[s, s']) = \tilde{c}^*(s') + c(a[s, s'])$. Thus, from the assumption $\bar{c}(s) > \tilde{c}^*(s)$ it follows that $\bar{c}(s) > \bar{c}(s') + c(a[s, s'])$. Addition of $\tilde{c}(s)$ to both sides of the last inequality gives $\bar{c}(s) + \tilde{c}(s) > \bar{c}(s') + c(a[s, s']) + \tilde{c}(s)$. By applying the consistency assumption (Equation 2.49) to the right side of the inequality, one obtains that $\bar{c}(s) + \tilde{c}(s) > \bar{c}(s') + \tilde{c}(s)$, and finally $c(s) > c(s')$. However, since state s was selected for expansion while state s' was available at less cost, the obtained inequality cannot hold. ■

The next lemma is a direct consequence of Lemma 2.9.

Lemma 2.11 [76, 81]. *If in Algorithm 2.11 (A* algorithm) the estimated cost function \tilde{c} is consistent and for any closed state $s \in S$ it follows that $\tilde{c}(s) \leq \tilde{c}^*(s)$, then $c(s) \leq c^*(s_{fin})$.*

Proof. Let s be a *closed* state. If $s = s_{fin}$, then $c(s) = c^*(s_{fin})$ and the lemma holds. Assume that $s \neq s_{fin}$. Then, state s was marked as *closed* before termination. According to Lemma 2.9, in the optimal path $a^*[s_{init}, s_{fin}]$ there exists an *open* state $s' \in a^*[s_{init}, s_{fin}]$ such that $c(s') \leq c^*(s_{fin})$. If $s = s'$ and it was expanded and marked as *closed*, then $c(s) = c(s') \leq c^*(s_{fin})$ and the lemma holds. Let $s \neq s'$. Since s' is an *open* state, state s was chosen and expanded before s' . Thus, $c(s) \leq c(s') \leq c^*(s_{fin})$ as required. ■

Finally, let us consider the relation between estimated costs $\tilde{c}(s)$ and the efficiency of the A* algorithm. Let **A** and **B** be two A* algorithms, that is, both **A** and **B** follow the lines of Algorithm 2.11 and differ in the methods of cost calculations. Denote by \tilde{c}_A and \tilde{c}_B the estimated cost functions, which are implemented in algorithms **A** and **B**, correspondingly. Assume that both **A** and **B** are admissible, which implies that for all states $s \in S$ it follows that $\tilde{c}_A(s) \leq \tilde{c}^*(s)$ and $\tilde{c}_B(s) \leq \tilde{c}^*(s)$. Then it can be said that algorithm **A** is *more informed* than algorithm **B** if for all states $s \in S$ it follows that $\tilde{c}_A(s) > \tilde{c}_B(s)$. The next theorem states that a more informed A* algorithm with consistent estimated costs outperforms a less informed A* algorithm with inconsistent expected costs.

Theorem 2.11 [81]. Let \mathbf{A} and \mathbf{B} be two admissible A* algorithms and let \mathbf{A} be more informed than \mathbf{B} . Assume that the estimated costs function $\tilde{c}_{\mathbf{A}}$, which is implemented in \mathbf{A} , meets the consistency assumption (Equation 2.49). Then if a state $s \in S$ is expanded by algorithm \mathbf{A} , it is also expanded by algorithm \mathbf{B} .

Proof. Assume, on the contrary, that there exists a state $s \in S$ such that it was expanded by \mathbf{A} but not expanded by \mathbf{B} . Since state s was expanded by \mathbf{A} , it is not a final state.

On the one hand, according to the assumption \mathbf{B} is admissible, that is, it results in the path with minimal evaluated cost. Thus, if \mathbf{B} does not expand s , then, because of its admissibility, $c^*(s) \geq c^*(s_{fin})$ has to hold true. Since $c^*(s) = \bar{c}^*(s) + \tilde{c}^*(s)$, and so $\tilde{c}^*(s) = c^*(s) - \bar{c}^*(s)$, the obtained inequality implies $\tilde{c}^*(s) \geq c^*(s_{fin}) - \bar{c}^*(s)$. Thus, the lower bound of $\tilde{c}^*(s)$ that is available to \mathbf{B} is $\tilde{c}_{\mathbf{B}}(s) = c^*(s_{fin}) - \bar{c}^*(s)$.

On the other hand, algorithm \mathbf{A} implements the evaluation cost function $c_{\mathbf{A}}(s) = \bar{c}(s) + \tilde{c}_{\mathbf{A}}(s)$. From the admissibility of \mathbf{A} and Lemma 2.11 it follows that $c_{\mathbf{A}}(s) \leq c^*(s_{fin})$. Thus $\bar{c}(s) + \tilde{c}_{\mathbf{A}}(s) \leq c^*(s_{fin})$ and so $\tilde{c}_{\mathbf{A}}(s) \leq c^*(s_{fin}) - \bar{c}(s)$. By Lemma 2.10, from the consistency of the costs in \mathbf{A} it follows that, while \mathbf{A} expands state s , it follows that $\bar{c}(s) = \bar{c}^*(s)$. Thus $\tilde{c}_{\mathbf{A}}(s) \leq c^*(s_{fin}) - \bar{c}^*(s)$.

Comparing the results regarding estimated costs in \mathbf{A} and \mathbf{B} , one obtains that $\tilde{c}_{\mathbf{A}}(s) \leq \tilde{c}_{\mathbf{B}}(s)$ in contradiction to the assumption that \mathbf{A} is more informed than \mathbf{B} . ■

The BF* algorithm (Procedure 2.5) and A* algorithm (Algorithm 2.11) are the best-known methods of path planning and search for a static target over graphs that implement the best-first strategies. For additional information regarding such strategies and algorithms and a general framework, see the book by Pearl [75], while for certain examples of applications of the algorithms and the recursive form of the A* algorithm, see [80]. For further analysis of the BF* and A* algorithms and comparison of their successors, see [83] and [84].

According to the implemented dynamic programming scheme in the BF* and A* algorithms, they require consideration of the complete list of *open* states and exchange between the lists of *open* and *closed* states. Such a technique results in the optimal path, if it exists, from the initial state s_{init} to the final state s_{fin} , and allows the algorithms to be applied for a static target search. However, for real-time path planning and MTS these algorithms are not applicable. Below, we consider the A*-type algorithms that can be implemented for real-time search and path planning and that provide near-optimal solutions.

2.3.2 Real-time search and learning real-time A* algorithm

An algorithm of path planning and search for a static target, which is based on the A* algorithm and returns the solution in real time during the search, is known as the Learning Real-Time A* (LRTA*) algorithm. This algorithm was constructed by Korf [85–87] in 1987–1990, and follows a line of Real-Time A* (RTA*) algorithms initiated by Korf [88] in 1985. Similar to the generic A* algorithm, the RTA* algorithm is performed over a finite graph $\mathcal{G} = (S, E)$ and applies a similar admissibility property (Equation 2.48) of the estimated costs $\tilde{c}(s)$, $s \in S$. However, the definition of the evaluated cost $c(s)$ in the RTA* algorithm differs from the definition in Equation 2.47, which is implemented in the A* algorithm. Before formulating the RTA* algorithm, let us stress this difference.

Let $\mathcal{G} = (S, E)$ be a finite but not necessarily directed graph, and let s_{init} be the initial state and s_{fin} be the final state, $s_{init}, s_{fin} \in S$. Assume, in addition, that from any state $s \in S$, including initial state s_{init} , there exists a path $a[s, s_{fin}]$ from this state to the final state.

Recall that according to the A* algorithm (Algorithm 2.11) the evaluation cost function c is implemented so that for any state $s \in S$ its evaluation cost $c(s)$ is defined as a sum

$$c(s) = \bar{c}(a[s_{init}, s]) + \tilde{c}(a^*[s, s_{fin}])$$

of the cost $\bar{c}(a[s_{init}, s])$ of the path already found from the initial state s_{init} to the current state s and the estimated cost $\tilde{c}(a^*[s, s_{fin}])$ of the optimal path from the current state s to the final state s_{fin} . Then, if at the current time the searcher considers the current state s_{curr} , then the next state s_{next} is chosen from the set $N(s_{curr})$ of *open* successors of the current state s_{curr} . The successors are marked as *open* if their newly calculated evaluation costs are less than the previously obtained evaluation costs, and the next state s_{next} is chosen so that its evaluated cost reaches its minimum over the *open* successors of the current state s_{curr} , that is,

$$s_{next} = \operatorname{argmin}_{s \in N(s_{curr}) \text{ and } s \text{ is open}} \{c(s)\}.$$

Notice again that the cost $c(s)$ of state s is calculated relative to the initial state s_{init} taking into account the estimated cost to the final state s_{fin} .

In contrast, in the RTA* algorithm [87] the evaluation cost $c_{RT}(s)$ of the state $s \in S$ is calculated relative to the current state s_{curr} rather than to the initial state. In other words, if at the current time the searcher is in the current state s_{curr} , then the evaluation cost $c_{RT}(s)$ of the state $s \in S$ is defined as follows [87]:

$$c_{RT}(s) = \bar{c}(a[s_{curr}, s]) + \tilde{c}(a^*[s, s_{fin}]), \quad (2.50)$$

where $\bar{c}(a[s_{curr}, s])$ stands for the cost of the path from the state s_{curr} to state s and $\tilde{c}(a^*[s, s_{fin}])$, as above, stands for the estimated cost of the optimal path from state s to the final state s_{fin} .

Given a finite graph $\mathcal{G} = (S, E)$, the RTA* algorithm that implements the evaluation cost $c_{RT}(s)$ acts as follows [87]. The algorithm maintains a list of *visited states* that includes states s with their estimated costs $\tilde{c}(s) = \tilde{c}(a^*[s, s_{fin}])$, which were actually visited by the searcher during its movement over the set of states S . The first state that is marked as *visited* is the initial state s_{init} . Starting from this initial state s_{init} , that is, with $s_{curr} = s_{init}$, the algorithm expands the current state s_{curr} and obtains the set $N(s_{curr})$ of its successors in the graph \mathcal{G} . For each state $s \in N(s_{curr})$ such that it was not marked as *visited*, the estimated cost $\tilde{c}(s)$ is calculated. Then, for each state $s \in N(s_{curr})$, the evaluation cost $c_{RT}(s)$ is obtained; for the already *visited* states s the evaluation cost $c_{RT}(s)$ is calculated by using the previously estimated costs, and for the states which are not marked as *visited*, the calculation is based on the new estimated costs. The next state s_{next} is chosen so that its evaluated cost reaches its minimum over all successors $s \in N(s_{curr})$ of the current state s_{curr} , that is,

$$s_{next} = \operatorname{argmin}_{s \in N(s_{curr})} \{c_{RT}(s)\}. \quad (2.51)$$

Note that, in contrast to the previous selection, the next state is chosen over all successors of the current state, while the evaluation costs of the successors are updated. After the selection of s_{next} , the current state s_{curr} is marked as *visited*, and the second minimal estimated cost is associated with this state; that is, if $s' = \operatorname{argmin}_{s \in (N(s_{curr}) \setminus \{s_{next}\})} \{c_{RT}(s)\}$, then $\tilde{c}(s_{curr})$ is updated as $\tilde{c}(s_{curr}) \leftrightarrow \tilde{c}(s')$. Finally, the searcher moves to the next state s_{next} , which from that moment is considered as the current state, that is, the current state s_{curr}

is updated as $s_{curr} \leftrightarrow s_{next}$. The states that are marked *visited* correspond to the resulting path. The above description of the RTA* algorithm is formalized as follows.

Algorithm 2.12 (RTA* algorithm) [87, 89]. Given graph $\mathcal{G} = (S, E)$, states s_{init} and s_{fin} , $s_{init}, s_{fin} \in S$, cost function c_{RT} defined by Equation 2.50, and initial costs $c_0(s)$ do:

1. Define $ResPath$ as a list of states.
2. Set $s_{curr} = s_{init}$.
3. While $s_{curr} \neq s_{fin}$ do:
 - 3.1 Expand s_{curr} to obtain the set $N(s_{curr})$ of its successors.
 - 3.2 Set $s_{next} = \operatorname{argmin}_{s \in N(s_{curr})} \{c_{RT}(s)\}$; ties are broken randomly.
 - 3.3 Set $\tilde{c}(s_{curr}) = \tilde{c}(s')$, where $s' = \operatorname{argmin}_{s \in (N(s_{curr}) \setminus \{s_{next}\})} \{c_{RT}(s)\}$.
 - 3.4 $ResPath.\ include(s_{curr})$.
 - 3.5 Set $s_{curr} = s_{next}$.
4. Return $ResPath$. ■

The implementation of the RTA* algorithm is similar to that of the BF* algorithm (Procedure 2.5) and requires functions and macros with the same functionality.

Recall that the graph $\mathcal{G} = (S, E)$ is finite, that is, both sets S and E contain a finite number of elements, and the costs $\bar{c}(s)$, $\tilde{c}(s)$, and $c_{RT}(s) = \bar{c}(s) + \tilde{c}(s)$ are non-negative and finite. Then, regarding the RTA* algorithm (Algorithm 2.12), the following statement holds.

Theorem 2.12 [87]. *If for every state $s \in S$, including initial state s_{init} , there exists a path $a[s, s_{fin}]$ from this state to the final state s_{fin} , then Algorithm 2.12 (RTA* algorithm) terminates with a resulting path.*

Proof. Assume, on the contrary, that the RTA* algorithm does not terminate. Then, since the graph \mathcal{G} is finite, there exists a cyclic path $a[s', s'']$ such that $s_{fin} \notin a[s', s'']$, and the algorithm follows this cycle for ever. In addition, according to the requirement of the theorem, in the cycle $a[s', s'']$ there is at least one state $s \in a[s', s'']$ such that there exists a path $a[s, s_{fin}]$.

Since the algorithm does not leave a cycle $a[s', s'']$ following the path $a[s, s_{fin}]$ from the state $s \in a[s', s'']$, the evaluated cost $c_{RT}(s)$ of this state is greater than the evaluated costs of the other states in the cycle. However, at each step, the algorithm updates the value of the estimated cost of the current state s_{curr} to the second minimal value of the evaluated cost (see Line 3.3). Since, at the previous step, the current state s_{curr} was selected, its evaluated cost was minimal, and the updating at the current step increases this cost. Thus, at each traverse around the cycle $a[s', s'']$, the algorithm increases the costs of the states with minimal previously assigned costs.

At a certain traverse, the costs of the neighbors of the state $s \in a[s', s'']$, from which the path $a[s, s_{fin}]$ to the final state starts, will become greater than the cost $c_{RT}(s)$.

Then, according to Line 3.2, the algorithm will choose state s . If the cost of the state s_{foll} , which follows state s in the path $a[s, s_{fin}]$, is less than the costs of the neighbors of s in the cycle $a[s', s'']$, then the algorithm leaves a cycle in contradiction to the assumption.

Otherwise, the algorithm returns to the cycle and continues increasing the costs up to the moment when $c_{RT}(s_{foll})$ is less than the costs of the neighbors of s in the cycle $a[s', s'']$. Since the costs are not bounded and the graph is finite, this moment will be reached in a finite number of walks around the cycle. ■

Note that at each step the RTA* algorithm makes locally optimal decisions. However, since the RTA* algorithm (Algorithm 2.12) implements local information regarding the target's state and already traversed path, it does not guarantee finding the optimal path from the initial state s_{init} to the final state s_{fin} . In addition, since the RTA* algorithm requires admissibility (Equation 2.48) of the estimated costs $\tilde{c}(s)$ for finding a sufficiently good path, but does not preserve their admissibility during the trial, it cannot implement the obtained information regarding the costs for further trials.

An algorithm that extends the RTA* algorithm and overcomes these problems was also suggested by Korf [87] and is called the Learning Real-Time A* algorithm. The LRTA* algorithm implements the same evaluation cost function (Equation 2.50) and decision-making condition (Equation 2.51), while the updating in the LRTA* algorithm differs from the updating used in the RTA* algorithm (Algorithm 2.12, Line 3.3). An outline of the LRTA* algorithm includes the following actions.

Algorithm 2.13 (LRTA* algorithm) [87, 90]. Given the graph $\mathcal{G} = (S, E)$, states s_{init} and $s_{fin}, s_{init}, s_{fin} \in S$, cost function c_{RT} defined by Equation 2.50, and initial costs $c_0(s)$ do:

1. Define $ResPath$ as a list of states.
2. Set $s_{curr} = s_{init}$.
3. While $s_{curr} \neq s_{fin}$ do:
 - 3.1 Expand s_{curr} to obtain the set $N(s_{curr})$ of its successors.
 - 3.2 Set $\tilde{c}(s_{curr}) = \max\{\tilde{c}(s_{curr}), \min_{s \in N(s_{curr})}\{c_{RT}(s)\}\}$.
 - 3.3 Set $s_{next} = \operatorname{argmin}_{s \in N(s_{curr})}\{c_{RT}(s)\}$; ties are broken randomly.
 - 3.4 $ResPath.include(s_{curr})$.
 - 3.5 Set $s_{curr} = s_{next}$.
4. Return $ResPath$.

The outline of the algorithm follows the definition presented by Shimbo and Ishida [90]. Note that in other formulations of the LRTA* algorithm (see, e.g., [89, 91]) Line 3.2 is substituted by its implementation:

- 3.2.a. Set $\tilde{c}'(s_{curr}) = \min_{s \in N(s_{curr})}\{c_{RT}(s)\}$.
- 3.2.b. If $\tilde{c}'(s_{curr}) > \tilde{c}(s_{curr})$ then: Set $\tilde{c}(s_{curr}) = \tilde{c}'(s_{curr})$.

Similar to the RTA* algorithm (Algorithm 2.12), the formulated LRTA* algorithm always terminates and results in a path $a[s_{init}, s_{fin}]$ from the initial state s_{init} to the final state s_{fin} , if it exists. Moreover, during the trial it preserves the admissibility of the estimated costs $\tilde{c}(s)$. However, in contrast to the RTA* algorithm, it does not make locally optimal decisions, but converges with repetitions of the trials to the optimal path $a^*[s_{init}, s_{fin}]$. These properties of the LRTA* algorithm (Algorithm 2.13) have been demonstrated by Korf [87]; below, following the line of Shimbo and Ishida [90], we formulate them in the form of separate statements.

Recall that, according to the definition (Equation 2.46) of actual evaluated cost $c^*(s) = \bar{c}^*(s) + \tilde{c}^*(s)$, the value $\tilde{c}^*(s) = \tilde{c}^*(a^*[s, s_{fin}])$ represents the *actual cost* of an optimal path from state $s \in S$ to the final state s_{fin} . Moreover, since in the LRTA* algorithm (Algorithm 2.13), as well as in the RTA* algorithm (Algorithm 2.12), at each step of search the current state is considered as an initial state s_{init} of the further path, the cost $\bar{c}^*(s) = \bar{c}^*(a^*[s_{init}, s]) = \bar{c}^*(a^*[s, s]) = 0$. Thus, in the RTA* and LRTA* algorithms, it follows that $c^*(s) = c^*(a^*[s, s_{fin}]) = \tilde{c}^*(s) = \tilde{c}^*(a^*[s, s_{fin}])$ for every state $s \in S$. Note that for the initial state s_{init} the value $c^*(s_{init}) = c^*(a^*[s_{init}, s_{fin}])$ and for the A* and RTA* and LRTA* algorithms provides the actual evaluated cost of an optimal path from the initial state s_{init} to the final state s_{fin} .

The properties of the LRTA* algorithm (Algorithm 2.13) are based on the admissibility assumption (Equation 2.48) regarding initial costs; that is, in all statements we assume that for every state $s \in S$ it follows that $c_0(s) = \tilde{c}_0(s) \leq \tilde{c}^*(s)$.

Theorem 2.13 [87]. (parallel to Theorem 2.12). *If for every state $s \in S$, including initial state s_{init} , there exists a path $a[s, s_{fin}]$ from this state to the final state s_{fin} , then Algorithm 2.13 (LRTA* algorithm) terminates with a resulting path.*

Proof. According to Line 3.2 of Algorithm 2.13, the estimated cost $\tilde{c}(s_{curr}) = \tilde{c}(a^*[s_{curr}, s_{fin}])$, which is stored by Line 3.4 with the current state s_{curr} , is increased with the cost $\bar{c}(a[s_{curr}, s])$ of the step to its neighbor $s \in N(s_{curr})$. Thus, the stored estimated cost $\tilde{c}(s_{curr})$ of the current state s_{curr} is always strictly greater than the estimated costs $\tilde{c}(s)$ of its neighbors $s \in N(s_{curr})$. From this point, the required statement is obtained by the same reasoning as in Theorem 2.12. ■

The next lemma provides the property that differentiates the LRTA* algorithm from the RTA* algorithm, and provides a basis for a further theorem.

Lemma 2.12 [87]. *Throughout the trial of Algorithm 2.13 (LRTA* algorithm) and at its termination, for all states $s \in S$ the admissibility property $\tilde{c}(s) \leq \tilde{c}^*(s)$ is preserved.*

Proof. Let $s_{curr} \in a^*[s_{init}, s_{fin}]$ be a state from the optimal path from the initial state s_{init} to the final state s_{fin} and let s_{full} be a state through which the optimal path has to pass after s_{curr} . Since s_{full} follows after state s_{curr} in the optimal path, it is one of its neighbors, that is, $s_{full} \in N(s_{curr})$.

Consider the update of the estimated cost $\tilde{c}(s_{curr})$ in Line 3.2. If $\tilde{c}(s_{curr}) \geq \min_{s \in N(s_{curr})} \{c_{RT}(s)\}$, then its value is not changed. According to the assumption, this value is admissible, and the lemma is proven.

Assume that $\tilde{c}(s_{curr}) < \min_{s \in N(s_{curr})} \{c_{RT}(s)\}$. Then, since s_{foll} is the next state in the optimal path, $\tilde{c}(s_{curr})$ will be updated to the value

$$\tilde{c}'(s_{curr}) = \min_{s \in N(s_{curr})} \{c_{RT}(s)\} = \bar{c}(s_{foll}) + \tilde{c}(s_{foll}).$$

According to the admissibility assumption (Equation 2.48), $\tilde{c}(s_{foll}) \leq \tilde{c}^*(s_{foll})$. Hence, the updated value of the cost is

$$\tilde{c}'(s_{curr}) \leq \bar{c}(s_{foll}) + \tilde{c}^*(s_{foll}).$$

But s_{foll} is in the optimal path, so then $\bar{c}(s_{foll}) = \tilde{c}^*(s_{foll})$. Thus on the right side one obtains $\bar{c}(s_{foll}) + \tilde{c}^*(s_{foll}) = \tilde{c}^*(a^*[s_{curr}, s_{foll}]) + \tilde{c}(a^*[s_{foll}, s_{fin}]) = \tilde{c}^*(s_{curr})$, and, finally, $\tilde{c}'(s_{curr}) \leq \tilde{c}^*(s_{curr})$, as required. ■

Since the LRTA* algorithm (Algorithm 2.13), in contrast to the RTA* algorithm (Algorithm 2.12), preserves the admissibility (Equation 2.48) of the estimated costs $\tilde{c}(s)$, the costs, which are obtained by updating during the trial, can be used for further trials. Moreover, regarding repetitions of the trials, the following theorem holds.

Theorem 2.14 [87]. *Algorithm 2.13 (LRTA* algorithm) converges with trials to the optimal path $a^*[s_{init}, s_{fin}]$ with minimal actual evaluated cost $c^*(a^*[s_{init}, s_{fin}])$ and such that for each state $s \in S$ it follows that $\tilde{c}(s) = \tilde{c}^*(s)$.*

Proof. Let s_{curr} be a state and $s_{next} \in N(s_{curr})$ be its successor that, after a choice by Line 3.3 of the algorithm, would follow the state s_{curr} in the resulting path. Then, after updating, the cost of the current state will obtain the value $\tilde{c}(s_{curr}) = \bar{c}(s_{next}) + \tilde{c}(s_{next})$.

Assume that, after termination of a certain trial of the algorithm, for the cost of the state s_{next} it follows that $\tilde{c}(s_{next}) = \tilde{c}^*(s_{next})$. Then $\tilde{c}(s_{curr}) = \bar{c}(s_{next}) + \tilde{c}^*(s_{next})$. Since the state s_{next} was chosen, the value $\tilde{c}(s_{curr})$, which is obtained after the updating in Line 3.2, is minimal over all successors of the state s_{curr} . The cost $\tilde{c}^*(s_{next})$ is already minimal as a cost of the optimal path $a^*[s_{next}, s_{fin}]$ from the state s_{next} to the final state s_{fin} . Thus, the cost $\bar{c}(s_{next})$ is minimal, that is, $\bar{c}(s_{next}) = \tilde{c}^*(s_{next})$. So the cost of the state s_{curr} is updated to the value $\tilde{c}(s_{curr}) = \bar{c}^*(s_{next}) + \tilde{c}^*(s_{next}) = \tilde{c}^*(s_{curr})$, which is the cost of the optimal path $a^*[s_{curr}, s_{fin}]$ from the state s_{curr} to the final state s_{fin} .

Let $a[s_{init}, s_{fin}]$ be a path from the initial state s_{init} to the final state s_{fin} , and let $s \in a[s_{init}, s_{fin}]$ be the last state in the path such that $\tilde{c}(s_{curr}) \leq \tilde{c}^*(s_{curr})$, while for the state $s_{foll} \in a[s_{init}, s_{fin}]$, which follows after the state s_{curr} in the path $a[s_{init}, s_{fin}]$, it follows that $\tilde{c}(s_{foll}) = \tilde{c}^*(s_{foll})$. Such a state s_{foll} always exists, and in the worst case $s_{foll} = s_{fin}$ with $\tilde{c}(s_{foll}) = \tilde{c}^*(s_{fin}) = 0$. Then, following the reasoning above, one obtains that when the algorithm reaches state s , its cost $\tilde{c}(s)$ is updated to the cost $\tilde{c}^*(s)$ of the optimal path $a^*[s, s_{fin}]$. Such an updating holds at each trial of the algorithm, hence by every pass over the path $a[s_{init}, s_{fin}]$, the costs $\tilde{c}(s)$ of states s , after which the states s_{foll} follow with $\tilde{c}(s_{foll}) = \tilde{c}^*(s_{foll})$, are updated to the costs $\tilde{c}^*(s)$.

From the admissibility of the estimated costs and the rule of updating (see Line 3.2 in Algorithm 2.13) it follows that if for each pair of states s' and s'' , $s' \neq s''$, $\tilde{c}(s') \neq \tilde{c}(s'')$ and $\tilde{c}^*(s') \neq \tilde{c}^*(s'')$, then the number of states s with the costs which were updated to $\tilde{c}^*(s)$ increases with trials. Since the number of states is finite, after a finite number of trials all states which are included in the resulting path will have the costs $\tilde{c}^*(s)$, as required.

Assume, in contrast, that there exist states s' and s'' , $s' \neq s''$, such that $\tilde{c}(s') = \tilde{c}(s'')$ or $\tilde{c}^*(s') = \tilde{c}^*(s'')$ or both. Recall that the choice between the states with equal costs in Line 3.3 of the algorithm is conducted randomly. Since the number of states is finite, an infinite number of random choices will break all possible ties. Thus, by an infinite number of trials the costs of all states s , including the states in the resulting path, are updated to the actual costs $\tilde{c}^*(s)$, which gives an optimal path $a^*[s_{init}, s_{fin}]$ with actual evaluated cost $\tilde{c}^*(s_{init}) = c^*(a^*[s_{init}, s_{fin}])$. ■

The actions of the LRTA* algorithm (Algorithm 2.13) are illustrated in the following example. For consistency, we implement the LRTA* algorithm over the same graph as in the previous examples illustrating the actions of the BF* and A* algorithms.

Example 2.22 Let the states graph $\mathcal{G} = (S, E)$ and the costs and their estimations be the same as before in Examples 2.19 and 2.20. For convenience, this graph is shown again in Figure 2.33a. Over the presented graph, the LRTA* algorithm (Algorithm 2.13) acts as follows.

Trial 1. As above, the searcher starts with the initial state $s^{t=0} = s_{init}$, and, expanding it, obtains the single successor $s^{t=1} = s_1$. Following the above reasoning, the evaluated cost $c_{RT}(s_1)$ is calculated as follows:

$$c_{RT}(s_1) = \bar{c}(a[s_{init}, s_1]) + \tilde{c}(a^*[s_1, s_{fin}]) = 3 + 6 = 9,$$

and the estimated cost $\tilde{c}(s_{init})$ is updated to this value $\tilde{c}(s_{init}) \leftarrow c_{RT}(s_1) = 9$. Then the searcher moves to the state $s^{t=1} = s_1$.

Expansion of state s_1 results in the successor set $N(s_1) = \{s_2, s_3\}$. According to the graph, the estimated costs of states s_2 and s_3 are $\tilde{c}(a^*[s_2, s_{fin}]) = 2$ and $\tilde{c}(a^*[s_3, s_{fin}]) = 4$, while the costs of the movement from state s_1 to these states are $\bar{c}(a[s_1, s_2]) = 5$ and $\bar{c}(a[s_1, s_3]) = 3$. Thus, the evaluated costs for states s_2 and s_3 are calculated as follows:

$$c_{RT}(s_2) = \bar{c}(a[s_1, s_2]) + \tilde{c}(a^*[s_2, s_{fin}]) = 5 + 2 = 7,$$

$$c_{RT}(s_3) = \bar{c}(a[s_1, s_3]) + \tilde{c}(a^*[s_3, s_{fin}]) = 3 + 4 = 7.$$

Since the evaluated costs are equal, the choice between these states is conducted randomly. Assume that, as above, the chosen state is s_2 . Then the estimated cost $\tilde{c}(s_1)$ of state s_1 is updated to the value $\tilde{c}(s_1) \leftarrow c_{RT}(s_2) = 7$, and the searcher moves to the state $s^{t=2} = s_2$.

Expansion of state s_2 results in its single successor s_4 with estimated cost $\tilde{c}(a^*[s_4, s_{fin}]) = 1$, while the cost of movement from state s_2 to state s_4 is $\bar{c}(a[s_2, s_4]) = 2$. Thus the evaluated cost of state s_4 is

$$c_{RT}(s_4) = \bar{c}(a[s_2, s_4]) + \tilde{c}(a^*[s_4, s_{fin}]) = 2 + 1 = 3.$$

The searcher updates the estimated cost $\tilde{c}(s_2) \leftarrow c_{RT}(s_4) = 3$ and moves to the state $s^{t=3} = s_4$.

Expansion of state s_4 results in the final updating

$$\tilde{c}(s_4) \leftarrow c_{RT}(s_{fin}) = \bar{c}(a[s_4, s_{fin}]) + \tilde{c}(a^*[s_{fin}, s_{fin}]) = 2 + 0 = 2,$$

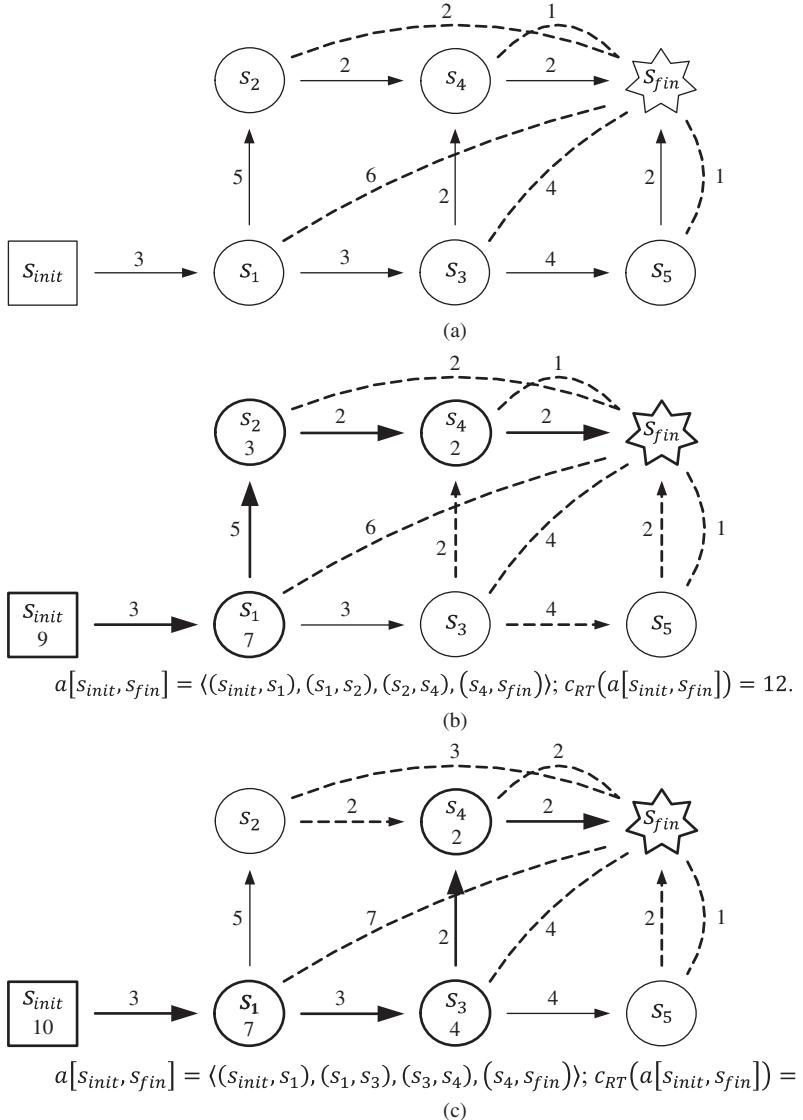


Figure 2.33 Example of the expanded states and resulting paths obtained by two trials of the LRTA* algorithm. (a) Graph of the states with initial estimated costs $c_0(s)$ and costs $\bar{c}(s) = c^*(a^*[s, s_{next}])$. (b) Resulting path and updated costs after the first trial. (c) Resulting path and updated costs after the second trial.

and movement to final state $s^{t=4} = s_{fin}$. The resulting path

$$a[s_{init}, s_{fin}] = \langle (s_{init}, s_1), (s_1, s_2), (s_2, s_4), (s_4, s_{fin}) \rangle$$

has the evaluated cost $c_{RT}(a[s_{init}, s_{fin}]) = 3 + 5 + 2 + 2 = 12$, and it is not optimal. The path with updated costs, which are obtained in this trial of the LRTA* algorithm, are shown in Figure 2.33b.

Trial 2. Since the LRTA* algorithm (Algorithm 2.13) preserves admissibility of the estimated costs, the updated values can be used for the next trial. Recall that during the first trial the estimated costs of the states were updated, so, in contrast to the initial costs, the costs after the first trial are as follows:

$$\tilde{c}(s_1) = 7, \quad \tilde{c}(s_2) = 3, \quad \tilde{c}(s_4) = 2.$$

Then, the algorithm at the second trial implements these costs and acts as follows (see Figure 2.33c). Expanding the initial state $s^{t=0} = s_{init}$, the searcher calculates the evaluated cost $c_{RT}(s_1)$, which in this trial obtains the following value:

$$c_{RT}(s_1) = \bar{c}(a[s_{init}, s_1]) + \tilde{c}(a^*[s_1, s_{fin}]) = 3 + 7 = 10.$$

Thus, the estimated cost $\tilde{c}(s_{init})$ is updated as $\tilde{c}(s_{init}) \leftarrow c_{RT}(s_1) = 10$, and the searcher moves to the state $s^{t=1} = s_1$.

Expansion of state s_1 results in its successors s_2 and s_3 with the evaluated costs

$$c_{RT}(s_2) = \bar{c}(a[s_1, s_2]) + \tilde{c}(a^*[s_2, s_{fin}]) = 5 + 3 = 8,$$

$$c_{RT}(s_3) = \bar{c}(a[s_1, s_3]) + \tilde{c}(a^*[s_3, s_{fin}]) = 3 + 4 = 7.$$

Note that in the calculation of the cost $c_{RT}(s_2)$ for state s_2 the updated value of the estimated cost $\tilde{c}(s_2) = \tilde{c}(a^*[s_2, s_{fin}]) = 3$ is used. Since $c_{RT}(s_3) < c_{RT}(s_2)$, state s_3 is chosen. The estimated cost $\tilde{c}(s_1)$ of state s_1 is updated to the value $\tilde{c}(s_1) \leftarrow c_{RT}(s_3) = 7$, and the searcher moves to the state $s^{t=2} = s_3$.

The successors of the chosen state s_3 are states s_4 and s_5 . The evaluated costs of these states have the following values:

$$c_{RT}(s_4) = \bar{c}(a[s_3, s_4]) + \tilde{c}(a^*[s_4, s_{fin}]) = 2 + 2 = 4,$$

$$c_{RT}(s_5) = \bar{c}(a[s_3, s_5]) + \tilde{c}(a^*[s_5, s_{fin}]) = 4 + 1 = 5.$$

Here, the updated estimated cost $\tilde{c}(s_4) = \tilde{c}(a^*[s_4, s_{fin}]) = 2$ is applied. Since $c_{RT}(s_4) < c_{RT}(s_5)$, the searcher chooses state s_4 . Then, the estimated cost $\tilde{c}(s_3)$ of state s_3 is updated to the value $\tilde{c}(s_3) \leftarrow c_{RT}(s_4) = 4$, and the searcher moves to the state $s^{t=3} = s_4$.

The final step of the algorithm in the second trial is similar to the final step of the first trial. State s_4 is expanded and, after updating its estimated cost by $\tilde{c}(s_4) \leftarrow c_{RT}(s_{fin}) = 2$, the searcher moves to the final state $s^{t=4} = s_{fin}$ and the search terminates.

The resulting path of this trial is optimal

$$a[s_{init}, s_{fin}] = \langle (s_{init}, s_1), (s_1, s_3), (s_3, s_4), (s_4, s_{fin}) \rangle,$$

and its evaluated cost is $c_{RT}(a[s_{init}, s_{fin}]) = 3 + 3 + 2 + 2 = 10$. The obtained path with updated costs, which are obtained in the second trial of the LRTA* algorithm, are shown in Figure 2.33c.

Note that the resulting path $a[s_{init}, s_{fin}]$, which is obtained by the LRTA* algorithm in the last trial, is similar to the paths obtained by the BF* algorithm (Example 2.20) and A* algorithm (Example 2.21); however, the way in which this path is obtained differs from the way in which it is obtained in both the BF* and A* algorithms. ■

The LRTA* algorithm (Algorithm 2.13) forms a basis for different algorithms of real-time search over graphs, and in recent years a number of versions of the LRTA* algorithm have been suggested. These versions were generated by different definitions of the cost function c_{RT} and assumptions regarding the structure of the graphs $\mathcal{G} = (S, E)$, on which the algorithm acts. An overview of such algorithms and analysis of the learning process have been presented by Shimbo and Ishida [90]; for a general framework of learning real-time search and its applications see, for example, [91, 92]. Further modifications of the LRTA* algorithm, which have been developed in the last few years, and other real-time search algorithms based on the A* algorithm can be found in, for example, [89, 93–95]. In the following chapters, we will consider an informational version of the LRTA* algorithm and its application in search for a static target.

As indicated above, the considered algorithms define search for a static target. Below, we present certain modifications of the A* and LRTA* algorithms that address search for a moving target.

2.3.3 Moving target search and the fringe-retrieving A* algorithm

The algorithm of search for a moving target over a finite graph, which is called the *Moving Target Search* algorithm, was suggested in 1991 by Ishida and Korf [77, 96]. This algorithm is based on the LRTA* algorithm (Algorithm 2.13), but, in addition, takes into account the possible movements of the target. Similar to the A* and LRTA* algorithms, the MTS algorithm acts over a finite states graph $\mathcal{G} = (S, E)$, where the complete set S is considered as a set of *searcher's states*. In addition, a set $S_{tar} \subseteq S$ and an appropriate set $E_{tar} \subseteq E$ are defined; the set S_{tar} is considered as a set of *target's states*. If the movements of the target are not restricted relatively to the searcher's movements, then it is assumed that $S_{tar} = S$ and $E_{tar} = E$.

In the MTS algorithm it is assumed that an initial state s_{init} of the searcher is specified, but, in contrast to the A* and LRTA* algorithms, the final searcher's state that represents a node, in which the target is located, is not defined and depends on the target's movements. In addition to the *initial state* $s_{init} \in S$ of the searcher, in the MTS algorithm an *initial state* $y_{init} \in S_{tar}$ of the target is specified. It is assumed that the target starts from its initial state y_{init} and at each step of the search moves one step over the set S_{tar} or stays in its current state. The searcher cannot control the target's movement up to termination of the algorithm, and the target is not informed of the searcher's behavior. The search stops when the states of the searcher and the target are identical – thus, the target is found by the searcher.

Let, as above, $\mathcal{G} = (S, E)$ be a finite but not necessarily directed graph, and let $S_{tar} \subseteq S$ be a set of the target's states. Similar to the A* and LRTA* algorithms, the MTS algorithm implements the estimated cost function \tilde{c} and actual cost function \tilde{c}^* . However, in contrast to the previous algorithms, for the MTS these functions specify the costs with respect to the target's state $y \in S_{tar}$ rather than to the final state $s_{fin} \in S$. In particular, given the searcher's state $s \in S$ and the target's state $y \in S_{tar}$, these functions specify the *estimated cost* $\tilde{c}(s, y) = \tilde{c}(a^*[s, y])$ and the *actual cost* $\tilde{c}^*(s, y) = \tilde{c}^*(a^*[s, y])$ of the optimal path $a^*[s, y]$ from the searcher's state s to the target's state y [77, 96]. In a certain sense, these costs can be considered as the costs of the path $a^*[s, s_{fin}]$ with varying final state s_{fin} .

Since the searcher is not informed about the target's movements, in the MTS algorithm two evaluation cost functions are defined, c_{MT}^+ and c_{MT}^- , that correspond to the searcher's and the target's movements. The functions, correspondingly, define the evaluated costs

$c_{MT}^+(s, y) = c_{MT}^+(a[s, y])$ and $c_{MT}^-(s, y) = c_{MT}^-(a[s, y])$ of paths from the searcher's states $s \in S$ to the target's states $y \in S_{tar}$ that are based on the estimated costs $\tilde{c}(s, y)$ and are obtained as follows [77, 96]:

$$c_{MT}^+(s, y) = \bar{c}(a[s_{curr}, s]) + \tilde{c}(a^*[s, y]) = 1 + \tilde{c}(a^*[s, y]), \quad (2.52a)$$

$$c_{MT}^-(s, y) = -\bar{c}(a[s_{curr}, s]) + \tilde{c}(a^*[s, y]) = -1 + \tilde{c}(a^*[s, y]), \quad (2.52b)$$

where $s_{curr} \in S$ is the current state of the searcher and $s \in S$ is a successor of the state s_{curr} , that is, $s \in N(s_{curr})$. From these definitions of the evaluation cost it follows that in the original form the cost of the movement s to the successor of the state s_{curr} does not depend on the actual weights of the edges of the graph \mathcal{G} and are specified by constant values $\bar{c}(a[s_{curr}, s]) = 1$, or, alternatively, all the graph weights are equal to 1. The first function (Equation 2.52a) corresponds to the searcher's movement that increases the movements' cost, while the second function (Equation 2.52b) corresponds to movement of the target while the searcher does not move, so the cost of the searcher's movements decreases.

By using the defined costs, the MTS algorithm is outlined as follows.

Algorithm 2.14 (MTS algorithm) [77, 96]. Given graph $\mathcal{G} = (S, E)$, initial searcher's state $s_{init} \in S$, initial target's state $y_{init} \in S_{tar} \subseteq S$, cost functions c_{MT}^+ and c_{MT}^- defined by Equation 2.52, and initial costs $c_0(s, y)$ do:

1. Set $s_{curr} = s_{init}$.
2. Set $y_{curr} = y_{init}$.
3. While $s_{curr} \neq y_{curr}$ do:
 - Searcher's turn: Implements evaluation cost function c_{MT}^+ (Equation 2.52a)
 - 3.1 Expand s_{curr} to obtain the set $N(s_{curr})$ of its successors.
 - 3.2 Set $s_{next} = \operatorname{argmin}_{s \in N(s_{curr})} \{\tilde{c}(s, y_{curr}) + 1\}$; ties are broken randomly.
 - 3.3 Set $\tilde{c}(s_{curr}, y_{curr}) = \max\{\tilde{c}(s_{curr}, y_{curr}), \min_{s \in N(s_{curr})} \{\tilde{c}(s, y_{curr}) + 1\}\}$.
 - 3.4 Set $s_{curr} = s_{next}$.
 - Target's turn: Implements evaluation cost function c_{MT}^- (Equation 2.52b)
 - 3.5 Target moves from y_{curr} to $y_{next} \in N(y_{curr})$ or stays in y_{curr} .
 - 3.6 Set $\tilde{c}(s_{curr}, y_{curr}) = \max\{\tilde{c}(s_{curr}, y_{curr}), \tilde{c}(s_{curr}, y_{next}) - 1\}$.
 - 3.7 Set $y_{curr} = y_{next}$.
4. Return s_{curr} . ■

Notice again that the updating scheme in Line 3.3 implies that, since any path from the current searcher's state s_{curr} to the current target's state y_{curr} must pass through one of the successors $s \in N(s_{curr})$ of the state s_{curr} , the estimated cost $\tilde{c}(a^*[s_{curr}, y_{curr}])$ of the path from s_{curr} to y_{curr} must be as large as the minimum cost of the path $\tilde{c}(a^*[s, y_{curr}])$ through any of the $s \in N(s_{curr})$. Similarly, the updating scheme in Line 3.6 implies that, since the current y_{curr} and next y_{next} states of the target are at most one unit apart, as $\bar{c}(a[y_{curr}, y_{next}]) = 1$, the estimated cost $\tilde{c}(a^*[s_{curr}, y_{curr}])$ of the path to the target's current

state y_{curr} must be at least as large as the estimated cost $\tilde{c}(a^*[s_{curr}, y_{next}])$ of the path to the next state y_{next} minus 1.

Let $s', s'' \in S$ be two states and assume that for the estimated costs $\tilde{c}(s', s'')$ and the actual costs $\tilde{c}^*(s', s'')$ it follows that

$$\tilde{c}(s', s'') \leq \tilde{c}^*(s', s''). \quad (2.53)$$

This assumption is a straightforward generalization of the admissibility assumption (Equation 2.48) that was implemented in the A* and LRTA* algorithms. As indicated above, the estimated costs, which meet the inequality in Equation 2.53, are called *admissible*, whereas the inequality in Equation 2.53 is called the *admissibility assumption*. Since $S_{tar} \subseteq S$, for the costs of the paths between the searcher's states $s \in S$ and the target's states $y \in S_{tar}$ the admissibility assumption (Equation 2.53) holds, that is,

$$\tilde{c}(s, y) \leq \tilde{c}^*(s, y). \quad (2.53a)$$

As proven by Ishida and Korf [77, 96], if the initial costs are admissible, that is, it follows that $c_0(s, y) = \tilde{c}_0(s, y) \leq \tilde{c}^*(s, y)$, then for the MTS algorithm (Algorithm 2.14) the following properties hold. In general, we assume that the admissibility (Equation 2.53) of the costs of all states in the graph $\mathcal{G} = (S, E)$ is satisfied.

Lemma 2.13 [77, 96]. (parallel to Lemma 2.12). *Throughout the trial of Algorithm 2.14 (MTS algorithm) and at its termination, for all searcher's states $s \in S$ and target's states $y \in S_{tar}$ the admissibility property $\tilde{c}(s, y) \leq \tilde{c}^*(s, y)$ is preserved.*

Proof. Let us start with the first updating in Line 3.3, which is based on the function c_{MT}^+ (Equation 2.52a). If $\tilde{c}(s_{curr}, y_{curr}) \geq \min_{s \in N(s_{curr})}\{\tilde{c}(s, y_{curr}) + 1\}$, then, according to Line 3.3, $\tilde{c}(s_{curr}, y_{curr})$ does not change and the admissibility is preserved. Assume that $\tilde{c}(s_{curr}, y_{curr}) < \min_{s \in N(s_{curr})}\{\tilde{c}(s, y_{curr}) + 1\}$, and let $s_{next} \in N(s_{curr})$ be a state in which the minimum is reached. Then, the estimated cost $\tilde{c}(s_{curr}, y_{curr}) = \tilde{c}(a[s_{curr}, s_{next}] \circ a^*[s_{next}, y_{curr}])$ of the path from the current state s_{curr} to the state y_{curr} via the state s_{next} is as follows (operation ‘ \circ ’ stands for a concatenation of the paths):

$$\tilde{c}(s_{curr}, y_{curr}) = \tilde{c}(a[s_{curr}, s_{next}]) + \tilde{c}(a^*[s_{next}, y_{curr}]) = 1 + \tilde{c}(a^*[s_{next}, y_{curr}]),$$

where the equivalence $\tilde{c}(a[s_{curr}, s_{next}]) = \bar{c}(a[s_{curr}, s_{next}]) = 1$, which follows from the assumption that $s_{next} \in N(s_{curr})$ is implemented. Since $\tilde{c}(a^*[s_{next}, y_{curr}])$ is minimal, the cost $\tilde{c}(s_{curr}, y_{curr})$ is also minimal, hence the admissibility is preserved.

Now consider the second updating term in Line 3.6, which is based on the function c_{MT}^- (Equation 2.52b). Since the target moves to the successor state $y_{next} \in N(y_{curr})$ of its current state y_{curr} or stays in y_{curr} , the cost $\bar{c}(a[y_{curr}, y_{next}])$ of this movement is at most equal to 1. Hence, with respect to the direction of the target's movement, for the cost $\tilde{c}(s_{curr}, y_{next})$ it follows that $\tilde{c}(s_{curr}, y_{curr}) - 1 \leq \tilde{c}(s_{curr}, y_{next}) \leq \tilde{c}(s_{curr}, y_{curr}) + 1$, or, equivalently, $\tilde{c}(s_{curr}, y_{curr}) - 2 \leq \tilde{c}(s_{curr}, y_{next}) - 1 \leq \tilde{c}(s_{curr}, y_{curr})$. Thus, the updating scheme does not increase the cost $\tilde{c}(s_{curr}, y_{curr})$, and the admissibility is preserved. ■

Note that this lemma is based on the assumption of known unit costs of the paths between adjacent states, thus it is less general than the similar lemmas regarding the A* and LRTA* algorithms of search for a static target. The next theorem provides termination of the MTS algorithm by implementing the same assumption.

Theorem 2.15 [77, 96] (parallel to Theorem 2.13). *If for every pair of states $s \in S$ and $y \in S_{tar}$, including the initial states s_{init} and y_{init} , there exists a path $a[s, y]$ from the searcher's state s to the target's state y , and if the target periodically skips movement and stays in its current state y_{curr} , then Algorithm 2.14 (MTS algorithm) terminates.*

Proof. Let $CErr = \sum(\tilde{c}^*(a^*[s', s'']) - \tilde{c}(a^*[s', s'']))$, where the sum is calculated over all pairs of states $s', s'' \in S$, be the estimated cost error. Since the estimated costs are admissible, the estimated error is non-negative. In addition, let $CDis(s, y) = CErr - \tilde{c}(s, y)$ be the cost disparity between the searcher's state $s \in S$ and the target's state $y \in S_{tar}$.

Consider the movement of the searcher from state s_{curr} to state $s_{next} \in N(s_{curr})$. If $\tilde{c}(s_{next}, y_{curr}) < \tilde{c}(s_{curr}, y_{curr})$, then, according to Line 3.3 in Algorithm 2.14, the cost is not updated. Hence, following the searcher's movement (Line 3.4), the cost $\tilde{c}(s_{curr}, y_{curr})$ is at least one unit less than before the movement, and the disparity $CDis(s_{curr}, y_{curr})$ decreases by at least one unit. If, in contrast, $\tilde{c}(s_{next}, y_{curr}) \geq \tilde{c}(s_{curr}, y_{curr})$, then, according to Line 3.3, the cost $\tilde{c}(s_{curr}, y_{curr})$ is updated to $\tilde{c}(s_{next}, y_{curr}) + 1$. Hence, following the searcher's movement in Line 3.4, the cost error $CErr$ is updated so that in the sum, instead of the cost $\tilde{c}(s_{curr}, y_{curr})$, one obtains $\tilde{c}(s_{next}, y_{curr}) + 1$. As a result, the error $CErr$ decreases by $\tilde{c}(s_{next}, y_{curr}) + 1 - \tilde{c}(s_{curr}, y_{curr})$. At the same time, the cost $\tilde{c}(s_{curr}, y_{curr})$ is increased by $\tilde{c}(s_{next}, y_{curr}) - \tilde{c}(s_{curr}, y_{curr})$. Thus, again, the disparity $CDis(s_{curr}, y_{curr})$ decreases by at least one unit.

Now consider the updating scheme when the target moves from its current state y_{curr} to the state $y_{next} \in N(y_{curr})$. If $\tilde{c}(s_{curr}, y_{next}) \leq \tilde{c}(s_{curr}, y_{curr}) + 1$, then, according to Line 3.6, the cost is not updated. Hence, with respect to the direction of the target's movement, $\tilde{c}(s_{curr}, y_{curr})$ increases by at most one unit, so the cost error $CErr$ and the disparity $CDis(s_{curr}, y_{curr})$ also increase by one unit at most. If, in contrast, $\tilde{c}(s_{curr}, y_{next}) > \tilde{c}(s_{curr}, y_{curr}) + 1$, then, by Line 3.6, the cost $\tilde{c}(s_{curr}, y_{curr})$ is updated to $\tilde{c}(s_{curr}, y_{next}) - 1$. Hence, the cost error $CErr$ increases by $\tilde{c}(s_{curr}, y_{next}) - 1 - \tilde{c}(s_{curr}, y_{curr})$, while the cost $\tilde{c}(s_{curr}, y_{curr})$ is increased by $\tilde{c}(s_{curr}, y_{next}) - \tilde{c}(s_{curr}, y_{curr})$. As a result, once again, the disparity $CDis(s_{curr}, y_{curr})$ increases by at most one unit.

Hence, the movement of the searcher decreases the disparity by at least one unit, while the movement of the target increases the disparity by at most one unit. Thus, the consequent movements of the searcher and the target do not increase the disparity, and if the target periodically skips its movements and stays in its current state, the disparity decreases.

According to Line 3, Algorithm 2.14 terminates if the searcher's and the target's states are equivalent, that is, if $s_{curr} = y_{curr}$. Assume that such equivalence is not reached. As indicated above, the estimated costs are non-negative and finite, and from the admissibility property of the estimated costs it follows that $CErr$ is non-negative. From the definitions of the cost error $CErr$ and the disparity $CDis$ it follows that the disparity is also non-negative, and, since the estimated costs are finite, for the finite graph \mathcal{G} the value of the disparity is also finite. Hence, in the steps of the algorithm in which the target skips its movements periodically, the disparity $CDis(s_{curr}, y_{curr}) = CErr - \tilde{c}(s_{curr}, y_{curr})$ decreases

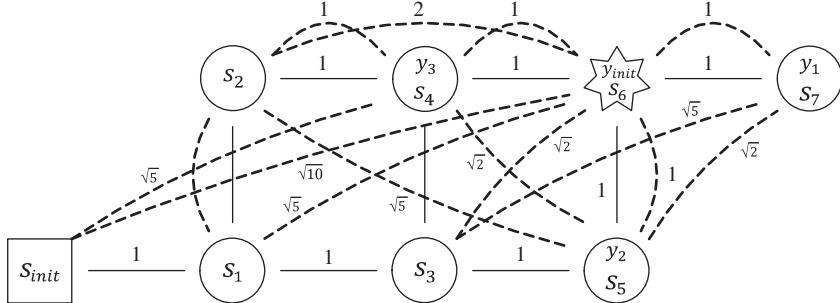


Figure 2.34 Example of the graph of states for the MTS algorithm.

and reaches a zero value. However, $CDis(s_{curr}, y_{curr}) = 0$ if and only if both $CErr = 0$ and $\tilde{c}(s_{curr}, y_{curr}) = 0$, which, according to the admissibility assumption, is satisfied if and only if $s_{curr} = y_{curr}$, which is a specific termination condition. ■

Note that the MTS algorithm implements a conservative strategy; hence, the search by the algorithm is not optimal. The actions of the MTS algorithm are illustrated by the following example.

Example 2.23 Assume that in the non-directed graph $\mathcal{G} = (S, E)$ the states set S includes eight states, $S = \{s_{init}, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$, which represent the searcher's states, and let the target's states be $S_{tar} = \{y_{init} = s_6, y_1 = s_7, y_2 = s_5, y_3 = s_4\}$. So the searcher starts with the state s_{init} and can move over all eight states from the set S , while the target starts with the state $y_{init} = s_6$ and can move directly to four states, s_4, s_5, s_6 , and s_7 . In addition, assume that, as required by the MTS algorithm, the costs of the adjacent states are equal to 1, and the estimated costs are admissible and calculated by the Euclidean distance between the states. The graph with the initial states of the searcher and the target and some values of actual and estimated costs is shown in Figure 2.34.

Let us consider the steps of the algorithm. Each step includes a turn of the searcher and a turn of the target.

Step 1. The searcher starts with the initial state $s_{curr} = s_{init}$, and expansion of this state results in its single successor s_1 . Thus the searcher chooses this state as the next state $s_{next} = s_1$ and updates the estimated cost as follows. For the searcher's current state $s_{curr} = s_{init}$ the estimated cost is $\tilde{c}(s_{curr}, y_{curr}) = \sqrt{10} \approx 3.16$, while the estimated cost for the next state $s_{next} = s_1$ is $\tilde{c}(s_{next}, y_{curr}) = \sqrt{5} \approx 2.24$. Hence, $\tilde{c}(s_{next}, y_{curr}) + 1 \approx 2.24 + 1 = 3.24 > \tilde{c}(s_{curr}, y_{curr}) \approx 3.16$, and the estimated cost of the current state s_{curr} is updated as $\tilde{c}(s_{curr}, y_{curr}) \leftarrow \tilde{c}(s_{next}, y_{curr}) + 1 = \sqrt{5} + 1$. After updating, the searcher moves to the state $s_{curr} = s_{next} = s_1$.

The target starts with the initial state $y_{curr} = y_{init} = s_6$, and its set of successors is $N(s_6) = \{s_4, s_5, s_7\}$. Assume that it chooses to move to the state $y_{next} = y_3 = s_4$. After the updating and movement at the searcher's turn, $\tilde{c}(s_{curr}, y_{curr}) = \sqrt{5} + 1 \approx 3.34$, while $\tilde{c}(s_{curr}, y_{next}) = \sqrt{2} \approx 1.41$. Thus $\tilde{c}(s_{curr}, y_{curr}) \approx 3.34 > \tilde{c}(s_{curr}, y_{next}) - 1 \approx 1.41 - 1 = 0.41$, and the value of the estimated cost $\tilde{c}(s_{curr}, y_{curr})$ is not changed. After updating, the target moves to the state $y_{curr} = y_3 = s_4$.

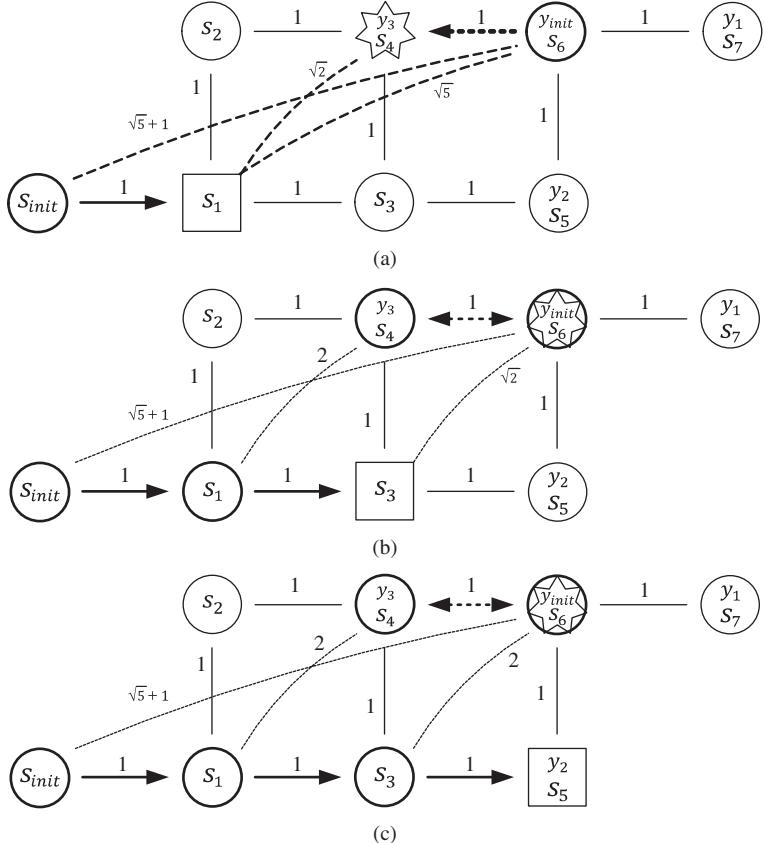


Figure 2.35 Example of the expanded states and the resulting paths of the searcher and the target for the MTS algorithm. (a) The movements of the searcher and the target, and the updated costs after Step 1. (b) The movements of the searcher and the target, and the updated costs after Step 2. (c) The movements of the searcher and the target, and the updated costs after Step 3.

The movements of the searcher and the target, and the updated costs after Step 1, are shown in Figure 2.35a.

Step 2. Now the searcher is at state s_1 . Expanding this state, the search obtains the successor set $N(s_1) = \{s_{init}, s_2, s_3\}$. According to the graph \mathcal{G} , for the state s_{init} the estimated cost is $\tilde{c}(s_{init}, y_{curr}) = \sqrt{5} \simeq 2.24$, while for both states s_2 and s_3 the estimated costs are equal to the actual costs, that is, $\tilde{c}(s_2, y_{curr}) = 1$ and $\tilde{c}(s_3, y_{curr}) = 1$. The cost $\tilde{c}(s_{init}, y_{curr})$ is greater than the costs of states s_2 and s_3 , while the costs of states s_2 and s_3 are equal. Thus, the searcher chooses one of the states s_2 and s_3 randomly. Assume that by random choice the searcher chooses the state s_3 , so $s_{next} = s_3$. The estimated costs are updated as follows. After the first step, the searcher is located at the state $s_{curr} = s_1$ and the target is located at the state $y_{curr} = s_4$, and the estimated cost $\tilde{c}(s_{curr}, y_{curr}) = \sqrt{2} \simeq 1.41$. For the chosen next state of the searcher, $s_{next} = s_3$, thus $\tilde{c}(s_{next}, y_{curr}) = 1$. Hence, $\tilde{c}(s_{next}, y_{curr}) + 1 = 1 + 1 = 2 >$

$\tilde{c}(s_{curr}, y_{curr}) = \sqrt{2} \simeq 1.41$, and the estimated cost of the current state s_{curr} is updated as $\tilde{c}(s_{curr}, y_{curr}) \leftarrow \tilde{c}(s_{next}, y_{curr}) + 1 = 1 + 1 = 2$. Note that this value is equal to the actual cost between the states $s_{curr} = s_1$ and $y_{curr} = s_4$. After updating, the searcher moves to the state $s_{curr} = s_{next} = s_3$.

The target is at the state $y_{curr} = s_4$. The successors set of this state with respect to the target's set of states $S_{tar} = \{s_4, s_5, s_6, s_7\}$ is $N(y_{curr}) = \{s_6\}$. Assume that the target chooses to move to this state $y_{next} = s_6$, which is equivalent to its initial state. After the updating and movement at the searcher's turn, $\tilde{c}(s_{curr}, y_{curr}) = 2$, while $\tilde{c}(s_{curr}, y_{next}) = \sqrt{2} \simeq 1.41$. Thus $\tilde{c}(s_{curr}, y_{curr}) = 2 > \tilde{c}(s_{curr}, y_{next}) - 1 \simeq 1.41 - 1 = 0.41$, and the value of the estimated cost $\tilde{c}(s_{curr}, y_{curr})$ is not changed, in correspondence to its equivalence to the actual cost. After updating, the target moves to the state $y_{curr} = y_{init} = s_6$.

The movements of the searcher and the target, and the updated costs after Step 2, are shown in Figure 2.35b.

Step 3. At the beginning of this step, the search is at the state $s_{curr} = s_3$ and the target is at the state $y_{curr} = s_6$, so $\tilde{c}(s_{curr}, y_{curr}) = \sqrt{2} \simeq 1.41$. The searcher acts as before at the previous step. Expansion of state s_3 results in the successor set $N(s_3) = \{s_1, s_4, s_5\}$, and the estimated costs of these states are $\tilde{c}(s_1, y_{curr}) = \sqrt{2} \simeq 1.41$, $\tilde{c}(s_4, y_{curr}) = 1$, and $\tilde{c}(s_5, y_{curr}) = 1$. The costs of states s_4 and s_5 are equal, The cost of state s_1 is greater than the costs of states s_4 and s_5 , while the costs of the last two states are equal. Thus, the searcher chooses one of the states s_4 and s_5 randomly. Assume that by random choice the searcher chooses the state $s_{next} = s_5$. Similar to the previous step, since $\tilde{c}(s_{next}, y_{curr}) + 1 = 1 + 1 = 2 > \tilde{c}(s_{curr}, y_{curr}) = \sqrt{2} \simeq 1.41$, the estimated cost of the current state s_{curr} is updated as $\tilde{c}(s_{curr}, y_{curr}) \leftarrow \tilde{c}(s_{next}, y_{curr}) + 1 = 1 + 1 = 2$, which is equal to the actual cost between the states $s_{curr} = s_1$ and $y_{curr} = s_4$. After updating, the searcher moves to the state $s_{curr} = s_{next} = s_5$.

Assume that the target chooses to stay at its current state $y_{curr} = s_6$, so its next state is $y_{next} = y_{curr} = s_6$. Then, $\tilde{c}(s_{curr}, y_{curr}) = 2 > \tilde{c}(s_{curr}, y_{next}) - 1 = 2 - 1 = 1$, and the cost $\tilde{c}(s_{curr}, y_{curr})$, which is equal to the actual cost, is not changed. At the end of the turn, the target's current state is set to $y_{curr} = y_{next} = s_6$.

The movements of the searcher and the target, and the updated costs after Step 3, are shown in Figure 2.35c.

The three steps above represent the possible cases of movement and updating, and at further steps of the algorithm act similarly up to finding the target.

In addition to the movements of the searcher and the target, the steps of the MTS algorithm demonstrate that, in full correspondence with Lemma 2.13 and Theorem 2.14, during the search the estimated costs are updated and become closer to the actual costs. ■

Finally, let us briefly consider the other algorithm of search for a moving target that, in contrast to the MTS algorithm (Algorithm 2.14), is constructed as a direct generalization of the A* algorithm (Algorithm 2.11). This algorithm was developed by Sun *et al.* in 2009 and is called the Fringe-Retrieving A* (FRA*) algorithm [97].

The FRA* algorithm [97] acts on the graph $\mathcal{G} = (S, E)$ and at each step applies the A* algorithm (Algorithm 2.11). It implements the evaluation cost function (Equation 2.47) for the costs $c(s)$, $s \in S$, and the admissibility assumption (Equation 2.48) regarding actual costs $\tilde{c}^*(s)$ and estimated costs $\tilde{c}(s)$. However, in contrast to the A* algorithm, in the costs

$\tilde{c}^*(s) = \tilde{c}^*(a^*[s, s_{fin}])$ and $\tilde{c}(s) = \tilde{c}(a^*[s, s_{fin}])$ the initial state s_{init} and the final state s_{fin} vary according to the searcher's and target's movements. So, similar to the MTS algorithm (Algorithm 2.14), in the FRA* algorithm it is assumed that the final state s_{fin} represents the current state of the target, that is, $s_{fin} = y_{curr}$.

In addition to the functions that were defined for the BF* algorithm (Procedure 2.5) and the A* algorithm (Algorithm 2.11), the FRA* algorithm implements the function $parent(s)$, $s \in S$, that results in the predecessor of the state in the path. In particular, if $a[s_{init}, y_{curr}] = a[s_{init}, s] \circ a[s, s_{foll}] \circ a[s_{foll}, y_{curr}]$, then $parent(s_{foll})$ results in state s . Since both s and s_{foll} are in the path $a[s_{init}, y_{curr}]$, state s_{foll} is one of the successors of state s in the graph G , that is, $s_{foll} \in N(s) = expand(s)$. As in the BF* (Procedure 2.5) and A* (Algorithm 2.11) algorithms, the FRA* algorithm maintains the list *OpenList* of the states, which are marked as open, and the list *ClosedList* of the states, which are marked as closed. The actions of the FRA* algorithm are based on the updating and reordering of the lists *OpenList* and *ClosedList* with respect to the results obtained in the previous step.

Recall that $\bar{c}(s) = \bar{c}(a[s_{init}, s])$ stands for the evaluated cost of the path from the initial state s_{init} to state s as found up to the current step of the A* algorithm, and $\tilde{c}(s) = \tilde{c}(a^*[s, s_{fin}])$ stands for the estimated evaluated cost of the optimal path from state s to the final state s_{fin} . In addition, recall that $\tilde{c}^*(s) = \tilde{c}^*(a^*[s, s_{fin}])$ denotes the actual cost of the optimal path $a^*[s, s_{fin}]$ from state s to the final state s_{fin} . The updating of the lists in the FRA* algorithm is based on the properties of the A* algorithm, which, for convenience, are summarized as follows:

1. For every state $s \in ClosedList$ it follows that:

1.1 If $s \neq s_{init}$, then

$parent(s) \in ClosedList$ and $\bar{c}(s) = \bar{c}(parent(s)) + c(parent(s), s)$;

1.2 $\bar{c}(s) = \bar{c}(s_{init}) + \tilde{c}^*(s) = \bar{c}(a[s_{init}, s]) + \tilde{c}^*(a^*[s, s_{fin}])$.

2. The *OpenList* contains the following states:

2.1 If *ClosedList* is empty, then *OpenList* includes only initial state s_{init} .

Otherwise, there exists a state $s \in OpenList$ such that one of its successors is in the *ClosedList*, that is, for the successor set $N(s) = expand(s)$ it follows that $N(s) \cap ClosedList \neq \emptyset$;

2.2 For every state $s \in OpenList$ it follows that:

2.2.1. $parent(s) \in ClosedList$ and $\bar{c}(parent(s)) + c(parent(s), s)$ is minimal over the successor set $N(parent(s))$;

2.2.2. $\bar{c}(s) = \bar{c}(parent(s)) + c(parent(s), s)$.

3. The A* algorithm terminates while either $OpenList = \emptyset$ and no path $a[s_{init}, s_{fin}]$ exists, or the optimal path $a^*[s_{init}, s_{fin}]$ is found.

By using the properties of the A* algorithm, the FRA* algorithm acts as follows [97]. Let the searcher start with the state $s_{init} \in S$ and let the target's state be $s_{fin} \equiv y_{curr} \in S_{tar} \subseteq S$. Assume that the initial costs $c_0(s)$, $s \in S$, are admissible with respect to the target's state y_{curr} . First, the FRA* algorithm implements the A* algorithm and finds an

optimal path $a^*[s_{init}, y_{curr}]$, where $y_{curr} \equiv s_{fin}$. Since the A* algorithm acts there, the above-listed properties 1 and 2 hold. After obtaining the path $a^*[s_{init}, y_{curr}]$, the searcher moves along this path. If the target moves over the states y such that $y \in a^*[s_{init}, y_{curr}]$, then the searcher continues moving over the states $s \in a^*[s_{init}, y_{curr}]$. Otherwise, if at a certain step the target moves to the state y_{new} such that $y_{new} \notin a^*[s_{init}, y_{curr}]$, then the last searcher's state $s_{last} \in a^*[s_{init}, y_{curr}]$ is specified as the new initial state $s_{init} \leftarrow s_{last}$ and the new target's state y_{new} is specified as its current state $y_{curr} \leftarrow y_{new}$. Then the searcher updates the lists *OpenList* and *ClosedList* to guarantee that properties 1 and 2 hold. After updating, the A* algorithm is implemented over updated lists *OpenList* and *ClosedList* and updated states s_{init} and y_{curr} . A more formal outline of the FRA* algorithm is as follows.

Algorithm 2.15 (FRA* algorithm) [97]. Given graph $\mathcal{G} = (S, E)$, $s_{init} \in S$ and $y_{init} \in S_{tar} \subseteq S$, cost function c defined by Equation 2.47, and admissible initial costs $c_0(s)$ do:

1. Define a list *OpenList* and a list *ClosedList* and set both as empty lists.
2. Start with the searcher's state $s_{curr} = s_{init}$ and target's state $y_{curr} = y_{init}$.
3. While $s_{curr} \neq y_{curr}$ do:
 - 3.1. Apply the A* algorithm (Algorithm 2.11) over the states $s_{init} = s_{curr}$ and $s_{fin} = y_{curr}$ and the lists *OpenList* and *ClosedList* (for formal actions see Procedure 2.5 (BF* algorithm) starting from Line 2).
 - 3.2. If the A* algorithm terminates with failure, then terminate and inform that a path from the searcher's state s_{curr} and the target's state y_{curr} does not exist. Otherwise, the A* algorithm returns an optimal path $a^*[s_{init}, s_{fin}]$.
 - 3.3. While $y_{curr} \in \text{ClosedList}$ do: The searcher follows the path $a^*[s_{init}, s_{fin}]$.
 - 3.3.1. The searcher selects the state $s_{foll} \in a^*[s_{init}, s_{fin}]$ that follows after the state s_{curr} in the path $a^*[s_{init}, s_{fin}]$, that is, a state s_{foll} such that $s_{curr} = \text{parent}(s_{foll})$.
 - 3.3.2. The target chooses its next state y_{next} in the set S_{tar} , that is, $y_{next} \in N(y_{curr}) = \text{expand}(y_{curr}) \cap S_{tar}$.
 - 3.3.3. If $s_{foll} = y_{next}$, then terminate and inform that the target is found in the state y_{next} .
Else: Set $s_{curr} = s_{foll}$ (the searcher moves to the state s_{foll}) and set $y_{curr} = y_{next}$ (the target moves to the state y_{next}).
 - 3.4. Reorder the lists *OpenList* and *ClosedList* using initial state $s_{init} = s_{curr}$ and final state $s_{fin} = y_{curr}$ so that properties 1 and 2 hold.
4. Return s_{curr} . ■

The outline above specifies general actions of the FRA* algorithm; a particular description of the algorithm with C-style pseudo-code is presented in Sun *et al.*'s original paper [97].

The FRA* algorithm is based on iterative implementation of the A* algorithm. It has the same properties and always terminates either with a failure or with finding the target.

At each iteration of the FRA* algorithm, the obtained path is optimal; however, a path of the searcher, which is obtained from the beginning of the search up to termination, is not optimal.

Note that the FRA* algorithm of search for a moving target at each iteration applies the A* algorithm which defines search for a static target. Thus, in a certain sense, the FRA* algorithm acts according to the same principles as Brown's algorithm (Algorithm 2.2), in which the search and screening of the moving target is implemented on the basis of iterative application of Stone's algorithm (Algorithm 2.1) of search for a static target.

The MTS and FRA* algorithms (similar to the A* and LRTA* algorithms) do not apply any probabilistic information about the target location and are based purely on a pessimistic assumption regarding the target's steps. Some progress in the application of probabilistic methods has been achieved by the application of dynamic programming principles [98]. Later, we will consider such a variant of the LRTA* algorithm, called the Min–Max LRTA* [78, 99], that implements min–max strategies for a search in non-deterministic domains.

2.4 Summary

This chapter presented three approaches to the search problem: search and screening, group testing, and search over graphs. These known methods are associated to each other, although they use different notations and definitions of the search problem. The search and screening approach is directly related to the physical search for targets in two-dimensional geographic space. The group-testing approach relies on additional assumptions regarding the ability to search simultaneously a subset of points in the search space. It represents a clear isomorphism between search for a static target and the coding of an item – the codeword of an item represents the results of the search stages over subsets of points. Such an isomorphism is less trivial for the case of search for a moving target. Finally, search over graphs provides a general scheme for path planning that is commonly used in the AI literature.

In the next chapter we propose a unified framework that bridges the gap between the approaches mentioned above. It can consider a physical or a semantic search problem; it uses group testing but applies it also to a moving target; and it uses an AI framework of learning and searching over a graph.

References

1. B.O. Koopman (1946) Search and Screening. Operation Evaluation Research Group Report, 56, Center for Naval Analysis, Rosslyn, VA; See also: B. O. Koopman (1956–1957). The theory of search, I–III. *Operations Research*, 1956, **4**, 324–346; 1956, **4**, 503–531; 1957, **5**, 613–626.
2. B. O. Koopman (1979). Search and its optimization. *The American Mathematical Monthly*, **86** (7), 527–540.
3. J.R. Frost and L.D. Stone (2001) Review of Search Theory: Advances and Applications to Search and Rescue Decision Support, US Coast Guard Research and Development Center, Groton.
4. D.C. Cooper, J.R. Frost, and R. Quincy Robe (2003) Compatibility of Land SAR Procedures with Search Theory, US Department of Homeland Security, Washington, DC.
5. F. Dambreville and J.-P. Le Cadre (1999) Detection of a Markovian Target with Optimization of the Search Efforts under Generalized Linear Constraints. Technical Report 1278, Institut de Recherche en Informatique et Systèmes Aléatoires, Rennes Cedex.

6. B. O. Koopman (1980). *Search and Screening: General Principles with Historical Applications*. Pergamon Press: New York.
7. L.D. Stone (1975). *Theory of Optimal Search*. Academic Press: New York, San Francisco, CA, London and (1989). 2nd edn, INFORMS, Linthicom, MD.
8. O. Hellman (1985). *Introduction to the Optimal Search Theory*. Nauka: Moscow (In Russian).
9. K. Chelst (1978). An algorithm for deploying a crime directed (tactical) patrol force. *Management Science*, **24** (12), 1314–1327.
10. L.D. Stone (1983). The process of search planning: current approaches and continuing problems. *Operations Research*, **31** (2), 207–233.
11. A.P. Ciervo (1976) Search for Moving Targets. PSR Report 619, Pacific-Sierra Research Corporation, Santa-Monica, CA, for Office of Naval Research, Arlington, VA.
12. S. S. Brown (1980). Optimal search for a moving target in discrete time and space. *Operations Research*, **28** (6), 1275–1289.
13. A.R. Washburn (1983) Search for a moving target: the FAB algorithm. *Operations Research*, **31** (4), 739–751.
14. A.R. Washburn (1998) Branch and bound methods for a search problem. *Naval Research Logistics*, **45**, 243–257.
15. A.R. Washburn (1982). *Search and Detection*. ORSA Books: Arlington, VA.
16. H. M. Taha (1997). *Operations Research: An Introduction*. Prentice Hall: Upper Saddle River, NJ.
17. W.L. Winston (2004). *Operations research Applications and Algorithms*. Thomson Learning: Belmont, CA.
18. D. J. Luenberger (1969). *Optimization by Vector Space Methods*. John Wiley & Sons, Inc.: New York.
19. R. T. Rockafellar (1972). *Convex Analysis*. Princeton University Press: Princeton, NJ.
20. A. Stenz (1994) Optimal and efficient path planning for partially-known environments. Proceeding of IEEE International Conference on Robotics and Automation, San Diego, CA, May 8–13, 1994, Vol. 4, pp. 3310–3317.
21. O. Hellman (1972). On the optimal search for a randomly moving target. *SIAM Journal on Applied Mathematics*, **22**, 545–552.
22. A.R. Washburn (1980) On a search for a moving target. *Naval Research Logistics Quarterly*, **27**, 315–322.
23. T.J. Stewart (1979). Search for a moving target when searcher motion is restricted. *Computers and Operation Research*, **6**, 129–140.
24. R. Dorfman (1943). The detection of defective members of large population. *Annals of Mathematical Statistics*, **14**, 436–440.
25. A. Sterrett (1956) An Efficient Methods for the Detection of Defective Members of Large Populations. PhD thesis. University of Pittsburg, Pittsburg.
26. A. Sterrett (1957). On the detection of defective members of large populations. *Annals of Mathematical Statistics*, **28**, 1033–1036.
27. D.-Z. Du and F.K. Hwang (1993). *Combinatorial Group Testing and its Applications*. World Scientific Publishing: Singapore.
28. I. Ben-Gal (2004). An upper bound on the weight-balanced testing procedure with multiple testers. *IIE Transactions*, **36**, 481–493.
29. D. Gupta and R. Malina (1999). Group testing in presence of classification errors. *Statistics in Medicine*, **18**, 1049–1068.

30. Y. T. Herer and T. Raz (2000). Optimal parallel inspection for finding the first nonconforming unit in a batch – an information theoretic approach. *Management Science*, **46** (6), 845–857.
31. J. O. Berger (1985). *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag: Berlin.
32. M. H. DeGroot (1970). *Optimal Statistical Decisions*. McGraw-Hill: New York.
33. A. Wald (1950). *Statistical Decision Functions*. John Wiley & Sons, Ltd: Chichester, etc.
34. R. Ahlsweide and I. Wegener (1987). *Search Problems*. John Wiley & Sons, Inc.: New York.
35. T. M. Cover and J. A. Thomas (1991). *Elements of Information Theory*. John Wiley & Sons, Inc.: New York.
36. R.S. Sutton and A.G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press: Cambridge, MA.
37. I. Ben-Gal and M. Caramanis (2002) Sequential DOE via dynamic programming, *IIE Transactions*, **34** (12), 1087–1100.
38. D. P. Bertsekas (1995). *Dynamic Programming and Optimal Control*. Athena Scientific Publishers: Boston, MA.
39. D. P. Bertsekas and S. E. Shreve (1978). *Stochastic Optimal Control: The Discrete Time Case*. Academic Press: New York.
40. S. Alpern and S. Gal (2003). *The Theory of Search Games and Rendezvous*. Kluwer Academic Publishers: New York, etc.
41. S. Gal. *Search Games*. Academic Press: New York, 1988.
42. F.R.K. Chung, J. E. Cohen and R. L. Graham (1988). Pursuit-evasion games on graphs. *Journal of Graph Theory*, **12** (2), 159–167.
43. R. Isaacs (1965). *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley & Sons, Inc.: New York, London, Sydney.
44. T. H. Cormen, C. E. Leiserson and R. L. Rivest (1990). *Introduction to Algorithms*. MIT Press: Cambridge, MA.
45. D. Knuth (1998). *Sorting and Searching. The Art of Computer Programming*, Vol. **3**, 2nd edn, Addison-Wesley: Reading, MA.
46. L. E. Graff and R. Roeloffs (1974). A group-testing procedure in the presence of test error. *Journal of the American Statistical Association*, **69** (345), 159–163.
47. M. Sobel and P. Groll (1959). Group-testing to eliminate efficiently all defectives in a binomial sample. *Bell System Technical Journal*, **38**, 1179–1252.
48. P.J. Rodriguez, F.K. Hwang, and M. Sobel (1984) Group Testing to Identify the Sample Minimum from a Discrete Uniform Distribution. SRD Research Report CENSUS/SDR/RR-84/26, Statistical Research Division, Bureau of the Census, Washington, DC.
49. F. K. Hwang (1976). An optimal nested procedure in binomial group testing. *Biometrics*, **32** (4), 939–943.
50. S. Kumar and M. Sobel (1971). Finding a single defective in binomial group-testing. *Journal of the American Statistical Association*, **66** (336), 824–828.
51. N. Mehravari (1986). Generalized binary binomial group testing. *SIAM Journal on Algebraic and Discrete Methods*, **7** (1), 159–166.
52. N. Johnson, S. Kotz, and R. N. Rodrigues (1989). Dorfman-sterrett screening (group-testing) schemes and the effects of faulty inspection. *Communications in Statistics – Theory and Methods*, **18** (4), 1469–1484.
53. N. Johnson, S. Kotz, and X. Wu (1991). *Inspection Errors for Attributes in Quality Control*. Chapman & Hall: London.
54. S. Kotz, N. L. Johnson, and N. Gunther (1988). Inspection errors in graff-roeloffs modification of hierarchical dorfman screening procedures. *Communications in Statistics – Theory and Methods*, **17** (1), 139–168.

55. L.G. Kraft, Jr., (1949) A device for quantizing, grouping, and coding amplitude-modulated pulses. MSc thesis. MIT, Cambridge, MA.
56. B. Mandelbrot (1954) On recurrent noise-limiting coding. *Proceedings of Symposium on Information Networks*, Polytechnic Institute of Brooklyn: New York, pp. 205–221.
57. L. Szilard (1929). Über die entropieverminderung in einem thermodynamischen system bei einsgriffen intelligenter wesen, *Zeitschrift für Physik*, **53**, 840–856; English transl.: L. Szilard (1964). On the decrease of entropy in a thermodynamic system by the intervention of intelligent beings, *Behavioral Science*, **9**, 301–310.
58. D. A. Huffman (1952). A method of the construction of minimal-redundancy codes. *Proceedings of the Institute of Radio Engineers*, **40**, 1098–1101.
59. S. Zimmerman (1959) An optimal search procedure. *American Mathematics Monthly*, **66**, 690–693.
60. S. Ganapathy and V. Rajaraman (1973). Information theory applied to the conversion of decision tables to computer programs. *Communications of the ACM*, **16** (9), 532–539.
61. R. M. Goodman and P. Smyth (1988). Decision tree design from a communication theory standpoint. *IEEE Transactions on Information Theory*, **34** (5), 979–994.
62. C. R. P. Hartmann, P. K. Varshney, K. G. Mehrotra, and C. L. Gerberich (1982). Application of information theory to the construction of efficient decision trees. *IEEE Transactions on Information Theory*, **28** (4), 565–577.
63. C.E. Shannon (1948). A mathematical theory of communication. *Bell System Technical Journal*, **27**(3), 379–423; **27**(4), 623–656.
64. W. Feller (1970). *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, Inc.: New York.
65. Y.G. Sinai (1977). *Introduction to Ergodic Theory*. Princeton University Press: Princeton, NJ.
66. J. Abrahams (1994). Parallelized Huffman and Hu-Tucker searching. *IEEE Transactions on Information Theory*, **40** (2), 508–510.
67. I. Ben-Gal, E. Kagan, and N. Shkolnik (2008) Constructing classification trees via data mining and design of experiments concepts. Proceedings of ENBIS'8, Athens, Greece, September 21–25, 2008, e-publ.
68. I. Ben-Gal, N. Shkolnik, and E. Kagan (2007) Greedy learning algorithms and their applications to decision trees. Proceedings of the 7th European Annual Conference ENBIS'07, Dortmund, September 24–26, 2007, p. 36, e-publ.
69. E. Kagan and I. Ben-Gal (2013) Moving Target Search Algorithm with Informational Distance Measures. To appear in Entropy.
70. E. W. Dejkstra (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, **1**, 269–271.
71. E.F. Moore (1959) The shortest path through a maze. Proceedings of an International Symposium on the Theory of Switching Part II, Cambridge, MA, 2–5 April, 1957, pp. 285–292.
72. D. Walden (2003) The Bellman-Ford Algorithm and ‘Distributed Bellman-Ford’, On-line, <http://www.walden-family.com/public/bf-history.pdf>.
73. L.R. Ford, Jr., (1956) Network Flow Theory. Technical Report P-923, RAND, Santa Monica, CA.
74. R. Bellman (1958) On a routing problem. *Quarterly of Applied Mathematics*, **XVI** (1), 87–90.
75. J. Pearl (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley: Reading, MA.
76. P. E. Hart, N. J. Nilsson and B. Raphael (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, **SSC-4** (2), 100–107.

77. T. Ishida and R. E. Korf (1995). Moving target search: a real-time search for changing goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17** (6), 609–619.
78. S. Koenig and R.G. Simmons (1995) Real-time search on non-deterministic domains. Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-95), pp. 1660–1667.
79. R. Dechter and J. Pearl (1985). Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, **32** (3), 505–536.
80. S. Russell and P. Norvig (2010). *Artificial Intelligence. A Modern Approach*. 3rd edn. Pearson Education Inc.: Upper Saddle River, NJ.
81. N.J. Nilsson (1971). *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill: New York.
82. D. Gelperin (1977). On the optimality of A*. *Artificial Intelligence*, **8**, 69–76.
83. H. Farreny (1999). Completeness and admissibility for general heuristic search algorithms – a theoretical study: basic concepts and proofs. *Journal of Heuristics*, **5**, 353–376.
84. I. Pohl (1977). Practical and theoretical considerations in heuristic search algorithms. In *Machine Intelligence*, Vol. **8**, E. W. Elcock, D. Michie (eds), New York: John Wiley & Sons, Inc., pp. 55–72.
85. R.E. Korf (1987) Real-time heuristic search: first results. Proceedings of AAAI'87, pp. 133–138.
86. R.E. Korf (1988) Real-time heuristic search: new results. Proceedings of AAAI'88, pp. 139–144.
87. R. E. Korf (1990). Real-time heuristic search. *Artificial Intelligence*, **42** (2–3), 189–211.
88. R.E. Korf (1985). Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, **27**, 97–109.
89. S. Sigmundarson (2006) Backtracking and Value Back-Propagation in Real-Time Search. MSc thesis. Reykjavík University, Reykjavík, Iceland.
90. M. Shimbo and T. Ishida (2003). Controlling the learning process of real-time heuristic search. *Artificial Intelligence*, **146**, 1–41.
91. V. Bulitko and G. Lee (2006). Learning in real-time search: a unifying framework. *The Journal of Artificial Intelligence Research*, **25**, 119–157.
92. V. Bulitko, N. Sturtevant, J. Lu, and T. Yau (2007). Graph abstraction in real-time heuristic search. *The Journal of Artificial Intelligence Research*, **30**, 51–100.
93. Y. Björnsson, V. Bulitko, and N. Sturtevant (2009) TBA*: time-bounded A*. Proceedings of the 21st International Joint Conference on Artificial Intelligence IJCAI'09, pp. 431–436.
94. V. Bulitko and Y. Björnsson (2009) kNN LRTA*: simple subgoaling for real-time search. Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2009, Stanford, CA, pp. 2–7.
95. S. Koenig and M. Likhachev (2006) Real-time adaptive A*. Proceedings of AAMAS'06, 2006, pp. 281–288.
96. T. Ishida and R.E. Korf (1991) Moving target search. Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-91), pp. 204–210.
97. X. Sun, W. Yeoh, and S. Koenig (2009) Efficient incremental search for moving target search. Proceedings of the 21st International Joint Conference on Artificial Intelligence IJCAI'09, pp. 615–620.
98. R. P. Loui (1983). Optimal paths in graphs with stochastic or multidimensional weights, *Communications of the ACM*, **26** (9), 670–676.
99. S. Koenig (2001). Mini-max real-time heuristic search. *Artificial Intelligence*, **129** (1–2), 165–197.

3

Models of search and decision making

In the previous chapter, we considered several types of search problems and their methods. The presented models are used for analysis and to solve different tasks, and the implemented methods are based on different techniques. In this chapter, we present general models and methods for solving the search problems and consider the methods and algorithms which implement such methods.

We start with the framework of Markov decision processes (MDPs) that supports a description of sequential decision making and provides unified solutions for a wide range of search problems. Following the MDP framework, we will consider the methods of search and screening (see Section 2.1) and of group-testing search (see Section 2.2); such a consideration is finalized by the White method of search. The formulation of the search problem in the form of MDP allows its direct consideration by use of the dynamic programming method, as initiated by Ross, and by the use of suitable learning methods. To illustrate this, we will consider the Eagle and Washburn algorithms and their successors.

As indicated at the beginning of Chapter 2, if in the problem of search for a moving target, the target is informed about the searcher actions and tries to escape from the searcher, then the search problem is considered as a pursuit-evasion game. In such a model, the search for an uninformed target can be considered as a search game against nature. In this chapter, we will present the Chung *et al.* and Hespanha *et al.* models of the pursuit-evasion game on graphs (PEGGs) that generalize the A*-type algorithms presented above (see Section 2.3) for the probabilistic search and provide the criteria for the existence of the solution.

The chapter also includes particular models and algorithms of search which illustrate the MDP framework and generalize the previously considered methods, in particular the informational methods of search. The models provide a basis for further considerations and can be implemented for a wide range of practical tasks.

3.1 Model of search based on MDP

The studies of sequential decision making under uncertainty are based on different models. As indicated in Section 1.3, one of the first models of such processes was developed by Wald [1] in the 1940s on the basis of zero-sum games theory. This model addresses the problem of hypothesis testing and estimation of the distribution parameters using a finite number of tests such that the required payoff reaches a minimum. As a result, a sequence of tests is obtained, and the decision-maker follows this sequence up to the desired result. In group-testing theory (see Section 2.2), such a model predefines parameters of the groups, which have to be checked at certain times, and the manipulations, which have to be conducted over the chosen or remaining units. In general, the tests and the manipulations, together with the testing units, are called actions, so the goal of the models is to obtain the sequence of actions which meets minimization or maximization requirements.

An application of the dynamic programming approach for obtaining the required policy gave rise to the consideration of MDPs, initiated by Bellman [2] in 1957. Following the Markov property, in this model the choice of the next action depends on the current state of the considered system and currently applied action. Nowadays, MDPs are implemented for the analysis of different systems and appear in tasks that are far from statistical decision making [3, 4]. In particular, such models are applied when the analyzed system can be in one of the states and can move from one state to another according to its inner characteristics, which are not available to the decision-maker. However, the decision-maker can observe current states of the system and apply certain actions that change the system dynamics. Then, following the applied actions and the obtained corresponding dynamics of the system, the characteristics of the system and its behavior are estimated.

Below, we concentrate on the application of MDPs to the search problem, where, as above, the actions include observations of the chosen areas and Bayesian calculations of the posterior location probabilities.

3.1.1 General definitions

Let us start with the definitions of MDPs (for brevity). In the consideration below, we mostly follow the discourses presented by Ross [5] and by White [4]. In order to obtain the most general form of the MDP, we apply a notation that differs slightly from the one used for definitions of search problems; however, the relation between the introduced and the previously considered concepts is indicated.

We denote by $\mathcal{S} = \{s_1, s_2, \dots\}$ a finite or countable set of some abstract *states*. Following the methods and algorithms of search presented above, such states are associated with the searcher positions along with the available information regarding the search process. In particular, in the search and screening algorithms (see Section 2.1) as well as in the group-testing search (see Section 2.2) each state $s \in \mathcal{S}$ corresponds to the observed area $A \subset X$ from the sample space X along with the location probabilities, while in the A*-type algorithms of search over graphs (see Section 2.3) such a set \mathcal{S} corresponds to the set S of vertices of the graph $\mathcal{G} = (S, E)$ with corresponding evaluated costs $c(s)$, $s \in \mathcal{S}$. Below, we will consider the indicated correspondences in particular. Given the set $\mathcal{S} = \{s_1, s_2, \dots\}$ of states, assume that there exists a finite or countable set $\mathcal{A} = \{a_1, a_2, \dots\}$ of *actions*. Recall that in the GOTA (Generalized Optimal Testing Algorithm) search (Algorithm 2.10;

see also Examples 2.18 and 2.19) we applied a set of actions $\{\mathbb{A}_1, \dots, \mathbb{A}_4\}$ for explicit definition of the final partition, which specifies the desired level in the search tree. In the search and screening algorithms and in the algorithms of search over graphs, the actions represent the movements of the searcher with appropriate probabilities or cost updating, and in the group-testing search they correspond to observations of the areas $A \subset X$ with probability updating.

The MDP represents the system dynamics from the viewpoint of the decision-maker and is defined as follows. Let $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \dots\}$ be a set of possible states of the observed system and let $\mathcal{A} = \{a_1, a_2, \dots\}$ be a set of actions, which can be applied by the decision-maker, while certain information regarding the dynamics of the system is not available. At each time $t = 0, 1, 2, \dots$ the decision-maker observes the state $\mathbf{s}^t \in \mathcal{S}$ of the system and conducts an action $a^t \in \mathcal{A}$. Thus, at time t , the decision-maker is informed about the state $\mathbf{s}^t = \mathbf{s}_i$ and the applied action $a^t = a_k$. Now the decision-maker predicts the state $\mathbf{s}^{t+1} \in \mathcal{S}$, at which the system will be at the next time $t + 1$.

Let $\overrightarrow{\mathbf{s}} = \langle \mathbf{s}^t, \mathbf{s}^{t-1}, \dots, \mathbf{s}^1, \mathbf{s}^0 \rangle$ be a sequence of states, which were observed by the decision-maker up to time t , and let $\overrightarrow{a} = \langle a^t, a^{t-1}, \dots, a^1, a^0 \rangle$ be a sequence of actions, which were conducted by the decision-maker at the corresponding times up to time t . The sequence of actions \overrightarrow{a} for any time is usually called the *strategy*. The joint sequence $\overrightarrow{a \circ s} = \langle a^t, \mathbf{s}^t, a^{t-1}, \mathbf{s}^{t-1}, \dots, a^1, \mathbf{s}^1, a^0, \mathbf{s}^0 \rangle$ that includes both observed states \mathbf{s}^τ and conducted actions a^τ , $\tau = 0, 1, 2, \dots, t$, up to time t is called the *history of system dynamics* and includes the decision-maker. Since certain information regarding system dynamics is not available to the decision-maker, the state \mathbf{s}^{t+1} of the system cannot be evaluated and it is determined by the use of transition probability

$$\rho_{ij}(\overrightarrow{a \circ s}) = \Pr\{\mathbf{s}^{t+1} = \mathbf{s}_j | a^t = a_k, \mathbf{s}^t = \mathbf{s}_i, a^{t-1}, \mathbf{s}^{t-1}, \dots, a^1, \mathbf{s}^1, a^0, \mathbf{s}^0\},$$

which, in general, depends on the history $\overrightarrow{a \circ s}$ of the system dynamics and specifies the probability that at the next time $t + 1$ the system will be in the state $\mathbf{s}^{t+1} = \mathbf{s}_j$, while at the current time t the decision-maker observes the state $\mathbf{s}^t = \mathbf{s}_i$ and applies action $a^t = a_k$; at the previous time $t - 1$ the state and the action are \mathbf{s}^{t-1} and a^{t-1} , and so on down to time $t = 0$.

Nevertheless, since the actions $a \in \mathcal{A}$ are not restricted and can include all required operations, it can be achieved that the transition probabilities $\rho_{ij}(a \circ s)$ do not depend on the complete history $a \circ s$, but rather on the currently observed state $\mathbf{s}^t = \mathbf{s}_i$ and applied action $a^t = a_k$, that is,

$$\begin{aligned} \rho_{ij}(a_k) &= \Pr\{\mathbf{s}^{t+1} = \mathbf{s}_j | a^t = a_k, \mathbf{s}^t = \mathbf{s}_i\} \\ &= \Pr\{\mathbf{s}^{t+1} = \mathbf{s}_j | a^t = a_k, \mathbf{s}^t = \mathbf{s}_i, a^{t-1}, \mathbf{s}^{t-1}, \dots, a^1, \mathbf{s}^1, a^0, \mathbf{s}^0\} = \rho_{ij}(a \circ s). \end{aligned} \tag{3.1}$$

Such a property represents a Markovian property of the dynamics of the system, which includes the actions of the decision-maker. Note that similar assumptions were implemented in Section 2.1.2 in considering the dynamics of the target location density.

The main problem, which is addressed by the MDPs, is specifying a rule for choosing action a^t at each time t . In general, such a rule is represented by a probability distribution over the set of actions $\mathcal{A} = \{a_1, a_2, \dots\}$ with respect to the observed state

$s^t \in \mathcal{S} = \{s_1, s_2, \dots\}$, and it is assumed that at each time $t = 0, 1, 2, \dots$ and the state s_i , a certain action $a_k \in \mathcal{A}$ is chosen with probability

$$\delta_i^t(a_k) = \Pr\{a^t = a_k | s^t = s_i\}, \quad (3.2)$$

where for any time t it follows that $0 \leq \delta_i^t(a_k) \leq 1$ for all i and k , and $\sum_{a \in \mathcal{A}} \delta_i^t(a) = 1$ for any i . Similar to the above, the sequence of rules $\delta = \langle \delta^t, \delta^{t-1}, \dots, \delta^1, \delta^0 \rangle$ is called the *policy*. In certain applications, it is specified that $\delta^t(a_k) = 1$ for some k ; then the sequence $\overrightarrow{\delta}$ of rules unambiguously defines the strategy \overrightarrow{s} , and the terms ‘strategy’ and ‘policy’ are used interchangeably.

Let $\overrightarrow{\delta}$ be a policy that is applied by the decision-maker. To characterize the policy $\overrightarrow{\delta}$, assume that for any state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, a non-negative *reward* $R(s, a) \in [0, \infty]$ is defined that is obtained by the decision-maker, while after observation of the state $s \in \mathcal{S}$ the action $a \in \mathcal{A}$ is chosen. Since each element of the policy $\overrightarrow{\delta}$ is the probability of choosing action a given a state s , reward $R(s, a)$ is a random variable, which obtains its values according to the probabilities specified by the policy $\overrightarrow{\delta}$. Hence, the chosen policy $\overrightarrow{\delta}$ can be characterized by an *expected average reward* $E(R|\overrightarrow{\delta})$, and the goal of the decision-maker is to specify such a policy $\overrightarrow{\delta}^*$ that the expected average reward $E(R|\overrightarrow{\delta}^*)$ is maximal over all possible policies $\overrightarrow{\delta}$.

The introduced concepts for finite set of states $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ and set of actions $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$ are illustrated in Figure 3.1.

In the figure, it is assumed at time $t - 1$ the decision-maker observes that the system is in the state $s^t = s_i$. Then, there are m conditional probabilities $\delta_i^t(a_1), \delta_i^t(a_2), \dots, \delta_i^t(a_m)$ for choosing the corresponding actions a_1, a_2, \dots, a_m . Let the chosen action be a_1 . Then, for n possible states s_1, s_2, \dots, s_n there are n conditional probabilities $\rho_{i1}(a_1), \rho_{i2}(a_1), \dots, \rho_{in}(a_1)$, which specify the chance that the system will move to the corresponding state. Assume that the decision-maker observes that the system is in the state s_n , that is, $s^{t+1} = s_n$. Then, similar to the above, there are m conditional probabilities

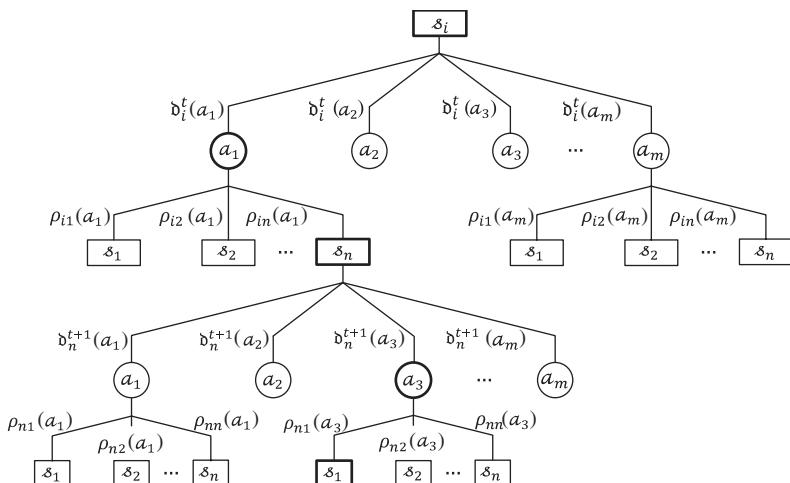


Figure 3.1 Example of the decision tree for MDP.

$\mathfrak{d}_n^t(a_1), \mathfrak{d}_n^t(a_2), \dots, \mathfrak{d}_n^t(a_m)$ for choosing the actions a_1, a_2, \dots, a_m . Assume that the decision-maker applies the action a_3 . Then the next state of the system will be chosen according to n conditional probabilities $\rho_{i1}(a_3), \rho_{i2}(a_3), \dots, \rho_{in}(a_3)$. In the figure, it is assumed that the decision-maker observes that the system moves to the state s_2 , and from this point the process continues. The part of the realized history of system dynamics for the considered two times is $\langle \dots, s_1, s_n, a_1, s_i, \dots \rangle$, and the expected average reward for the chosen two actions and two observed states is $E(R|\overrightarrow{\mathfrak{d}}) = \frac{1}{2}(R(s_i, a_1) + R(s_n, a_3))$.

There are a number of methods for finding the policy $\overrightarrow{\mathfrak{d}}^*$, which maximizes the expected average reward $E(R|\overrightarrow{\mathfrak{d}}^*)$. Let us consider the variant of constructing a policy $\overrightarrow{\mathfrak{d}}$ with infinite time [5]; the other methods will be considered below along with the consideration of certain search algorithms.

Assume that the set of states $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ and the set of actions $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$ are finite, and let expected average reward $E(R|\overrightarrow{\mathfrak{d}})$ be defined by the following limit:

$$E(R|\overrightarrow{\mathfrak{d}}) = \lim_{t \rightarrow \infty} \frac{1}{t} E \left(\sum_{\tau=0}^t R(s^\tau, a^\tau) \right), \quad (3.3)$$

where $E(\sum_{\tau=0}^t R(s^\tau, a^\tau))$ is an expected reward at time t , $t = 0, 1, 2, \dots$, and $R(s^\tau, a^\tau)$ is an immediate reward at time τ , $\tau = 0, 1, \dots, t$, while the system is in the state s^τ and the decision-maker chooses the action a^τ . Transition probabilities (Equation 3.1) with respect to the policy $\overrightarrow{\mathfrak{d}}$, which consists of the probabilities (Equation 3.2), are defined as follows [5]:

$$\rho_{ij}(\overrightarrow{\mathfrak{d}}) = \Pr\{s^{t+1} = s_j | s^t = s_i, \overrightarrow{\mathfrak{d}}\} = \sum_{k=1}^m \rho_{ij}(a_k) \mathfrak{d}_i^t(a_k). \quad (3.4)$$

Then, given policy $\overrightarrow{\mathfrak{d}}$, the probabilities (Equation 3.4) define a Markov chain over the set of states \mathcal{S} . Recall that the state $s \in \mathcal{S}$ is *aperiodic* if the process can return to this state at any arbitrary time, and the state s is *positive recurrent* if an expected time between two passes over the state s is finite. Aperiodic positive recurrent states are called *ergodic* states, and the Markov chain, in which all states are ergodic, is called the *ergodic* chain.

Assume that, given a policy $\overrightarrow{\mathfrak{d}}$, the Markov chain defined by the transition probabilities (Equation 3.4) is ergodic. Then, for each state s_i , $i = 1, 2, \dots, n$, and action a_k , $k = 1, 2, \dots, m$, there exists a *limiting or steady state probability*

$$\pi_{ik}(\overrightarrow{\mathfrak{d}}) = \pi(s_i, a_k | \overrightarrow{\mathfrak{d}}) = \lim_{t \rightarrow \infty} \Pr\{s^t = s_i, a^t = a_k | \overrightarrow{\mathfrak{d}}\}, \quad (3.5)$$

where it follows that $0 \leq \pi_{ik}(\overrightarrow{\mathfrak{d}}) \leq 1$ for all i and k , and $\sum_{i=1}^n \sum_{k=1}^m \pi_{ik}(\overrightarrow{\mathfrak{d}}) = 1$. For any $i = 1, 2, \dots, n$ and $k = 1, 2, \dots, m$, the value $\pi_{ik}(\overrightarrow{\mathfrak{d}})$ specifies a probability that, while the decision-maker implements the policy $\overrightarrow{\mathfrak{d}}$, the process will be in state s_i and the action a_k will be chosen.

Next, we need the following general statement regarding the limiting probabilities of the ergodic Markov chain.

Lemma 3.1 [5, 6]. *If ρ_{ij} are transition probabilities of an ergodic Markov chain such that any state $j = 1, 2, \dots$ can be reached from any other state $i = 1, 2, \dots$, then the limiting probability $\pi_j = \lim_{t \rightarrow \infty} (\rho_{ij})^t$ of the state j , where $(\rho_{ij})^t$ stands for the t th power of ρ_{ij} , is*

defined as a unique solution of the equality $\pi_j = \sum_i \pi_i \rho_{ij}$ such that $\sum_i \pi_i = 1$ and $\pi_i \geq 0$ for each $i = 1, 2, \dots$.

The Markov chain, in which any state can be reached from any other state, is called *irreducible*. From Lemma 3.1, it follows that for the considered Markov chain with the transition probabilities $\rho_{ij}(\vec{\delta})$, the next lemma holds.

Lemma 3.2 [5]. *For any policy $\vec{\delta}$, it follows that for each $j = 1, 2, \dots, n$*

$$\sum_{k=1}^m \pi_{jk}(\vec{\delta}) = \sum_{i=1}^n \sum_{k=1}^m \pi_{ik}(\vec{\delta}) \rho_{ij}(a_k). \quad (3.6)$$

Proof. To verify the statement of the lemma it is enough to note that since the sets \mathcal{S} and \mathcal{A} are finite, the ergodic Markov chain with transition probabilities $\rho_{ij}(\vec{\delta})$ is irreducible. Then, the lemma is the immediate consequence of Lemma 3.1. ■

The lemma specifies that for any policy $\vec{\delta}$ the steady state probabilities $\pi_{ik}(\vec{\delta})$, $i = 1, 2, \dots, n$, $k = 1, 2, \dots, m$, such that $\pi_{ik}(\vec{\delta}) \geq 0$ and $\sum_{i=1}^n \sum_{k=1}^m \pi_{ik}(\vec{\delta}) = 1$, are defined by Equation 3.6. Now let us demonstrate the backward statement.

Lemma 3.3 [5]. *Given the set of states $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ and the set of actions $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$, for any steady state probabilities π_{ik} such that $\pi_{ik} \geq 0$, $i = 1, 2, \dots, n$, $k = 1, 2, \dots, m$, and $\sum_{i=1}^n \sum_{k=1}^m \pi_{ik} = 1$, there exists a policy $\vec{\delta}$ such that*

$$\sum_{k=1}^m \pi_{jk} = \sum_{i=1}^n \sum_{k=1}^m \pi_{ik} \rho_{ij}(a_k).$$

Proof. Given steady state probabilities π_{ik} , assume that the policy $\vec{\delta}$ consists of the rules $\delta_j^t(a_k) = \Pr\{a^t = a_k | s^t = s_j\} = \pi_{jk} / \sum_{k=1}^m \pi_{jk}$, $t = 0, 1, 2, \dots$, and let $p_{ik}(\vec{\delta}) = \lim_{t \rightarrow \infty} \Pr\{s^t = s_i, a^t = a_k | \vec{\delta}\}$ be steady state probabilities for the policy $\vec{\delta}$. We need to demonstrate that $p_{ik}(\vec{\delta}) = \pi_{ik}$, $i = 1, 2, \dots, n$, $k = 1, 2, \dots, m$.

Since $p_{ik}(\vec{\delta})$ are limiting probabilities of the irreducible ergodic Markov chain, from Lemma 3.1 it follows that $\sum_{i=1}^n \sum_{k=1}^m p_{ik}(\vec{\delta}) = 1$ and

$$\begin{aligned} p_{jk}(\vec{\delta}) &= \sum_{i=1}^n \sum_{l=1}^m p_{il}(\vec{\delta}) \Pr\{s^{t+1} = s_j, a^{t+1} = a_k | s^t = s_i, a^t = a_l\} \\ &= \sum_{i=1}^n \sum_{l=1}^m p_{il}(\vec{\delta}) \rho_{ij}(a_l) \delta_j^t(a_k). \end{aligned}$$

Then, by use of the equality $\delta_j^t(a_k) = \pi_{jk} / \sum_{k=1}^m \pi_{jk}$, one obtains that the probabilities $p_{ik}(\vec{\delta})$, $i = 1, 2, \dots, n$, $k = 1, 2, \dots, m$, are defined by the unique solution of the equality $p_{jk}(\vec{\delta}) = \sum_{i=1}^n \sum_{l=1}^m p_{il}(\vec{\delta}) \rho_{ij}(a_l) \pi_{jk} / \sum_{k=1}^m \pi_{jk}$ such that $\sum_{i=1}^n \sum_{k=1}^m p_{ik}(\vec{\delta}) = 1$ and $p_{ik}(\vec{\delta}) \geq 0$ for all $i = 1, 2, \dots, n$ and $k = 1, 2, \dots, m$.

Thus, because of the uniqueness of the solution, the equality $p_{ik}(\vec{\delta}) = \pi_{ik}$ will be obtained by demonstrating that for the probabilities π_{ik} it also holds true that $\pi_{jk} = \sum_{i=1}^n \sum_{l=1}^m \pi_{il} \rho_{ij}(a_l) \pi_{jk} / \sum_{k=1}^m \pi_{jk}$ while $\sum_{i=1}^n \sum_{k=1}^m \pi_{ik} = 1$ and $\pi_{ik} \geq 0$, for all

$i = 1, 2, \dots, n$ and $k = 1, 2, \dots, m$. The last two requirements are conditioned by the lemma, while summing over k on both sides of the equality gives

$$\begin{aligned}\sum_{k=1}^m \pi_{jk} &= \sum_{k=1}^m \sum_{i=1}^n \sum_{l=1}^m \pi_{il} \rho_{ij}(a_l) \frac{\pi_{jk}}{\sum_{k=1}^m \pi_{jk}} \\ &= \sum_{k=1}^m \frac{\pi_{jk}}{\sum_{k=1}^m \pi_{jk}} \sum_{i=1}^n \sum_{l=1}^m \pi_{il} \rho_{ij}(a_l) \\ &= \sum_{i=1}^n \sum_{l=1}^m \pi_{il} \rho_{ij}(a_l)\end{aligned}$$

as required. ■

Now let us return to the problem of maximization of the expected average reward $E(R|\vec{\delta})$, which is defined by Equation 3.3. Using steady state probabilities $\pi_{ik}(\vec{\delta})$, $i = 1, 2, \dots, n$, $k = 1, 2, \dots, m$, this reward is defined as follows [5]:

$$E(R|\vec{\delta}) = \sum_{i=1}^n \sum_{k=1}^m \pi_{ik}(\vec{\delta}) R(s_i, a_k). \quad (3.7)$$

Hence, the problem of specifying the rules $\delta_i^t(a_k)$ for choosing actions a^t and resulting policy $\vec{\delta} = (\delta^t, \delta^{t-1}, \dots, \delta^1, \delta^0)$ obtains the form of a maximization problem, in which it is required to find a policy $\vec{\delta}$ such that

$$\sum_{i=1}^n \sum_{k=1}^m \pi_{ik}(\vec{\delta}) R(s_i, a_k) \rightarrow \max, \quad (3.8)$$

subject to $\pi_{ik}(\vec{\delta}) \geq 0$, for all $i = 1, 2, \dots, n$ and $k = 1, 2, \dots, m$,

$$\begin{aligned}\sum_{k=1}^m \pi_{jk}(\vec{\delta}) &= \sum_{i=1}^n \sum_{k=1}^m \pi_{ik}(\vec{\delta}) \rho_{ij}(a_k) \text{ for each } j = 1, 2, \dots, n, \text{ and} \\ \sum_{i=1}^n \sum_{k=1}^m \pi_{ik}(\vec{\delta}) &= 1.\end{aligned}$$

The maximization problem (3.8) can be solved by any method of linear programming optimization (see, e.g., [7, 8]) and, as follows from Lemma 3.3, if π_{ik}^* are the probabilities that provide the solution of Equation 3.8, then the optimal policy $\vec{\delta}^*$ consists of the rules $\delta_i^*(a_k) = \pi_{ik}^*/\sum_{k=1}^m \pi_{ik}^*$.

This method for formulating the problem of finding an optimal policy $\vec{\delta}^*$ in the form of a linear programming problem provides a simple method for the analysis of models defined by MDP. Notice that the method implements an infinite time and is based on the existence of limiting probabilities $\pi_{ik}(\vec{\delta})$, $i = 1, 2, \dots, n$, $k = 1, 2, \dots, m$, and limiting expected average reward $E(R|\vec{\delta})$. A similar assumption was implemented in the algorithms of search and screening (see Section 2.1) and implied in the search over graphs (see Section 2.3), especially in search for a moving target. However, in search for a static target, as well as in the practical tasks of search, where the time is finite, other methods are applied. In the next sections, we will present such methods along with consideration of certain search models, while below we start with the formulation of search in the form of MDP and present informational decision-making criteria.

3.1.2 Search with probabilistic and informational decision rules

In the previous section, a general description of MDP and a simple method of its analysis for an infinite time horizon were presented. Let us apply the MDP model for the search process. Below, we start with the MDP model of search as it appears in the search and screening algorithm (see Section 2.1.1) and in group-testing search (see Section 2.2), and then specify the relation between this model and the other search methods.

Recall that in the search procedures and algorithms, which are considered in Chapter 2, the searcher observes the chosen observed area or state, and then, with respect to the search problem, updates the corresponding location probabilities or evaluated costs. Such an observation implies that the choice of a certain action $a^t \in \mathcal{A}$, which is implemented in MDPs, is similar to the choice of the observed area $A^t \in \mathcal{X}$ as used in the group-testing search algorithms (see, e.g., Procedure 2.1).

As in previous considerations, let $X = \{x_1, x_2, \dots, x_n\}$ be a sample space which consists of possible target locations x_i , $i = 1, \dots, n$, and let $\mathcal{X} = 2^X \setminus \emptyset$ be a search space that includes all possible observed areas $A \subset X$. Similar to Section 2.1.1, assume that the target's movement is governed by a Markov process with transition probability matrix $\rho = ||\rho_{ij}||_{n \times n}$, where $\rho_{ij} = \Pr\{x^{t+1} = x_j | x^t = x_i\}$ and $\sum_{j=1}^n \rho_{ij} = 1$ for each $i = 1, 2, \dots, n$. For any time $t = 0, 1, 2, \dots$ over the sample space X the following probabilities are defined:

- location probabilities $p^t(x_i) = \Pr\{x^t = x_i\}$, $0 \leq p^t(x_i) \leq 1$, and $\sum_{i=1}^n p^t(x_i) = 1$;
- observed location probabilities $\bar{p}^t(x_i) = \Pr\{x^t = x_i | z^t\}$, where $z^t = z(A^t, x^t)$ is a result of observation of the observed area A^t while the target is located at the point $x^t \in X$;
- estimated location probabilities $\tilde{p}^t(x_i) = \sum_{j=1}^n \rho_{ij} \bar{p}^t(x_j)$, $i = 1, 2, \dots, n$.

In addition, following Section 2.2.4, for each of the indicated probabilities it is specified that the probability of the observed area $A \subset X$ is given by the sum $p(A) = \sum_{x \in A} p(x)$.

As indicated in Section 2.1.1, the searcher starts with the estimated location probabilities $\tilde{p}^{t=0}(x_i) = p^{t=0}(x_i)$, $i = 1, 2, \dots, n$. At each time $t = 0, 1, 2, \dots$, after choosing the observed area A^t and obtaining observation result $z^t = z(A^t, x^t)$, the searcher applies z^t over the probabilities $\tilde{p}^t(x_i)$ and obtains the observed probabilities $\bar{p}^t(x_i)$. Finally, the target's transition probability matrix $\rho = ||\rho_{ij}||_{n \times n}$ is applied and the estimated probabilities $\tilde{p}^{t+1}(x_i)$ for the next time $t + 1$ are calculated. Then, the goal of the searcher is to find a rule for choosing the observed areas $A^t \in \mathcal{X}$ such that the target would be detected in a minimal number of steps.

In terms of the MDP above, the presented search process is defined over the set of states \mathcal{S} such that each state $s \in \mathcal{S}$ is associated with the observed area $A \in \mathcal{X}$ along with the possible observation result z^t , that is, $s = (A, z^t)$. Since the target can occupy only the points $x \in X$, the states, which correspond to the target locations, are specified by the single-point areas $A \in \mathcal{X}$; that is, such areas that $|A| = 1$. Note that the defined set of states \mathcal{S} is not necessarily finite and depends on the definition of observation function z . However, for perfect observation with $z^t \in \{0, 1\}$, the set of states \mathcal{S} is finite; this fact will be used below.

Similarly, the searcher's actions $a \in \mathcal{A}$ are associated with observations of the chosen areas $A \in \mathcal{X}$ and the corresponding operations of updating the estimated probabilities $\tilde{p}^{t+1}(x)$, $x \in X$. In addition, the set of actions is completed with the action $a_0 = \text{terminate}$

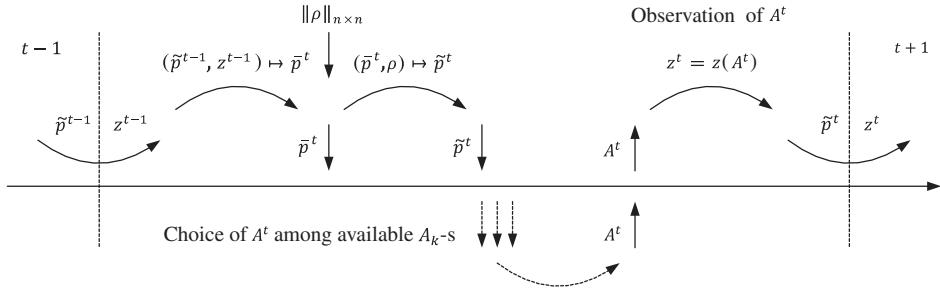


Figure 3.2 General MDP for the search procedure.

which terminates the search. For the finite sample space X and, consequently, the finite search space \mathcal{X} , the set of actions \mathcal{A} in the associated MDP is also finite.

The MDP model of search, which implements the indicated states and actions, is shown in Figure 3.2.

According to the figure, at time t the searcher starts with the estimated probabilities $\tilde{p}^{t-1}(x_i)$, $i = 1, 2, \dots, n$, and observation result z^{t-1} , which are obtained at the previous time $t - 1$. On the basis of this data, the searcher calculates observed location probabilities $\bar{p}^t(x_i)$, $i = 1, 2, \dots, n$, that usually are obtained by the Bayesian rule. The next operation, which is conducted by the searcher, requires the target's transition probability matrix $\rho = \|\rho_{ij}\|_{n \times n}$. For a static target this matrix is a unitary matrix $\rho = I_{n \times n}$, while for a moving target it is a standard stochastic matrix with $\sum_{j=1}^n \rho_{ij} = 1$ for each $i = 1, 2, \dots, n$. If the target's transition probabilities are unknown, then the searcher applies the matrix ρ with equal probabilities, which are specified according to the characteristics of the target's movement. After calculating estimated probabilities $\tilde{p}^t(x_i)$, $i = 1, 2, \dots, n$, the searcher selects an area $A^t \in \mathcal{X}$ for observation at the current time t . Finally, observation of the chosen area A^t is conducted. The obtained observation result z^t along with the previously calculated estimated probabilities $\tilde{p}^t(x_i)$, $i = 1, 2, \dots, n$, are used at the next time $t + 1$ for similar operations. In addition to the presented dataflow it is assumed that if either the size of the observed area $|A^t| = 1$ and observation result $z^t = 1$, or $|A^t| = n - 1$ and observation result $z^t = 0$, then the search terminates and results in the target location x^* that is a single point in the area $A^t = \{x^*\}$ or, correspondently, in the area $\{x^*\} = X \setminus A^t$.

In considering the search and screening methods (see Section 2.1.2), Theorem 2.1, which addresses the search for a Markovian target, was applied and the recurrent equation of the target's location density was obtained. Similarly, let us define the relation between the estimated and the observed probabilities as follows. Let $\tilde{p}^{t-1}(x_i)$, $i = 1, 2, \dots, n$, be estimated probabilities at time $t - 1$. Assume, in addition, that the chosen area is $A^{t-1} \in \mathcal{X}$ and the observation result is $z^{t-1} = z(A^{t-1}, x^{t-1})$. Then, following Theorem 2.1, we say that the observed probabilities $\bar{p}^t(x_i)$, $i = 1, 2, \dots, n$, at time t are defined by the Bayesian rule as follows:

$$\bar{p}^t(x_i) = \frac{z(A^{t-1}, x_i) \tilde{p}^{t-1}(x_i)}{\sum_{j=1}^n z(A^{t-1}, x_j) \tilde{p}^{t-1}(x_j)} \quad (3.9)$$

with $\tilde{p}^0(x_i) = p^0(x_i)$ and

$$\tilde{p}^t(x_i) = \sum_{j=1}^n \rho_{ij} \bar{p}^t(x_j), i = 1, 2, \dots, n, t = 0, 1, 2, \dots \quad (3.10)$$

In particular, for certain observations, while $z^t = \mathbf{1}(A^t, x^t)$, where, as above, $\mathbf{1}$ is the indicator function such that $\mathbf{1}(A^t, x^t) = 1$ if $x^t \in A^t$ and $\mathbf{1}(A^t, x^t) = 0$ otherwise, the equation for the observed probabilities becomes

$$\bar{p}^t(x_i) = \frac{\mathbf{1}(A^{t-1}, x_i)\tilde{p}^{t-1}(x_i)}{\sum_{j=1}^n \mathbf{1}(A^{t-1}, x_j)\tilde{p}^{t-1}(x_j)}, \quad (3.11)$$

which provides a simple normalization of the probabilities over the sample space X .

The next example illustrates this MDP model of the search process.

Example 3.1. Let $X = \{x_1, x_2, x_3, x_4\}$ be a sample space with location probabilities

$$p^0(x_1) = 0.1, p^0(x_2) = 0.2, p^0(x_3) = 0.3, \text{ and } p^0(x_4) = 0.4,$$

and assume that the searcher conducts certain observations such that for the chosen area $A^t \in \mathcal{X}$, the observation result is defined by the indicator $\mathbf{1}$ as $z^t = \mathbf{1}(A^t, x^t)$ and the observed probabilities $\bar{p}^t(x_i)$ are obtained by use of Equation 3.11 with the initial expected probabilities $\tilde{p}^0(x_i) = p^0(x_i)$, $i = 1, \dots, 4$.

Let the target's transition probability matrix be defined as follows:

$$\rho = \begin{pmatrix} 0 & 0.4 & 0.4 & 0.2 \\ 0.4 & 0 & 0.2 & 0.4 \\ 0.4 & 0.2 & 0 & 0.4 \\ 0.2 & 0.4 & 0.4 & 0 \end{pmatrix}.$$

The Markov process that governs the target's movement is shown in Figure 3.3.

It is easy to show that the steady state probabilities $\pi(x_i)$, $i = 1, \dots, 4$, of this process, as specified by Lemma 3.1, are as follows:

$$\pi(x_1) = 0.25, \pi(x_2) = 0.25, \pi(x_3) = 0.25, \text{ and } \pi(x_4) = 0.25.$$

Since the steady state probabilities are equivalent, they do not provide additional information regarding the target location, and a search over the search space \mathcal{X} has to be conducted. For the sample space $X = \{x_1, x_2, x_3, x_4\}$, the search space $\mathcal{X} = 2^X \setminus \emptyset$ includes

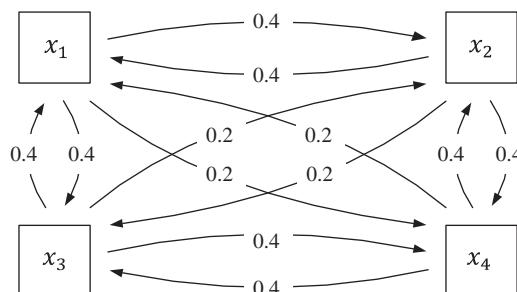


Figure 3.3 Example of the Markov process for the target's movements.

the following observed areas:

$$\begin{aligned} A_1 &= \{x_1\}, A_2 = \{x_2\}, A_3 = \{x_3\}, A_4 = \{x_4\}, \\ A_5 &= \{x_1, x_2\}, A_6 = \{x_1, x_3\}, A_7 = \{x_1, x_4\}, A_8 = \{x_2, x_3\}, A_9 = \{x_2, x_4\}, A_{10} = \{x_3, x_4\}, \\ A_{11} &= \{x_1, x_2, x_3\}, A_{12} = \{x_1, x_2, x_4\}, A_{13} = \{x_1, x_3, x_4\}, A_{14} = \{x_2, x_3, x_4\}, \\ A_{15} &= \{x_1, x_2, x_3, x_4\} \end{aligned}$$

Assume that the searcher implements the following formal policy. As indicated above, let $s^t = (A^t, z^t)$ be a state which consists of the chosen area A^t and a result z^t of its observation. Then if $|A^t| = 1$ and $z^t = 1$, or $|A^t| = n - 1$ and $z^t = 0$, the searcher chooses the action $a_0 = \text{terminate}$ with probability one; that is, the probability (Equation 3.2) for choosing action a_0 is

$$\begin{aligned} \mathfrak{d}^t(a_0) &= \Pr\{a^t = a_0 | s^t = (A^t, z^t), (|A^t| = 1 \text{ and } z^t = 1) \vee (|A^t| = n - 1 \text{ and } z^t = 0)\} \\ &= 1, \end{aligned}$$

while for all other actions a_k , which correspond to the observed areas $A_k \in \mathcal{X}$, $A_k \neq A^t$, it is specified that $\mathfrak{d}^t(a_k) = 0$. Otherwise, the searcher chooses with probability one an action a_k or, the same thing, an area A_k such that it provides a maximal probability with respect to the size of the area, that is,

$$\mathfrak{d}^t(a) = \Pr\left\{a^t = a | s^t = (A^t, z^t), A^t = \operatorname{argmax}_{A_k \in \mathcal{X}} \left\{ \frac{p(A_k)}{|A_k|} \right\}\right\} = 1,$$

and $\mathfrak{d}^t(a_k) = 0$ while $a_k \neq a$. If there are several areas such that $A^t = \operatorname{argmax}_{A_k \in \mathcal{X}} \left\{ \frac{p(A_k)}{|A_k|} \right\}$, then the probability $\mathfrak{d}^t(a)$ is proportional to the number of such areas. In other words, the first condition represents termination of the search, while the exact point, where the target is located, is found, and the second condition represents the choice of area with maximal target location probability, while the cost of the check is proportional to the size of the area.

- **Step $t = 0$.** Assume that at time $t = 0$ the searcher starts with the observed area $A^0 = A_{15} = X$ and obtains an observation result $z^0 = \mathbf{1}(A^0, x^0) = 1$, which indicates that the target is located at one of the points of the sample space X . The obtained results are considered at the next time $t = 1$.
- **Step $t = 1$.** According to Equation 3.11, $\bar{p}^1(x_i) = \tilde{p}^0(x_i)$, and according to Equation 3.10 the estimated probabilities $\tilde{p}^1(x_i)$, $i = 1, \dots, 4$, are as follows:

$$\tilde{p}^1(x_1) = 0.28, \tilde{p}^1(x_2) = 0.26, \tilde{p}^1(x_3) = 0.24, \text{ and } \tilde{p}^1(x_4) = 0.22.$$

Implementation of the searcher's policy results in the area $A^1 = \{x_1\}$, that is, a single area, which satisfies $A^1 = \operatorname{argmax}_{A_k \in \mathcal{X}} \left\{ \frac{\tilde{p}^1(A_k)}{|A_k|} \right\}$. Thus, the searcher chooses this area with probability $\mathfrak{d}_{15}^1(a_1) = 1$, where action a_1 is associated with the area $A_1 = \{x_1\}$. Now the searcher conducts an observation of the chosen area $A^1 = \{x_1\}$, and assumes that the observation fails; that is, let $z^1 = \mathbf{1}(A^1, x^1) = 0$. Then the estimated probabilities $\tilde{p}^1(x_i)$, $i = 1, \dots, 4$, along with the observation result $z^1 = 0$ are passed to the next consideration at time $t = 2$.

- **Step $t = 2$.** An application of Equation 3.11 at time $t = 2$ results in the following observed probabilities:

$$\overline{p}^2(x_1) = 0, \overline{p}^2(x_2) = 0.361, \overline{p}^2(x_3) = 0.333, \text{ and } \overline{p}^2(x_4) = 0.3056.$$

Then, according to Equation 3.10, the estimated probabilities are

$$\tilde{p}^2(x_1) = 0.339, \tilde{p}^2(x_2) = 0.189, \tilde{p}^2(x_3) = 0.194, \text{ and } \tilde{p}^2(x_4) = 0.278.$$

Implementation of the searcher's policy again results in the area $A^2 = \{x_1\}$, which is chosen with probability $\delta_1^2(a_1) = 1$. Assume, again, that observation of the area $A^2 = \{x_1\}$ results in $z^2 = \mathbf{1}(A^2, x^2) = 0$, and observation probabilities $\tilde{p}^2(x_i)$, $i = 1, \dots, 4$, with the result $z^2 = 0$, are considered at the next time $t = 3$.

Continuation of the search by use of the indicated policy $\vec{\delta} = \langle \dots, \delta_1^2(a_1), \delta_{15}^1(a_1), \delta_{15}^0(a_{15}) \rangle$ results in repeated checks of the observed area $A = \{x_1\}$. The limiting probabilities $\pi_{jk}(\vec{\delta})$, $j, k = 1, \dots, 15$, for the MDP model of search with the policy $\vec{\delta}$ are as follows: $\pi_{11}(\vec{\delta}) = 0.3255$, $\pi_{21}(\vec{\delta}) = 0.2117$, $\pi_{31}(\vec{\delta}) = 0.2117$, $\pi_{41}(\vec{\delta}) = 0.2511$, and $\pi_{jk}(\vec{\delta}) = 0$, while $j = 5, \dots, 15$ and $k = 2, \dots, 15$. According to Equation 3.7, the expected average reward $E(R|\vec{\delta})$, which is obtained by such a policy, is

$$E(R|\vec{\delta}) = \sum_{j=1}^4 \pi_{j1}(\vec{\delta}) \frac{\pi_{j1}(\vec{\delta})}{|\{x_j\}|} = \sum_{j=1}^4 (\pi_{j1}(\vec{\delta}))^2 = 0.2586.$$

The implemented policy directly defines decision rules $\delta_i^t(a_k)$, which are equivalent to all times $t = 0, 1, 2, \dots$, and since it does not use information which is available at each time, it is not necessarily optimal. Below, we will consider another example where a different policy is implemented. ■

The policy, which is illustrated by the above, implements a *maximum probability criterion* and does not depend on the probabilities $\rho_{ij}(a_k)$, as specified by Lemmas 3.2 and 3.3, and consequently by the optimization in Equation 3.8. Let us consider certain heuristics that are based on a *maximum information criterion*. As will be demonstrated by numerical results, such a criterion improves the search process in the sense of minimizing the required number of search steps.

Recall the definition of the Shannon entropy (see Equation 2.16 and the considerations in Section 2.2.4). Let $X = \{x_1, x_2, \dots, x_n\}$ be a sample space and assume that random variable x obtains the values for the space X with probabilities $p(x_i) = \Pr\{x = x_i\}$, where $\sum_{i=1}^n p(x_i) = 1$, and $0 \leq p(x_i) \leq 1$, $i = 1, 2, \dots, n$. Then the Shannon entropy of the random variable x is defined as follows [9]:

$$H(x) = H(X) = -\sum_{i=1}^n p(x_i) \log p(x_i),$$

where the logarithm has been taken to base 2 and it is assumed that $0 \log 0 = 0$. In the search problem, where $p(x_i)$ represent target location probabilities, the value $H(x)$ represents the uncertainty regarding the target location from the searcher's point of view. For the absorbing state of the process, in which for some point x_i it holds that $p(x_i) = 1$ while for other points x_j , $j \neq i$, the probability is $p(x_j) = 0$, the entropy is minimal, $H(x) = 0$.

The maximal value of the entropy $H(\mathbf{x}) = -\log 1/n$ is obtained for equal probabilities $p(x_1) = p(x_2) = \dots = p(x_n) = 1/n$; for the case of two-point space see Figure 2.27.

Let \mathbf{y} be another random variable defined over the same set X . Then, the *conditional entropy* of random variable \mathbf{x} given random variable \mathbf{y} is defined as follows [9]:

$$H(\mathbf{x}|\mathbf{y}) = -\sum_{j=1}^n \sum_{i=1}^n p(x_i, y_j) \log p(x_i|y_j),$$

where $p(x_i, y_j) = \Pr\{\mathbf{x} = x_i, \mathbf{y} = y_j\}$ is a joint probability of the variables \mathbf{x} and \mathbf{y} , and $p(x_i|y_j) = \Pr\{\mathbf{x} = x_i | \mathbf{y} = y_j\}$ is a conditional probability of \mathbf{x} given \mathbf{y} . For the conditional entropy $H(\mathbf{x}|\mathbf{y})$ it follows that $0 \leq H(\mathbf{x}|\mathbf{y}) \leq H(\mathbf{x})$, while the lower bound is reached for $\mathbf{x} = \mathbf{y}$, that is, $H(\mathbf{x}|\mathbf{x}) = 0$, and the upper bound $H(\mathbf{x}|\mathbf{y}) = H(\mathbf{x})$ is reached if \mathbf{x} and \mathbf{y} are statistically independent. Finally, the *mutual information* for random variables \mathbf{x} and \mathbf{y} is as follows [9]:

$$I(\mathbf{x}; \mathbf{y}) = H(\mathbf{x}) - H(\mathbf{x}|\mathbf{y}) = H(\mathbf{y}) - H(\mathbf{y}|\mathbf{x}).$$

Since $H(\mathbf{x}|\mathbf{y}) \leq H(\mathbf{x})$, it is always true that $0 \leq I(\mathbf{x}; \mathbf{y}) \leq \min\{H(\mathbf{x}), H(\mathbf{y})\}$, and if \mathbf{x} and \mathbf{y} are statistically independent, then $I(\mathbf{x}; \mathbf{y}) = 0$.

Let us apply the entropy and information measures to the MDP model of search, which is illustrated in Figure 3.2. First, let us define the *observed entropy* at the time t

$$H(\bar{\mathbf{x}}^t) = \bar{H}^t(X) = -\sum_{i=1}^n \bar{p}^t(x_i) \log \bar{p}^t(x_i)$$

that represents the entropy of the target locations after observation of the selected area $A^t \in \mathcal{X}$. Similarly, the *expected entropy* at time $t+1$ is defined as follows:

$$H(\tilde{\mathbf{x}}^{t+1}) = \tilde{H}^{t+1}(X) = -\sum_{i=1}^n \tilde{p}^{t+1}(x_i) \log \tilde{p}^{t+1}(x_i),$$

which is calculated by using the observed probabilities $\bar{p}^t(x_i)$ that were obtained at time t and the target's transition probabilities ρ_{ij} , $i, j = 1, 2, \dots, n$.

Finally, we say that the *expected information* on the target locations is given by

$$I(\tilde{\mathbf{x}}^{t+1}; \bar{\mathbf{x}}^t) = \tilde{I}(A^t) = \tilde{p}^{t+1}(A^t)(\tilde{H}^{t+1}(X) - \bar{H}^t(X)), \quad (3.12)$$

where, as above, $\tilde{p}^{t+1}(A^t) = \sum_{x \in A^t} \tilde{p}^{t+1}(x)$. This information value represents the expected number of bits that will be obtained at time $t+1$, while the uncertainty at time t is given by $\bar{H}^t(X)$ and the selected observed area is A^t . Note that, with respect to the considered MDP model of search, the definition of the expected information is not unique and, in general, it can be calculated for different time horizons, and, as it is being calculated over more steps, it is less myopic, yet requires more calculations.

The application of the entropy and informational measures for the MDP model of search is illustrated by the following example.

Example 3.2 Let us apply the entropy and information measures to the sample space $X = \{x_1, x_2, x_3, x_4\}$ with the same location and transition probabilities as in Example 3.1. As above, we assume that the searcher conducts certain observations, that is, $z^t = \mathbf{1}(A^t, x^t)$ for $A^t \in \mathcal{X}$, $x^t \in X$, and $t = 0, 1, 2, \dots$

In contrast to the previous example, assume that the searcher implements the following maximum expected information criteria:

- single-point case: $A^{t+1} = \operatorname{argmax}_{A \in \mathcal{X}, |A|=1} \{\tilde{p}^{t+1}(A)(\tilde{H}^{t+1}(X) - \overline{H}^t(X))\}$,
- multi-point case: $A^{t+1} = \operatorname{argmax}_{A \in \mathcal{X}} \{\tilde{p}^{t+1}(A)(\tilde{H}^{t+1}(X) - \overline{H}^t(X))\}$,

which are calculated according to Equation 3.12.

Let us consider the choice of the observed area.

- **Step $t = 0$.** Let the searcher at time $t = 0$ start with the observed area $A^0 = A_{15} = X$ and obtain an observation result $z^0 = \mathbf{1}(A^0, x^0) = 1$. This result with the observed probabilities is considered next.
- **Step $t = 1$.** As above, at time $t = 1$, $\overline{p}^1(x_i) = \tilde{p}^0(x_i)$, $i = 1, \dots, 4$, that is, $\overline{p}^1(x_1) = 0.1$, $\overline{p}^1(x_2) = 0.2$, $\overline{p}^1(x_3) = 0.3$, and $\overline{p}^1(x_4) = 0.4$, and the estimated probabilities obtain the same values $\tilde{p}^1(x_1) = 0.28$, $\tilde{p}^1(x_2) = 0.26$, $\tilde{p}^1(x_3) = 0.24$, and $\tilde{p}^1(x_4) = 0.22$.

Choice of the area $A^{t=1}$. The searcher is required to choose an area A^2 , which will be observed at time $t = 2$. For simplicity, let us consider the case of single-point areas $A_1 = \{x_1\}$, $A_2 = \{x_2\}$, $A_3 = \{x_3\}$, and $A_4 = \{x_4\}$, while the calculations for the multi-point areas are the same. Assume that the searcher chooses the area A_1 and supposes that the observation of this area results in $z_1 = \mathbf{1}(A_1, x^2) = 0$. Then the observed probabilities given the observation result z_1 are

$$\overline{p}^2(x_1|z_1) = 0, \overline{p}^2(x_2|z_1) = 0.328, \overline{p}^2(x_3|z_1) = 0.3330, \overline{p}^2(x_4|z_1) = 0.3386,$$

and the estimated probabilities given the observation result z_1 are

$$\tilde{p}^2(x_1|z_1) = 0.3323, \tilde{p}^2(x_2|z_1) = 0.2021, \tilde{p}^2(x_3|z_1) = 0.2011, \tilde{p}^2(x_4|z_1) = 0.2646.$$

Similarly, for the choice of area A_2 with $z_2 = 0$:

$$\begin{aligned} \overline{p}^2(x_1|z_2) &= 0.3245, \overline{p}^2(x_2|z_2) = 0, \overline{p}^2(x_3|z_2) = 0.3351, \overline{p}^2(x_4|z_2) = 0.3404, \\ \tilde{p}^2(x_1|z_2) &= 0.2021, \tilde{p}^2(x_2|z_2) = 0.3330, \tilde{p}^2(x_3|z_2) = 0.2660, \tilde{p}^2(x_4|z_2) = 0.1989; \end{aligned}$$

area A_3 with $z_3 = 0$:

$$\begin{aligned} \overline{p}^2(x_1|z_3) &= 0.3262, \overline{p}^2(x_2|z_3) = 0.3316, \overline{p}^2(x_3|z_3) = 0, \overline{p}^2(x_4|z_3) = 0.3422, \\ \tilde{p}^2(x_1|z_3) &= 0.2021, \tilde{p}^2(x_2|z_3) = 0.2674, \tilde{p}^2(x_3|z_3) = 0.3337, \tilde{p}^2(x_4|z_3) = 0.1979; \end{aligned}$$

and area A_4 with $z_4 = 0$:

$$\begin{aligned} \overline{p}^2(x_1|z_4) &= 0.3280, \overline{p}^2(x_2|z_4) = 0.3333, \overline{p}^2(x_3|z_4) = 0.3387, \overline{p}^2(x_4|z_4) = 0, \\ \tilde{p}^2(x_1|z_4) &= 0.2688, \tilde{p}^2(x_2|z_4) = 0.1989, \tilde{p}^2(x_3|z_4) = 0.1978, \tilde{p}^2(x_4|z_4) = 0.3344. \end{aligned}$$

Then, the expected information (Equation 3.2) for the areas $A_1 = \{x_1\}$, $A_2 = \{x_2\}$, $A_3 = \{x_3\}$, and $A_4 = \{x_4\}$ is

$$\tilde{I}(A_k) = \tilde{p}^2(A_k)(\tilde{H}^2(X|z_k) - \overline{H}^2(X|z_k)),$$

where $\tilde{p}^2(A_k) = \sum_{x \in A_k} \tilde{p}^2(x|z_k) = \tilde{p}^2(x|z_k)$, $\tilde{H}^2(X|z_k) = -\sum_{i=1}^4 \tilde{p}^2(x_i|z_k) \log \tilde{p}^2(x_i|z_k)$, $\overline{H}^2(X|z_k) = -\sum_{i=1}^4 \overline{p}^2(x_i|z_k) \log \overline{p}^t(x_i|z_k)$, and obtains the following values:

$$\tilde{I}(A_1) = 0.0869, \tilde{I}(A_2) = 0.0872, \tilde{I}(A_3) = 0.0873, \text{ and } \tilde{I}(A_4) = 0.0875.$$

Hence, following the criterion of maximum expected information, for the observation at time $t = 1$ the searcher chooses the area $A_4 = \{x_4\}$. Note that such a choice differs from the choice which, according to the maximum probability criterion, was implemented previously in Example 3.1.

Observation of the area $A^{t=1}$. Now the searcher conducts an actual observation of the chosen area $A^1 = A_4 = \{x_4\}$. If an observation succeeds, then the search terminates. Otherwise the searcher continues the search by passing the estimated probabilities $\tilde{p}^1(x_i)$, $i = 1, \dots, 4$, along with the observation result $z^1 = 0$, to the next consideration at time $t = 2$.

Step $t = 2$. The searcher starts with the estimated probabilities $\tilde{p}^1(x_i)$, $i = 1, \dots, 4$, and observation result $z^1 = 0$, which were obtained at the previous time $t = 1$. The required decision making is based on the conditional probabilities $\overline{p}^3(x_i|z_k)$ and $\tilde{p}^3(x_i|z_k)$, $i, k = 1, \dots, 4$. In the same manner the search continues up to finding the target.

For the multi-point observed areas $A_k \in \mathcal{X}$, $k = 1, \dots, 15$, the consideration is similar; however, the resulting observed areas are different, which provides a different average number of search steps up to finding the target. ■

The examples above illustrate the dynamics of MDP in representing search for a static or moving target with maximum probability and maximum expected information criteria. To illustrate the dependence of the search process on the decision criteria below, we present an example of simulated search over a sample space X , which includes nine points, that is, $|X| = 9$. The complete search space $\mathcal{X} = 2^X \setminus \emptyset$ consists of $|\mathcal{X}| = 2^9 - 1 = 511$ areas, including nine single-point areas.

The simulations implemented the maximum probability and maximum expected information criteria, which were presented in Examples 3.1 and 3.2, and were conducted with three types of target movements:

- static target with unitary transition probability matrix;
- Markovian target with uniformly distributed transition probabilities and non-zero probabilities of staying at the current location;
- Brownian target with uniformly distributed transition probabilities and probabilities of staying at the current location.

Each simulation included 1000 replications of the search algorithm, and the search was terminated if one of the following held:

- the target was found, as assumed by the MDP model of search; or
- the search failed to find the target within 100 steps, which actually happened in the search for a moving (Markovian or Brownian) target.

The results of these simulations are presented in the following table. Note that the number of steps includes the initial step $t = 0$, in which the complete sample space $A^0 = X$

is observed. Since it was specified for all simulations that for all times the target is in the sample space, the result of such observations was $z^0 = 1$.

In the following simulated examples, let us assume that, at the beginning of the search, an offline search procedure can be used. An offline procedure is particularly appealing when a priori information on the target location is updated gradually in an increasing rate: the information is relatively fixed at the beginning of the search, thus an offline procedure is feasible, while most of the updates are revealed as the searcher reaches the target's neighborhood, thus requiring an online procedure. For example, in software testing the exact location of a bug is known once the non-conforming module has been identified and inspected; or, similarly, in the quality inspection of wafers, information on the exact location of contaminated spots is evident once the visual inspection equipment is focused close to the contaminated area. Accordingly, the implication of the above understanding is that a fast zoom-in process can be obtained at the beginning of group testing by an offline procedure, and then, in the vicinity of the target, an online procedure can be used to account for new side information. Moreover, most of the offline group-testing procedures, as well as the proposed ILRTA* algorithm with a Huffman neighborhood (see the next sections), can approach the entropy lower bound for the average number of tests. This bound, in the worst case, is close to the entropy of the location probability plus one [9]. This fact implies that when the search space contains, for example, 10 000 units, it will require up to 14 rounds of binary group tests to reach the vicinity of the target. Taking all this into consideration, in the simulated study we focus on the last stage of the search near the target, while ignoring the relatively fixed (and optimal) number of test rounds till this area is reached. Accordingly, the sample space is relatively small.

	Search criterion	Number of steps			
		Minimum	Maximum	Mean	Standard deviation
Static target	Maximum estimated probability	1	9	4.928	2.547
	Maximum expected information, <i>single-point areas</i>	1	9	4.885	2.548
	Maximum expected information, <i>multi-point areas</i>	1	8	4.459	1.205
Markovian target	Maximum estimated probability	1	>100	–	–
	Maximum expected information, <i>single-point areas</i>	1	54	5.535	5.451
	Maximum expected information, <i>multi-point areas</i>	1	>100	–	–
Brownian target	Maximum estimated probability	1	>100	–	–
	Maximum expected information, <i>single-point areas</i>	1	29	4.747	3.840
	Maximum expected information, <i>multi-point areas</i>	1	28	6.103	3.930

In the table, the cases in which the search failed to find the target in 100 steps are denoted by ‘> 100.’ From the results it follows that in the search for a static target, both probabilistic and informational criteria lead to finding the target, while in the search for a moving (Markovian or Brownian) target the maximum probability criterion failed and maximum expected information criteria lead to finding the target. In the case of a Markovian target, the target was found by using the maximum expected information criterion with single-point observed areas, and in the case of the Brownian target, by using the maximum expected information criterion with both single-point and multi-point observed areas.

A statistical comparison of the obtained results for the replications, in which different criteria lead to finding the target (static target search and Brownian target search), demonstrates the following:

In the search for a static target:

- maximum probability criterion and maximum information criterion with single-point areas are significantly *equivalent*;
- maximum probability criterion and maximum information criterion with multi-point areas are significantly *different*;
- maximum information criterion with single-point areas and maximum information criterion with multi-point areas are significantly *different*.

The best result is provided by the maximum information criterion with multi-point subsets.

In the search for a Brownian target:

- maximum information criterion with single-point areas and maximum information criterion with multi-point areas are significantly *different*.

The best result is provided by the maximum information criterion with single-point areas.

The results obtained as well as the optimal procedures presented in Section 2.2.4 support an application of information theory methods in the online search procedures, and in Chapter 4 we develop certain informational algorithms of search and consider their properties.

3.2 Partially observable MDP model and dynamic programming approach

In the previous consideration of the MDP model of search, we assumed that the observations are certain, and at each time the search obtains exact information regarding target location in the observed area. However, when considering search and screening problems (see Section 2.1) and group-testing search (see Section 2.2), it was indicated that the observation result is not necessarily certain and often is specified by the probability of detecting or not detecting the target in the observed area. To analyze such a situation with the same Markovian assumptions regarding the target movements and system dynamics, a partially observable Markov decision process (POMDP) is applied.

Below, we consider two basic models of search, which apply POMDP representation to the problem, and implement the dynamic programming method for its solution.

3.2.1 MDP with uncertain observations

Let us start with general definitions of POMDP on the basis of the MDP considered above. The basis for this consideration is provided by the survey in [10] and tutorial in [11] with certain adaptations with respect to the search problems. As indicated above, following the needs of the search models, we restrict ourselves to the consideration of finite sets of states and actions.

Let, as above, $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_n\}$ be a set of states, and assume that over the set \mathcal{S} a core Markov process is defined with the transition probability matrix $\rho = ||\rho_{ij}||_{n \times n}$, $\rho_{ij} = \Pr\{\mathbf{s}'^{t+1} = \mathbf{s}_j | \mathbf{s}^t = \mathbf{s}_i\}$. In the MDP above, it was implied that at each time t , $t = 0, 1, 2, \dots$, observation of the core process by the decision-maker certainly results in the current state $\mathbf{s}^t \in \mathcal{S}$ of the core process.

Now, in contrast, assume that, similar to the search and screening methods (see Section 2.1.2), the observational abilities of the decision-maker are restricted and the observations provide uncertain information regarding the state of the core process. We denote by $\widehat{\mathbf{s}}^t \in \mathcal{S}$ a state which is obtained by the decision-maker as a result of the core process observation at time t , $t = 0, 1, 2, \dots$, if the uncertain observations are allowed; then, in general, $\widehat{\mathbf{s}}^t \neq \mathbf{s}^t$, where $\mathbf{s}^t \in \mathcal{S}$ is an actual state of the core process at the considered time. Then, remaining with this notation, which is used in Section 2.1.2, the detection probability of the state is as follows:

$$\psi_{ij} = \Pr\{\text{detected state } \widehat{\mathbf{s}} = \mathbf{s}_j | \text{core process is in the state } \mathbf{s} = \mathbf{s}_i\}, i, j = 1, 2, \dots, n.$$

Note that, in contrast to the detection probability function ψ , which is implemented in Section 2.1.2, in this definition it can occur that $\psi_{ij} > 0$ while $i \neq j$, that is, false observations are allowed. In general, the set of states which are observed by the decision-maker can be different from the actual set of states \mathcal{S} and can have a different nature. Then, the set of the observed states is called the set of messages $\mathbb{M} = \{\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_l\}$ and the detection probability is defined as $\psi_{ij} = \Pr\{\mathbf{m} = \mathbf{m}_j | \mathbf{s} = \mathbf{s}_i\}$, $i = 1, 2, \dots, n$, $j = 1, 2, \dots, l$. Nevertheless, for the purposes of the search problems we will assume that $\mathbb{M} = \mathcal{S}$ and will use the definition of the detection probability given above.

Let $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$ be a set of actions which are available to the decision-maker. Similar to the above definition (Equation 3.1), the conditional transition probability

$$\rho_{ij}(a_k) = \Pr\{\mathbf{s}'^{t+1} = \mathbf{s}_j | a^t = a_k, \mathbf{s}^t = \mathbf{s}_i\}, \quad (3.13)$$

where $i, j = 1, 2, \dots, n$ and $k = 1, 2, \dots, m$, specifies the dynamics of the core process, while at time t , $t = 0, 1, 2, \dots$, the decision-maker chooses an action $a^t = a_k$. In addition, we assume that the detection probability also depends on the chosen action $a^t = a_k$ and specify that

$$\psi_{ij}(a_k) = \Pr\{\widehat{\mathbf{s}}^t = \mathbf{s}_j | a^t = a_k, \mathbf{s}^t = \mathbf{s}_i\}, \quad (3.14)$$

where $i, j = 1, 2, \dots, n$ and $k = 1, 2, \dots, m$.

The decision making regarding the choice of action a^t at each time t is conducted on the basis of the *detection* or *observation history*, which is similar to the above history $\overrightarrow{a \circ s}$ of system dynamics, but, in contrast, is defined by the use of detected states $\widehat{\mathbf{s}} \in \mathcal{S}$ as follows:

$$\overrightarrow{a^t \circ \widehat{\mathbf{s}}^t} = \langle a^t, \widehat{\mathbf{s}}^t, a^{t-1}, \widehat{\mathbf{s}}^{t-1}, \dots, a^1, \widehat{\mathbf{s}}^1, a^0, \widehat{\mathbf{s}}^0 \rangle, t = 0, 1, 2, \dots$$

Similar to the probability (Equation 3.1), on the basis of the detection history $\overrightarrow{a^t \circ \hat{s}^t}$, the probability is defined

$$\begin{aligned}\hat{\mathbb{I}}_i(\overrightarrow{a^{t-1} \circ \hat{s}^{t-1}}) &= \hat{\mathbb{I}}^t(s_i) \\ &= \Pr\{s^t = s_i | a^{t-1}, \hat{s}^{t-1}, \dots, a^1, \hat{s}^1, a^0, \hat{s}^0\}\end{aligned}\quad (3.15)$$

such that at time t the core Markov process is in state $s_i \in \mathcal{S}$ given the detection history $\overrightarrow{a^{t-1} \circ \hat{s}^{t-1}}$ up to the previous time $t - 1$. The probability vector

$$\mathbb{I}^t = (\hat{\mathbb{I}}^t(s_1), \hat{\mathbb{I}}^t(s_2), \dots, \hat{\mathbb{I}}^t(s_n)), \quad (3.16)$$

where $\sum_{i=1}^n \hat{\mathbb{I}}^t(s_i) = 1$, $\hat{\mathbb{I}}^t(s_i) \geq 0$, $i = 1, 2, \dots, n$, $t = 0, 1, 2, \dots$, which given a time t includes the probabilities $\hat{\mathbb{I}}_i^t = \hat{\mathbb{I}}^t(s_i)$ for all states $s_i \in \mathcal{S}$, $i = 1, 2, \dots, n$, is called the *information vector* and summarizes all information necessary for the decision making regarding the choice of action a^t at time t [10, 12].

According to the described model, the core Markov process is not directly observable; thus it is often called a *hidden* or *partially observable* Markov process, and the model for decision making is called the *hidden Markov model* (HMM) [11, 13] or POMDP [4, 10, 14].

This partially observable decision process acts on the set of states \mathcal{S} , while for decision making the information vector (Equation 3.16) is applied. The analysis of the POMDP models is based on the next general fact that allows a representation of the partially observable process over the set \mathcal{S} in the form of completely observable MDP over the information vectors.

Lemma 3.4 [10, 14, 15]. *For any time t , $t = 0, 1, 2, \dots$, and any fixed sequence of actions $\overrightarrow{a} = \langle a^t, a^{t-1}, \dots, a^1, a^0 \rangle$, the sequence $\langle \mathbb{I}^t, \mathbb{I}^{t-1}, \dots, \mathbb{I}^1, \mathbb{I}^0 \rangle$ of information vectors forms a Markov process, that is,*

$$\begin{aligned}\Pr\{\mathbb{I}^{t+1} \text{ is an information vector} | \mathbb{I}^t, \mathbb{I}^{t-1}, \dots, \mathbb{I}^1, \mathbb{I}^0, a^t\} \\ = \Pr\{\mathbb{I}^{t+1} \text{ is an information vector} | \mathbb{I}^t, a^t\}.\end{aligned}$$

This lemma allows consideration of the POMDP models by means of MDPs, acting on the set

$$\mathbb{S}(n) = \left\{ \mathbb{I} = (\hat{\mathbb{I}}_1, \hat{\mathbb{I}}_2, \dots, \hat{\mathbb{I}}_n) \mid \sum_{i=1}^n \hat{\mathbb{I}}_i = 1, \hat{\mathbb{I}}_i \geq 0, i = 1, 2, \dots, n \right\}$$

of the information vectors. Note that in contrast to the set of states \mathcal{S} , which is finite or, at most, countable, the set $\mathbb{S}(n)$ is uncountable. In such a case, the rules for choosing actions a^t , $t = 0, 1, 2, \dots$, are defined by the probabilities

$$\mathfrak{d}_{\mathbb{I}}^t(a_k) = \Pr\{a^t = a_k | \mathbb{I}^t, \mathbb{I}^t \in \mathbb{S}(n)\},$$

which, in contrast to Equation 3.2, are determined by certain probability density functions over $\mathbb{S}(n)$.

Similar to the above, let $\overrightarrow{\mathfrak{d}}(\mathbb{I}^t) = \langle \mathfrak{d}_{\mathbb{I}}^t, \mathfrak{d}_{\mathbb{I}}^{t-1}, \dots, \mathfrak{d}_{\mathbb{I}}^1, \mathfrak{d}_{\mathbb{I}}^0 \rangle$ be a policy which is implemented by the decision-maker using the information vector \mathbb{I}^t , and $R(s^t, a^t)$ be an

immediate reward at time t , $t = 0, 1, 2, \dots$, while the core Markov process is in the state s^t and the decision-maker chooses the action a^t . Then, the *expected discounted infinite horizon reward* of POMDP is defined as follows [10]:

$$E_{\text{POMDP}}^{\text{inf}}(R | \overrightarrow{\delta}(\mathbb{I}^\infty)) = E \left(\sum_{t=0}^{\infty} \delta^t R(s^t, a^t) | \overrightarrow{\delta}(\mathbb{I}^t) \right),$$

where actions a^t are chosen according to the rules δ^t from the policy $\overrightarrow{\delta}$, and $\delta \in (0, 1)$ is a real number called the *discount factor*. The goal is to find a policy $\overrightarrow{\delta}^*(\mathbb{I}^\infty)$ such that the expected reward $E_{\text{POMDP}}^{\text{inf}}(R | \overrightarrow{\delta}^*(\mathbb{I}^\infty))$ reaches its maximum over all possible policies $\overrightarrow{\delta}(\mathbb{I}^\infty)$:

$$E_{\text{POMDP}}^{\text{inf}}(R | \overrightarrow{\delta}^*(\mathbb{I}^\infty)) = \sup_{\overrightarrow{\delta}(\mathbb{I}^\infty)} \left\{ E_{\text{POMDP}}^{\text{inf}}(R | \overrightarrow{\delta}(\mathbb{I}^\infty)) \right\}.$$

The policy $\overrightarrow{\delta}^*(\mathbb{I}^\infty)$, which provides the maximal value of the expected reward given information vectors \mathbb{I}^t , $t = 0, 1, 2, \dots$, is called the *optimal policy*, and a value of the expected reward, which is provided by the optimal policy, is usually denoted by

$$V(\mathbb{I}^\infty) = E_{\text{POMDP}}^{\text{inf}}(R | \overrightarrow{\delta}^*(\mathbb{I}^\infty)).$$

In the above consideration of the MDP model, the linear programming maximization problem was formulated. Now, let us formulate the problem of maximization of the value $V(\mathbb{I}^\infty)$ in dynamic programming form.

The Bellman equations for the recursive calculation of the value $V(\mathbb{I}^\infty)$ are defined as follows [10, 16]. Assume that by implementing the rule δ_i^t at time t , $t = 0, 1, 2, \dots$, the information vector $\mathbb{I}^t \in \mathbb{S}(n)$ is obtained and an action a^t is chosen. We denote by

$$\phi_{jk}(\mathbb{I}^t) = \Pr\{\hat{s}^t = s_j | a^t = a_k, \mathbb{I}^t\} \quad (3.17)$$

where $i, j = 1, 2, \dots, n$ and $k = 1, 2, \dots, m$, a probability that if the chosen action is $a^t = a_k$ and the information vector is $\mathbb{I}^t \in \mathbb{S}(n)$, then the decision-maker detects the state $\hat{s}^t = s_j$. By using the values defined above (Equations 3.13–3.16), the probability (Equation 3.17) is calculated as follows:

$$\begin{aligned} \phi_{jk}(\mathbb{I}^t) &= \sum_{i=1}^n \Pr\{\hat{s}^t = s_j | a^t = a_k, s^t = s_i\} \\ &\quad \times \sum_{\hat{i}=1}^n \Pr\{s^{t+1} = s_i | a^t = a_k, s^t = s_i\} \mathbb{I}^t(s_i) \\ &= \sum_{i=1}^n \psi_{ij}(a_k) \times \sum_{\hat{i}=1}^n \rho_{ii}^t(a_k) \times \mathbb{I}^t(s_i), \end{aligned} \quad (3.18)$$

where $j = 1, 2, \dots, n$, $k = 1, 2, \dots, m$, and $t = 0, 1, 2, \dots$. Then the probability

$$\mathbb{I}_{jk}^{t+1}(s_i) = \Pr\{s^{t+1} = s_i | a^t = a_k, \hat{s}^t = s_j, a^{t-1}, \hat{s}^{t-1}, \dots, a^1, \hat{s}^1, a^0, \hat{s}^0\}$$

that the core process at time $t + 1$ will be in the state s_i , while at time t state s_j was detected and action a_k was chosen, is as follows:

$$\mathbb{I}_{jk}^{t+1}(s_i) = \frac{\psi_{ij}(a_k) \times \sum_{\hat{i}=1}^n \rho_{ii}^t(a_k) \times \mathbb{I}^t(s_i)}{\phi_{jk}(\mathbb{I}^t)}.$$

Given the detected state s_j and chosen action a_k , the information vector for time $t + 1$ includes the probabilities $\mathbb{I}_{jk}^{t+1}(s_i)$ for the states s_i , $j = 1, 2, \dots, n$, that is,

$$\mathbb{I}_{jk}^{t+1} = (\mathbb{I}_{jk}^{t+1}(s_1), \mathbb{I}_{jk}^{t+1}(s_2), \dots, \mathbb{I}_{jk}^{t+1}(s_n)).$$

Finally, we recall that $R(s_i, a_k)$ stands for an immediate reward with respect to the state s_i and action a_k , and denote by

$$\mathbf{R}_k = (R(s_1, a_k), R(s_2, a_k), \dots, R(s_n, a_k))$$

a vector of the immediate rewards over all states s_i , $i = 1, 2, \dots, n$, while the action a_k , $k = 1, 2, \dots, m$, is chosen. Then an application of the defined values results in the following lemma.

Lemma 3.5 [10, 24]. *The infinite horizon value $V(\mathbb{I}^\infty)$ satisfies the recursive optimality equation*

$$V(\mathbb{I}^t) = \max_{k=1,2,\dots,m} \left\{ \mathbb{I}^t \times \mathbf{R}_k + \delta \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1}) \phi_{jk}(\mathbb{I}^t) \right\}, \quad (3.19)$$

where $\mathbb{I}^t \times \mathbf{R}_k = \sum_{i=1}^n \mathbb{I}^t(s_i) R(s_i, a_k)$ is a scalar multiplication of the vectors.

Proof. Let $\overrightarrow{\mathfrak{d}}(\mathbb{I}^\infty)$ be some policy such that at time $t = 0$ the probabilities of choosing actions a_k are $\overrightarrow{\mathfrak{d}}_{\mathbb{I}}^0(a_k)$, $k = 1, 2, \dots, m$. Then

$$\begin{aligned} E_{\text{POMDP}}^{\text{inf}}(R | \overrightarrow{\mathfrak{d}}(\mathbb{I}^\infty)) &= E \left(\sum_{t=0}^{\infty} \delta^t R(s^t, a^t) | \overrightarrow{\mathfrak{d}}(\mathbb{I}^t) \right) \\ &= E(\delta^0 R(s^0, a^0) | \overrightarrow{\mathfrak{d}}(\mathbb{I}^0)) + E \left(\sum_{t=1}^{\infty} \delta^t R(s^t, a^t) | \overrightarrow{\mathfrak{d}}(\mathbb{I}^t) \right) \\ &= \sum_{k=1}^m \overrightarrow{\mathfrak{d}}_{\mathbb{I}}^0(a_k) \mathbb{I}^0 \\ &\quad \times \sum_{k=1}^m \overrightarrow{\mathfrak{d}}_{\mathbb{I}}^0(a_k) \sum_{j=1}^n E \left(\sum_{t=1}^{\infty} \delta^t R(s^t, a^t) | \overrightarrow{\mathfrak{d}}(\mathbb{I}^{t=1 \rightarrow \infty}) \right) \phi_{jk}(\mathbb{I}^0) \end{aligned}$$

where $\overrightarrow{\mathfrak{d}}(\mathbb{I}^{t=1 \rightarrow \infty})$ stands for the policy $\overrightarrow{\mathfrak{d}}(\mathbb{I}^\infty)$ starting from time $t = 1$, and $E \left(\sum_{t=1}^{\infty} \delta^t R(s^t, a^t) | \overrightarrow{\mathfrak{d}}(\mathbb{I}^{t=1 \rightarrow \infty}) \right)$ is a discounted reward, while the information vector at this time is \mathbb{I}^1 .

In other words, the value $V(\mathbb{I}^0 | \overrightarrow{\mathfrak{d}})$ of the expected reward given the policy $\overrightarrow{\mathfrak{d}}(\mathbb{I}^\infty)$ and an information vector at time $t = 0$ is as follows:

$$V(\mathbb{I}^0 | \overrightarrow{\mathfrak{d}}) = \sum_{k=1}^m \overrightarrow{\mathfrak{d}}_{\mathbb{I}}^0(a_k) \left[\mathbb{I}^0 \times \mathbf{R}_k + \sum_{j=1}^n E \left(\sum_{t=1}^{\infty} \delta^t R(s^t, a^t) | \overrightarrow{\mathfrak{d}}(\mathbb{I}^{t=1 \rightarrow \infty}) \right) \phi_{jk}(\mathbb{I}^0) \right].$$

However, because of the discounting, the inequality

$$E \left(\sum_{t=1}^{\infty} \delta^t R(s^t, a^t) | \overrightarrow{\mathfrak{d}}(\mathbb{I}^{t=1 \rightarrow \infty}) \right) \leq \delta V(\mathbb{I}^1 | \overrightarrow{\mathfrak{d}})$$

holds. Hence, for all $t = 0, 1, 2, \dots$ it follows that

$$V(\mathbb{I}^t | \overrightarrow{\mathfrak{d}}) \leq \sum_{k=1}^m \overrightarrow{\mathfrak{d}}_{\mathbb{I}}^t(a_k) \left[\mathbb{I}^t \times \mathbf{R}_k + \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1} | \overrightarrow{\mathfrak{d}}) \phi_{jk}(\mathbb{I}^t) \right]$$

$$\begin{aligned} &\leq \sum_{k=1}^m \mathfrak{d}'_k(a_k) \left[\max_{k=1,2,\dots,m} \left\{ \mathbb{I}^t \times R_k + \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1} | \vec{\mathfrak{d}}) \phi_{jk}(\mathbb{I}^t) \right\} \right] \\ &= \max_{k=1,2,\dots,m} \left\{ \mathbb{I}^t \times R_k + \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1} | \vec{\mathfrak{d}}) \phi_{jk}(\mathbb{I}^t) \right\}, \end{aligned}$$

and since the policy $\vec{\mathfrak{d}}(\mathbb{I}^\infty)$ is arbitrary, one finally obtains that

$$V(\mathbb{I}^t) \leq \max_{k=1,2,\dots,m} \left\{ \mathbb{I}^t \times R_k + \delta \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1}) \phi_{jk}(\mathbb{I}^t) \right\}.$$

Backwards, let $a_{\hat{k}}$ be such an action that

$$\mathbb{I}^t \times R_{\hat{k}} + \delta \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1}) \phi_{jk}(\mathbb{I}^t) = \max_{k=1,2,\dots,m} \left\{ \mathbb{I}^t \times R_k + \delta \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1}) \phi_{jk}(\mathbb{I}^t) \right\},$$

and let $\vec{\mathfrak{d}}(\mathbb{I}^\infty)$ be a policy such that at time $t = 0$ the action $a^0 = a_{\hat{k}}$ is chosen with probability $\mathfrak{d}_1^0(a_{\hat{k}}) = 1$, and starting from time $t = 1$ it follows that $V(\mathbb{I}^t | \vec{\mathfrak{d}}) \geq V(\mathbb{I}^t) - \varepsilon$, where $\varepsilon > 0$. Then

$$\begin{aligned} V(\mathbb{I}^t | \vec{\mathfrak{d}}) &= \mathbb{I}^t \times R_{\hat{k}} + \delta \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1} | \vec{\mathfrak{d}}) \phi_{jk}(\mathbb{I}^t) \\ &\geq \mathbb{I}^t \times R_{\hat{k}} + \delta \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1}) \phi_{jk}(\mathbb{I}^t) - \delta\varepsilon. \end{aligned}$$

This implies that

$$\begin{aligned} V(\mathbb{I}^t) &\geq \mathbb{I}^t \times R_{\hat{k}} + \delta \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1}) \phi_{jk}(\mathbb{I}^t) - \delta\varepsilon \\ &= \max_{k=1,2,\dots,m} \left\{ \mathbb{I}^t \times R_k + \delta \sum_{j=1}^n V(\mathbb{I}_{jk}^{t+1}) \phi_{jk}(\mathbb{I}^t) \right\} - \delta\varepsilon, \end{aligned}$$

and, since ε is arbitrary, the required equality follows. ■

The recursive equations which are similar to Equation 3.19 can be written for the minimization problem, where instead of rewards R_k certain costs C_k of the actions a_k , $k = 1, 2, \dots, m$, are used [17]. Since the information vectors \mathbb{I}^t are probability vectors which allow direct calculation of the entropy and information measures indicated above (see Section 3.1.2), the rewards and costs in informational form can be defined [18]. For additional information regarding MDP and POMDP models and their implementation see, for example, [12, 19]. Below, we implement the MDP and POMDP models and their dynamic programming solutions for two models of search.

3.2.2 Simple Pollock model of search

Let us start with a simple model of search for a moving target that was suggested by Pollock [20]. In the model, the target moves between two boxes according to a given Markov process, and the searcher is allowed to check one of the boxes and obtain perfect or imperfect observation results. The Pollock model provides an illustration of MDP and POMDP techniques for solving search problems, and forms a basis for the search algorithms which are considered in the next sections.

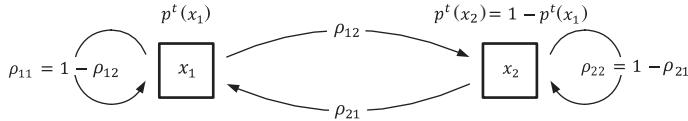


Figure 3.4 Target's movements in the Pollock model of search.

According to the Pollock model of search, let $X = \{x_1, x_2\}$ be a sample space which includes two possible target locations. As above, target location probabilities at time t , $t = 0, 1, 2, \dots$, are denoted by $p_i^t = p^t(x_i) = \Pr\{x = x_i\}$, $i = 1, 2$, and $p_1^t + p_2^t = 1$. Since $p_1^t = 1 - p_2^t$ for any time t , the Pollock model analyzes the search given the probabilities p_i^t of the target location at the point x_i . Regarding the target, it is assumed that its movement is governed by a Markov process with transition probabilities $\rho_{ij} = \Pr\{x^{t+1} = x_j | x^t = x_i\}$, $i = 1, 2$, and $\rho_{12} + \rho_{11} = 1$, $\rho_{21} + \rho_{22} = 1$. The Markov process that governs the target's movements in the Pollock model of search is illustrated in Figure 3.4.

Similar to Section 2.1.2, let the detection probability be as follows:

$$\psi_i = \psi(x_i)$$

$$= \Pr\{\text{target is detected in } x_i | \text{target is located in } x_i, \text{ the searcher checks } x_i\}.$$

Note that in contrast to the definition, which is given in Section 2.1.2, here we assume that the detection probability does not depend on time. If the detection probability $\psi_i = 1$, then the detection is said to be *perfect*, and if $0 < \psi_i < 1$, then the detection is said to be *imperfect*.

As indicated above, at each time t , $t = 0, 1, 2, \dots$, the searcher chooses a point x_i , $i = 1, 2$, for observation. In terms of search and screening (see Section 2.1.1) and of group-testing search (see Section 2.2.1) such a choice means selecting a single-point observed area $A^t \in \mathcal{X} = \{\{x_1\}, \{x_2\}\}$, and obtaining an observation result $z^t = z(A^t, x^t)$ such that $z^t = z(A^t, x^t) = \psi(x^t)$ if $x^t \in A^t$, and $z^t = 0$ otherwise. For perfect detection the observation result is $z^t = 1$ if the searcher checks the point at which the target is located, and $z^t = 0$ otherwise. Thus, according to the above MDP model of search, the actions of the searcher are associated with the choices of the points.

In the Pollock model, the searcher's strategy includes observations of the points x_i , $i = 1, 2$, with appropriate probability updating. Regarding these strategies, the model addresses two main problems:

1. What strategy provides a minimal expected number of observations up to detecting the target, and what is this number?
2. Given a number of observations, what strategy provides maximal probability of detecting the target, and what is the value of this probability?

Let us formalize these problems. As indicated above, since $p_1^t = 1 - p_2^t$, it is enough to consider the probability p_i^t of the target location at the point x_1 . Let $\tilde{p}_{1|i}^t = \tilde{p}^t(x_1 | z(\{x_i\}, x^{t-1}) = 0)$, $i = 1, 2$, be the probability of the target location at the point x_1 in time t , $t = 0, 1, 2, \dots$, given that observation of the point x_i at the previous time $t - 1$ failed. Then, before starting the search at $t = 0$, the probability $\tilde{p}_{1|i}^0$ is equal to

the location probability $\tilde{p}_{1|i}^0 = p_1^0$, and for the next times $t > 0$, it obtains the following values:

$$\begin{aligned}\tilde{p}_{1|1}^t &= \frac{\tilde{p}_{1|1}^{t-1}(1 - \psi_1)\rho_{11} + (1 - \tilde{p}_{1|1}^{t-1})\rho_{21}}{1 - \psi_1\tilde{p}_{1|1}^{t-1}}, \\ \tilde{p}_{1|2}^t &= \frac{\tilde{p}_{1|2}^{t-1}\rho_{11} + (1 - \tilde{p}_{1|2}^{t-1})(1 - \psi_2)\rho_{21}}{1 - \psi_2(1 - \tilde{p}_{1|2}^{t-1})}.\end{aligned}\quad (3.20)$$

For example, if $t = 1$, then

$$\tilde{p}_{1|1}^1 = \frac{p_1^0(1 - \psi_1)\rho_{11} + p_2^0\rho_{21}}{1 - \psi_1 p_1^0}, \quad \tilde{p}_{1|2}^1 = \frac{p_1^0\rho_{11} + p_2^0(1 - \psi_2)\rho_{21}}{1 - \psi_2 p_2^0}.$$

Notice that these expressions follow directly from Equations 3.9 and 3.10 applied for the search over the two-point sample space.

As required by the first problem, denote by $E_{\text{Pol}}(z|p_1)$ a minimal expected number of observations, which are required to detect the target, while the a priori probability of the target location at the point x_1 is p_1 . By using the posterior probabilities $\tilde{p}_{1|i}^t$, $i = 1, 2$, then $E_{\text{Pol}}(z|p_1)$ is defined as follows:

$$E_{\text{Pol}}(z|p_1) = 1 + \min \begin{cases} (1 - p_1\psi_1) E_{\text{Pol}}(z|\tilde{p}_{1|1}^1) & \text{first observation of } x_1, \\ (1 - (1 - p_1)\psi_2) E_{\text{Pol}}(z|\tilde{p}_{1|2}^1) & \text{first observation of } x_2, \end{cases} \quad (3.21)$$

where the first value represents the case in which the searcher starts with the observation point x_1 , and the second value represents the case in which the searcher starts with the observation of point x_2 . As indicated above, p_1 stands for the a priori probability of the target location at the point x_1 , so $p_2 = 1 - p_1$, and $\tilde{p}_{1|i}^0$, $i = 1, 2$, are defined by Equation 3.20. In addition, note that the function $E_{\text{Pol}}(z|p_1)$ is concave with respect to p_1 , and for any p_1 and ψ_1 and ψ_2 it follows that $E_{\text{Pol}}(z|p_1) < \infty$.

Similarly, denote by $P(p_1|t)$ a maximal probability of detecting the target at time t , $t = 0, 1, 2, \dots$, while the a priori probability of the target location at the point x_1 is p_1 . Then, the recurrent equations for the probabilities $P(p_1|t)$ are as follows:

$$\begin{aligned}P_{\text{Pol}}(p_1|t) \\ = \max \begin{cases} p_1\psi_1 + (1 - p_1\psi_1) P_{\text{Pol}}(\tilde{p}_{1|1}^1|t-1) & \text{first observation of } x_1, \\ (1 - p_1)\psi_2 + (1 - (1 - p_1)\psi_2) P_{\text{Pol}}(\tilde{p}_{1|2}^1|t-1) & \text{first observation of } x_2, \end{cases} \\ P_{\text{Pol}}(p_1|0) = 0,\end{aligned}\quad (3.22)$$

where, as above, the first value represents the case in which the searcher starts with the observation point x_1 , and the second value represents the case in which the searcher starts with the observation of point x_2 . The probability $P_{\text{Pol}}(p_1|t)$ can be obtained directly by iterating up to time t , $t = 0, 1, 2, \dots$, and with increasing t it quickly converges to unity [20].

Let us consider the minimal expected number of observations $E_{\text{Pol}}(z|p_1)$. Following Pollock [20], we start with the case of perfect detection, and then present bounds for $E_{\text{Pol}}(z|p_1)$ in the general case.

Perfect detection. In such a case, the detection probabilities are $\psi_1 = \psi_2 = 1$; that is, if the searcher observes point x_i , $i = 1, 2$, and the target is located at this point, then it is detected and the search terminates.

From Equation 3.20 it follows that if $\psi_1 = \psi_2 = 1$, then

$$\begin{aligned}\tilde{p}_{1|1}^1 &= \frac{p_1^0(1 - \psi_1)\rho_{11} + p_2^0\rho_{21}}{1 - \psi_1 p_1^0} = \frac{(1 - p_1^0)\rho_{21}}{1 - p_1^0} = \rho_{21}, \\ \tilde{p}_{1|2}^1 &= \frac{p_1^0\rho_{11} + p_2^0(1 - \psi_2)\rho_{21}}{1 - \psi_2 p_2^0} = \frac{(1 - p_2^0)\rho_{11}}{1 - p_2^0} = \rho_{11}.\end{aligned}$$

Thus, the minimal expected number of observations is specified as follows:

$$E_{\text{Pol}}(z|p_1) = 1 + \min \begin{cases} (1 - p_1) E_{\text{Pol}}(z|\rho_{21}) & \text{first observation of } x_1, \\ p_1 E_{\text{Pol}}(z|\rho_{11}) & \text{first observation of } x_2, \end{cases}$$

and the maximal probability of detecting the target at time t , $t = 0, 1, 2, \dots$, is given by the recurrent equation

$$P_{\text{Pol}}(p_1|t) = \max \begin{cases} p_1 + (1 - p_1) P_{\text{Pol}}(\rho_{21}|t-1) & \text{first observation of } x_1, \\ 1 - p_1 + p_1 P_{\text{Pol}}(\tilde{p}_{1|2}^1|t-1) & \text{first observation of } x_2, \end{cases}$$

$$P_{\text{Pol}}(p_1|0) = 0.$$

Let us consider the value $E_{\text{Pol}}(z|p_1)$. According to the minimization problem, the point x_1 can be chosen for the first observation if it follows that

$$(1 - p_1)E_{\text{Pol}}(z|\rho_{21}) \leq p_1 E_{\text{Pol}}(z|\rho_{11}),$$

or, in other words, if

$$p_1 \geq \frac{E_{\text{Pol}}(z|\rho_{21})}{E_{\text{Pol}}(z|\rho_{21}) + E_{\text{Pol}}(z|\rho_{11})} = p_1^*.$$

Then, the equation for the expected number $E_{\text{Pol}}(z|p_1)$ becomes

$$E_{\text{Pol}}(z|p_1) = \begin{cases} 1 + (1 - p_1) E_{\text{Pol}}(z|\rho_{21}) & \text{if } p_1 \geq p_1^*, \\ 1 + p_1 E_{\text{Pol}}(z|\rho_{11}) & \text{if } p_1 \leq p_1^*, \end{cases} \quad (3.23)$$

with an arbitrary choice while the equivalence $p_1 = p_1^*$ holds.

Assume that $\rho_{21} \geq p_1^*$ and $\rho_{11} \geq p_1^*$, and consider the first line of Equation 3.23. If $p_1 = \rho_{21}$, then

$$E_{\text{Pol}}(z|\rho_{21}) = 1 + (1 - \rho_{21})E_{\text{Pol}}(z|\rho_{21}) = \frac{1}{\rho_{21}},$$

and, further, for $p_1 = \rho_{11}$, one obtains that

$$\begin{aligned}E_{\text{Pol}}(z|\rho_{11}) &= 1 + (1 - \rho_{11})E_{\text{Pol}}(z|\rho_{21}) = 1 + (1 - \rho_{11})\frac{1}{\rho_{21}} \\ &= 1 + \frac{1 - \rho_{11}}{\rho_{21}} = 1 + \frac{\rho_{12}}{\rho_{21}}.\end{aligned}$$

Substitution of these values into the definition of the probability p_1^* results in the following value:

$$p_1^* = \frac{E_{\text{Pol}}(z|\rho_{21})}{E_{\text{Pol}}(z|\rho_{21}) + E_{\text{Pol}}(z|\rho_{11})} = \frac{1/\rho_{21}}{1/\rho_{21} + (1 + \rho_{12}/\rho_{21})} = \frac{1}{1 + \rho_{12} + \rho_{21}}.$$

Finally, using the obtained values $E_{\text{Pol}}(z|\rho_{21})$, $E_{\text{Pol}}(z|\rho_{11})$, and p_1^* in the equality (Equation 3.23) one obtains that [20]:

1. If $\rho_{22} \leq \rho_{21}(\rho_{12} + \rho_{21})$ and $\rho_{21} \leq \rho_{12}(\rho_{12} + \rho_{21})$, then

$$E_{\text{Pol}}(z|p_1) = \begin{cases} 1 + \frac{1 - p_1}{\rho_{21}} & \text{if } p_1 \geq \frac{1}{1 + \rho_{12} + \rho_{21}}, \\ 1 + p_1 \left(1 + \frac{\rho_{12}}{\rho_{21}}\right) & \text{if } p_1 \leq \frac{1}{1 + \rho_{12} + \rho_{21}}. \end{cases}$$

Using the three remaining possibilities for the relations between p_1^* , and ρ_{11} and ρ_{21} , the other solutions can be obtained and are written as follows [20]:

2. If $\rho_{12} \leq \rho_{21}(\rho_{12} + \rho_{21})$ and $\rho_{21} \leq \rho_{12}(\rho_{12} + \rho_{21})$, then

$$E_{\text{Pol}}(z|p_1) = \begin{cases} 1 + \frac{1 - p_1}{\rho_{21}} & \text{if } p_1 \geq \frac{\rho_{12}}{\rho_{12} + \rho_{21}}, \\ 1 + \frac{p_1}{\rho_{21}} & \text{if } p_1 \leq \frac{\rho_{12}}{\rho_{12} + \rho_{21}}. \end{cases}$$

3. If $\rho_{22} \geq \rho_{21}(\rho_{12} + \rho_{21})$ and $\rho_{11} \geq \rho_{12}(\rho_{12} + \rho_{21})$, then

$$E_{\text{Pol}}(z|p_1) = \begin{cases} 1 + (1 - p_1) \frac{1 + \rho_{21}}{1 - \rho_{12}\rho_{21}} & \text{if } p_1 \geq \frac{1 + \rho_{21}}{2 + \rho_{12} + \rho_{21}}, \\ 1 + p_1 \frac{1 + \rho_{12}}{1 - \rho_{12}\rho_{21}} & \text{if } p_1 \leq \frac{1}{2 + \rho_{12} + \rho_{21}}. \end{cases}$$

4. If $\rho_{12} \geq \rho_{21}(\rho_{12} + \rho_{21})$ and $\rho_{11} \leq \rho_{12}(\rho_{12} + \rho_{21})$, then

$$E_{\text{Pol}}(z|p_1) = \begin{cases} 1 + (1 - p_1) \left(1 + \frac{\rho_{21}}{\rho_{12}}\right) & \text{if } p_1 \geq \frac{\rho_{12} + \rho_{21}}{1 + \rho_{12} + \rho_{21}}, \\ 1 + \frac{p_1}{\rho_{12}} & \text{if } p_1 \leq \frac{1}{1 + \rho_{12} + \rho_{21}}. \end{cases}$$

The solution completely defines the minimal expected number of observations for the case of perfect detection. From the formulas obtained it follows that the value of $E_{\text{Pol}}(z|p_1)$ is completely defined by the initial target location probabilities p_1 and $p_2 = 1 - p_1$ and transition probabilities ρ_{ij} , $i, j = 1, 2$.

Now let us consider the general case of imperfect detection.

Imperfect detection. In the case of imperfect detection, it is assumed that the detection probabilities ψ_1 and ψ_2 are arbitrary and can obtain any positive values for the unit interval, that is, $\psi_1, \psi_2 \in (0, 1]$. Hence, while the searcher observes point x_i , $i = 1, 2$, at which the target is located, the observation can fail and the search continues.

In contrast to the case of perfect detection, for arbitrary detection probabilities ψ_1 and ψ_2 , compact analytical solutions, which specify the value of the minimal expected number of observations $E_{\text{Pol}}(z|p_1)$, can be obtained. However, similar to the definition (Equation 3.22) of the detection probability $P_{\text{Pol}}(p_1|t)$, recurrent equations regarding the minimal expected number of observations $E_{\text{Pol}}(z|p_1, t)$ given the search period t , $t = 0, 1, 2, \dots$, can be obtained. Following Equation 3.21 above for $E_{\text{Pol}}(z|p_1)$, the recurrent equations for $E_{\text{Pol}}(z|p_1, t)$, $t = 0, 1, 2, \dots$, are formulated as follows [20]:

$$\begin{aligned} E_{\text{Pol}}(z|p_1, t) &= 1 + \min \begin{cases} (1 - p_1\psi_1) E_{\text{Pol}}(z|\tilde{p}_{1|1}^1, t-1) & \text{first observation of } x_1, \\ (1 - (1-p_1)\psi_2) E_{\text{Pol}}(z|\tilde{p}_{1|2}^1, t-1) & \text{first observation of } x_2, \end{cases} \\ E_{\text{Pol}}(z|p_1, 0) &= 1. \end{aligned} \quad (3.24)$$

Iterative calculations of the probability of detecting the target and of the expected number of observations are illustrated by the following example. A similar iterative procedure for specifying the threshold probabilities has been suggested by Schweitzer [21].

Example 3.3 Assume that at time $t = 0$ the probabilities of the target location at the points x_1 and x_2 are equivalent, that is, $p_1^0 = p^0(x_1) = 0.5$ and $p_2^0 = p^0(x_2) = 0.5$, and that the transition probability matrix is $\rho = \begin{pmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{pmatrix}$. Regarding the searcher's abilities, assume that the detection probabilities are $\psi_1 = 0.7$ and $\psi_2 = 0.5$. Our goal is to find the maximal probability of detecting the target $P_{\text{Pol}}(p_1|t)$ given time t , $t = 0, 1, 2, \dots$, and minimal expected number of observations $E_{\text{Pol}}(z|p_1)$. ■

Detection probability $P_{\text{Pol}}(p_1|t)$. We start by calculating the probability $P_{\text{Pol}}(p_1|t)$, and consider the first steps of the iterations in particular.

- **Step $t = 0$.** According to Equation 3.22, at the initial time $t = 0$, it is specified that $P_{\text{Pol}}(p_1|0) = 0$.
- **Step $t = 1$.** Using the recurrent equality (Equation 3.22), one obtains

$$\begin{aligned} P_{\text{Pol}}(p_1|1) &= \max \left\{ p_1\psi_1 + (1 - p_1\psi_1) P_{\text{Pol}}(\tilde{p}_{1|1}^0|0), (1 - p_1)\psi_2 + (1 - (1-p_1)\psi_2) P_{\text{Pol}}(\tilde{p}_{1|2}^0|0) \right\} \\ &= \max \left\{ \frac{p_1\psi_1}{(1-p_1)\psi_2} \right\} = \max \left\{ \frac{0.5 \times 0.7}{0.5 \times 0.5} \right\} = 0.35. \end{aligned}$$

- **Step $t = 2$.** In this step, the equality (Equation 3.22) is as follows:

$$P_{\text{Pol}}(p_1|2) = \max \left\{ \begin{array}{l} p_1\psi_1 + (1 - p_1\psi_1)P_{\text{Pol}}(\tilde{p}_{1|1}^1|1) \\ (1 - p_1)\psi_2 + (1 - (1 - p_1)\psi_2)P_{\text{Pol}}(\tilde{p}_{1|2}^1|1) \end{array} \right\}.$$

Let us calculate the values $P_{\text{Pol}}(\tilde{p}_{1|1}^1|1)$ and $P_{\text{Pol}}(\tilde{p}_{1|2}^1|1)$. According to Equation 3.20, one obtains

$$\tilde{p}_{1|1}^1 = \frac{p_1^0(1 - \psi_1)\rho_{11} + p_2^0\rho_{21}}{1 - \psi_1 p_1^0} = \frac{0.5 \times (1 - 0.7) \times 0.2 + 0.5 \times 0.6}{1 - 0.7 \times 0.5} = 0.5077,$$

$$\tilde{p}_{1|2}^1 = \frac{p_1^0\rho_{11} + p_2^0(1 - \psi_2)\rho_{21}}{1 - \psi_2 p_2^0} = \frac{0.5 \times 0.2 + 0.5 \times (1 - 0.5) \times 0.6}{1 - 0.5 \times 0.5} = 0.3333.$$

where it is taken into account that $p_1^0 = 0.5$ and $p_2^0 = 0.5$. Thus,

$$\begin{aligned} P_{\text{Pol}}(\tilde{p}_{1|1}^1|1) &= \max \left\{ \begin{array}{l} \tilde{p}_{1|1}^1\psi_1 + (1 - \tilde{p}_{1|1}^1\psi_1)P_{\text{Pol}}(\tilde{p}_{1|1}^0|0) \\ (1 - \tilde{p}_{1|1}^1)\psi_2 + (1 - (1 - \tilde{p}_{1|1}^1)\psi_2)P_{\text{Pol}}(\tilde{p}_{1|2}^0|0) \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} \tilde{p}_{1|1}^1\psi_1 \\ (1 - \tilde{p}_{1|1}^1)\psi_2 \end{array} \right\} = \max \left\{ \begin{array}{l} 0.5077 \times 0.7 \\ (1 - 0.5077) \times 0.5 \end{array} \right\} = 0.3554. \end{aligned}$$

$$\begin{aligned} P_{\text{Pol}}(\tilde{p}_{1|2}^1|1) &= \max \left\{ \begin{array}{l} \tilde{p}_{1|2}^1\psi_1 + (1 - \tilde{p}_{1|2}^1\psi_1)P_{\text{Pol}}(\tilde{p}_{1|1}^0|0) \\ (1 - \tilde{p}_{1|2}^1)\psi_2 + (1 - (1 - \tilde{p}_{1|2}^1)\psi_2)P_{\text{Pol}}(\tilde{p}_{1|2}^0|0) \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} \tilde{p}_{1|2}^1\psi_1 \\ (1 - \tilde{p}_{1|2}^1)\psi_2 \end{array} \right\} = \max \left\{ \begin{array}{l} 0.3333 \times 0.7 \\ (1 - 0.3333) \times 0.5 \end{array} \right\} = 0.3333. \end{aligned}$$

Substitution of these values into the equation that specifies $P_{\text{Pol}}(p_1|2)$ results in the following probability:

$$\begin{aligned} P_{\text{Pol}}(p_1|2) &= \max \left\{ \begin{array}{l} 0.5 \times 0.7 + (1 - 0.5 \times 0.7) \times 0.3554 \\ (1 - 0.5) \times 0.5 + (1 - (1 - 0.5) \times 0.5) \times 0.3333 \end{array} \right\} \\ &= 0.5810. \end{aligned}$$

Step $t = 3$. Similar to the previous steps, one obtains

$$\tilde{p}_{1|1}^2 = 0.3575, \tilde{p}_{1|2}^2 = 0.2418, P_{\text{Pol}}(\tilde{p}_{1|1}^2|2) = 0.5475, P_{\text{Pol}}(\tilde{p}_{1|2}^2|2) = 0.5861,$$

$$P_{\text{Pol}}(p_1|3) = \max \left\{ \begin{array}{l} p_1\psi_1 + (1 - p_1\psi_1)P_{\text{Pol}}(\tilde{p}_{1|1}^2|2) \\ (1 - p_1)\psi_2 + (1 - (1 - p_1)\psi_2)P_{\text{Pol}}(\tilde{p}_{1|2}^2|2) \end{array} \right\} = 0.7059.$$

Continuation of the process demonstrates that if, for example, $t = 5$, then $P_{\text{Pol}}(p_1|5) = 0.9114$ and at the step $t = 10$ it is $P_{\text{Pol}}(p_1|10) = 0.9969$; that is, when t increases, the maximal probability of detecting the target $P_{\text{Pol}}(p_1|t)$, $t = 0, 1, 2, \dots$, quickly converges to unity.

Observations number $E_{\text{Pol}}(z|p_1)$. Now let us calculate the minimal expected number of observations $E_{\text{Pol}}(z|p_1)$. As above, let us consider the first steps of the iterations.

Step $t = 0$. According to the second equation in the definition (Equation 3.24), at the initial time $t = 0$, it is specified that $E_{\text{Pol}}(z|p_1, 0) = 1$.

Step $t = 1$. Application of the recurrent Equation 3.24 results in the following value:

$$\begin{aligned} E_{\text{Pol}}(z|p_1, 1) &= 1 + \min \left\{ \begin{array}{l} (1 - p_1 \psi_1) E_{\text{Pol}}(z|\tilde{p}_{1|1}^1, 0) \\ (1 - (1 - p_1) \psi_2) E_{\text{Pol}}(z|\tilde{p}_{1|2}^1, 0) \end{array} \right\} \\ &= 1 + \min \left\{ \begin{array}{l} 1 - p_1 \psi_1 \\ 1 - (1 - p_1) \psi_2 \end{array} \right\} = 1 + \min \left\{ \begin{array}{l} 1 - 0.5 \times 0.7 \\ 1 - (1 - 0.5) \times 0.5 \end{array} \right\} = 1.65. \end{aligned}$$

Step $t = 2$. In this step, the equality (Equation 3.24) obtains the following form:

$$E_{\text{Pol}}(z|p_1, 2) = 1 + \min \left\{ \begin{array}{l} (1 - p_1 \psi_1) E_{\text{Pol}}(z|\tilde{p}_{1|1}^1, 1) \\ (1 - (1 - p_1) \psi_2) E_{\text{Pol}}(z|\tilde{p}_{1|2}^1, 1) \end{array} \right\},$$

and we need to calculate the values $E_{\text{Pol}}(z|\tilde{p}_{1|1}^1, 1)$ and $E_{\text{Pol}}(z|\tilde{p}_{1|2}^1, 1)$. Using the previously calculated probabilities $\tilde{p}_{1|1}^1 = 0.5077$ and $\tilde{p}_{1|2}^1 = 0.3333$, we obtain

$$\begin{aligned} E_{\text{Pol}}(z|\tilde{p}_{1|1}^1, 1) &= 1 + \min \left\{ \begin{array}{l} (1 - \tilde{p}_{1|1}^1 \psi_1) E_{\text{Pol}}(z|\tilde{p}_{1|1}^0, 0) \\ (1 - (1 - \tilde{p}_{1|1}^1) \psi_2) E_{\text{Pol}}(z|\tilde{p}_{1|2}^0, 0) \end{array} \right\} \\ &= 1 + \min \left\{ \begin{array}{l} 1 - \tilde{p}_{1|1}^1 \psi_1 \\ 1 - (1 - \tilde{p}_{1|1}^1) \psi_2 \end{array} \right\} = 1 + \min \left\{ \begin{array}{l} 1 - 0.5077 \times 0.7 \\ 1 - (1 - 0.5077) \times 0.5 \end{array} \right\} \\ &= 1.6446, \end{aligned}$$

$$\begin{aligned} E_{\text{Pol}}(z|\tilde{p}_{1|2}^1, 1) &= 1 + \min \left\{ \begin{array}{l} (1 - \tilde{p}_{1|2}^1 \psi_1) E_{\text{Pol}}(z|\tilde{p}_{1|1}^0, 0) \\ (1 - (1 - \tilde{p}_{1|2}^1) \psi_2) E_{\text{Pol}}(z|\tilde{p}_{1|2}^0, 0) \end{array} \right\} \\ &= 1 + \min \left\{ \begin{array}{l} 1 - \tilde{p}_{1|2}^1 \psi_1 \\ 1 - (1 - \tilde{p}_{1|2}^1) \psi_2 \end{array} \right\} = 1 + \min \left\{ \begin{array}{l} 1 - 0.3333 \times 0.7 \\ 1 - (1 - 0.3333) \times 0.5 \end{array} \right\} \\ &= 1.6666, \end{aligned}$$

and, finally, the expected number of steps $E_{\text{Pol}}(z|p_1, 2)$ is as follows:

$$E_{\text{Pol}}(z|p_1, 2) = 1 + \min \left\{ \begin{array}{l} (1 - 0.5 \times 0.7) \times 1.6446 \\ (1 - (1 - 0.5) \times 0.5) \times 1.6666 \end{array} \right\} = 2.069.$$

Step $t = 3$. In the third step, as above the calculations result in the following values:

$$\tilde{p}_{1|1}^2 = 0.3575, \quad \tilde{p}_{1|2}^2 = 0.2418, \quad E_{\text{Pol}}(z|\tilde{p}_{1|1}^2, 2) = 2.1313,$$

$$E_{\text{Pol}}(z|\tilde{p}_{1|2}^2, 2) = 2.0349,$$

$$E_{\text{Pol}}(z|p_1, 3) = 1 + \min \left\{ \frac{(1 - p_1)\psi_1) E_{\text{Pol}}(z|\tilde{p}_{1|1}^2, 2)}{(1 - (1 - p_1)\psi_2) E_{\text{Pol}}(z|\tilde{p}_{1|2}^2, 2)} \right\} = 2.3853.$$

Continuing the process for the next times allows one to obtain the approximate upper bound of the minimal expected number of observations $E_{\text{Pol}}(z|p_1)$. For small values of t the approximated value increases, for example, $E_{\text{Pol}}(z|p_1, 4) = 2.4583 > E_{\text{Pol}}(z|p_1, 3) = 2.3853$; however, for the next $t = 5$ the calculations result in $E_{\text{Pol}}(z|p_1, 5) = 2.4537 < E_{\text{Pol}}(z|p_1, 4) = 2.4583$. Then, the values of $E_{\text{Pol}}(z|p_1, t)$ continue decreasing with increasing t : $E_{\text{Pol}}(z|p_1, 10) = 2.3216$, $E_{\text{Pol}}(z|p_1, 20) = 2.3001$, and so on, converging to the value $E_{\text{Pol}}(z|p_1) \approx 2.3$.

The Pollock model above illustrates the MDP and POMDP models of search, and, as well as the continuous time Dobbie two-cell model of search [22], forms a basis for different search algorithms. In this model, the efficiency of the search is characterized by the expected number of observations, which is considered as a cost function.

Similar techniques are applied below for a search over the sample space with an arbitrary number of points, and in the next sections the Pollock model will be used for analyzing the general informational algorithm of search.

3.2.3 Ross model with single-point observations

The Pollock simple model of search provides a straightforward illustration of the MDP and POMDP models of search applied to an artificial problem of detecting the target in two-point sample space. Now let us consider an application of a similar decision model to the search in a sample space with an arbitrary number of points. Such a model has been constructed by Ross [23, 24] and then applied for a moving target search (MTS) by White [4]. Note that this model differs from the Pollock model; the differences between them will be indicated in particular.

Let, as indicated above, $X = \{x_1, x_2, \dots, x_n\}$ be a sample space with target location probabilities $p_i^t = p^t(x_i)$, $0 \leq p^t(x_i) \leq 1$, $i = 1, 2, \dots, n$, and $\sum_{i=1}^n p^t(x_i) = 1$, $t = 0, 1, 2, \dots$, and assume that the target moves over the sample space X according to certain Markov process with transition probability matrix $\rho = [\rho_{ij}]_{n \times n}$, where $\sum_{j=1}^n \rho_{ij} = 1$ for each $i = 1, 2, \dots, n$.

As in the Pollock model of search, it is assumed that the searcher observes single-point areas $A \subset X$, that is, a search space \mathcal{X} consists of n sets $A = \{x\}$, $x \in X$. As indicated above, by ψ_i , $i = 1, 2, \dots, n$, we denote the probability of detecting the target at the point x_i , while it is located at this point and the searcher observes it, that is, $\psi_i = z(\{x_i\}, x_i)$. Since the considered sample space includes $n \geq 2$ points, as above (see Sections 2.1.1 and 3.1.2) it is convenient to define the posterior observed location probabilities \bar{p}_i^t and estimated location probabilities \tilde{p}_i^t , $i = 1, 2, \dots, n$, $t = 0, 1, 2, \dots$, while at the initial time $t = 0$ it follows that $\tilde{p}_i^0 = \bar{p}_i^0 = p_i^0$.

In addition, similar to the group-testing search procedures (see Section 2.2.3), denote by $c_i = c(x_i) \geq 0$, $i = 1, 2, \dots, n$, a cost of observation of the observed area $A = \{x_i\}$ or, the same thing, of the point x_i . Denote by $\tilde{p}^t = (\tilde{p}_1^t, \tilde{p}_2^t, \dots, \tilde{p}_n^t)$ a vector of posterior estimated

location probabilities, and let $\tilde{p}_{j|i}^t = \tilde{p}^t(x_j|z(\{x_i\}, x^{t-1}) = 0)$, $i, j = 1, 2, \dots, n$, be the probability of the target location at the point x_j given that observation of the point x_i failed.

Consider the search for a *static target*. By using the detection probabilities ψ_i , the probability $\tilde{p}_{j|i}^t$ is calculated as follows:

$$\tilde{p}_{j|i}^t = \tilde{p}^t(x_j|z(\{x_i\}, x^{t-1}) = 0) = \begin{cases} \frac{\tilde{p}_j^{t-1}}{1 - \psi_i \tilde{p}_i^{t-1}} & \text{if } j \neq i, \\ \frac{(1 - \psi_i) \tilde{p}_i^{t-1}}{1 - \psi_i \tilde{p}_i^{t-1}} & \text{if } j = i. \end{cases} \quad (3.25)$$

where $i, j = 1, 2, \dots, n$ and $t = 0, 1, 2, \dots$. Note that these probabilities have the same meaning as those defined by Equations 3.9 and 3.10, which were applied in the MDP model of search. Implementation of these formulas in the two-point sample space and to a static target results in the Pollock formulas (Equation 3.20).

Denote by U_S the transformation which corresponds to the static target search and is specified by Equation 3.25; that is, given a probability vector \tilde{p} and a number i , $i = 1, 2, \dots, n$, an operator U_S results in a probability vector $U_S(\tilde{p}, i) = (\tilde{p}_{1|i}, \tilde{p}_{2|i}, \dots, \tilde{p}_{n|i})$, where $\tilde{p}_{j|i}$, $i, j = 1, 2, \dots, n$, are calculated according to Equation 3.25.

Let $E_{\text{Ross}}(C|\tilde{p})$ be the minimal expected cost, which is required to pay for detecting the target given the vector \tilde{p} of posterior estimated location probabilities. Note that in the Pollock model, the observation costs $c_1 = c_2 = 1$; thus the minimal expected number of observations $E_{\text{Pol}}(z|p_1)$ has the same meaning as minimal expected cost. Backwards, if $c_i = 1$, $i = 1, 2, \dots, n$, then in the Ross model the value $E_{\text{Ross}}(C|\tilde{p})$ will represent a minimal expected number of observations. Similar to the first line of the Pollock equation (Equation 3.21), the minimal expected cost $E_{\text{Ross}}(C|\tilde{p})$ in the Ross model is defined as follows [24]:

$$E_{\text{Ross}}(C|\tilde{p}) = \min_{i=1,\dots,n} \{c_i + (1 - \tilde{p}_i \psi_i) E_{\text{Ross}}(C|U_S(\tilde{p}, i))\}, \quad (3.26)$$

with initial probability vector $\tilde{p} = \tilde{p}^0 = (\tilde{p}_1^0, \tilde{p}_2^0, \dots, \tilde{p}_n^0)$. Notice again that the recursive Equation 3.26 holds true for the static target search.

The optimal policy, which solves the minimization in Equation 3.26, is similar to the above policy determined by Equation 3.8, and the strategy is to choose and observe a point x_i , $i = 1, 2, \dots, n$, such that the value $\tilde{p}_i \psi_i / c_i$ reaches its maximum [24].

Now let us consider the Ross model for the MTS. As required by the Pollock model, consider the search over a sample space $X = \{x_1, x_2\}$, while the target moves according to the Markov process with transition probability matrix $\rho = \|\rho_{ij}\|_{2 \times 2}$. According to Equation 3.20, two operators U_1 and U_2 are defined as follows:

$$U_1(p_1) = \frac{p_1(1 - \psi_1)\rho_{11} + (1 - p_1)\rho_{21}}{1 - \psi_1 p_1} \quad \text{and} \quad U_2(p_1) = \frac{p_1\rho_{11} + (1 - p_1)(1 - \psi_2)\rho_{21}}{1 - \psi_2(1 - p_1)}. \quad (3.27)$$

Then, the minimal expected cost for the MTS is defined by the following recursion [24]:

$$E_{\text{Ross}}(C|p_1) = \min \begin{cases} c_1 + (1 - p_1 \psi_1) E_{\text{Ross}}(C|U_1(p_1)), \\ c_2 + (1 - (1 - p_1) \psi_2) E_{\text{Ross}}(C|U_2(p_1)). \end{cases} \quad (3.28)$$

It is clear that, for $c_1 = c_2 = 1$, Equation 3.27 is equivalent to the Pollock equation (Equation 3.21).

Let us demonstrate that the myopic policy, which implements the maximum probability criterion, is not optimal.

Example 3.4 [24] At the beginning of the search, let the probabilities of the target location at points x_1 and x_2 be $p_1^0 = 0.55$ and $p_2^0 = 0.45$, and let the transition probability matrix be $\rho = \begin{pmatrix} 0 & 1 \\ 0.5 & 0.5 \end{pmatrix}$. Assume that the costs are equivalent and $c_1 = c_2 = 1$, and that the detection probabilities are $\psi_1 = \psi_2 = 1$; that is, if the searcher checks the point at which the target is located, then the target will be found with certainty.

Following the myopic policy, the searcher immediately checks the point x_1 . If the search at the point x_1 is successful, then the search terminates. Assume that the search fails; that is, the target is at the point x_2 . Then, according to the transition probabilities, the target location probabilities at the next search step are $p_1^1 = 0.5$ and $p_2^1 = 0.5$, and, according to Equation 3.28, the expected cost at best is equal to

$$\begin{aligned} E_{\text{Ross}}(C|0.55) &= \min \left\{ \frac{1 + (1 - 0.55) E_{\text{Ross}}(C|U_1(0.55))}{1 + (1 - (1 - 0.55)) E_{\text{Ross}}(C|U_2(0.55))} \right\} \\ &= \min \left\{ \frac{1 + 0.45 \times (1 + 0.5)}{1 + 0.55 \times (1 + 0.5)} \right\} = 1 + 0.45 \times 1.5 = 1.675. \end{aligned}$$

Now assume that, in contrast to the myopic policy, the searcher starts with the point x_2 . As above, if the check is successful, then the search terminates. If, however, the check fails and the target is at the point x_1 , then the target will be certainly found at the next step since, according to the transition probabilities, $p_1^1 = 0$ and $p_2^1 = 1$. Thus the first check of the point x_2 results in an expected cost equal to 1.55, which is better than the calculated value of 1.675. ■

Example 3.4 demonstrates the necessity of the threshold probability and provides a basis for the policy, which is applied in the Pollock model.

Let us use the considered Ross model for a search over the sample space $X = \{x_1, x_2, \dots, x_n\}$ with an arbitrary number of points $n \geq 2$ [4]. Recall that the searcher is allowed to check only single-point areas $A = \{x\}, x \in X$. Hence, the observation results $z(\{x_k\}, x_j)$, $k, j = 1, 2, \dots, n$, define the detection probabilities as follows:

$$\psi_k = \begin{cases} z(\{x_k\}, x_j) & \text{if } k = j, \\ 0 & \text{otherwise,} \end{cases}$$

where $0 < z(\{x_k\}, x_j) < 1$ while $k = j$. By using the defined n detection probabilities, n operators U_k , $k = 1, 2, \dots, n$, are implemented, acting over the probability vectors $\tilde{p} = (\tilde{p}_1, \tilde{p}_2, \dots, \tilde{p}_n)$. Given the transition probability matrix $\rho = \|\rho_{ij}\|_{n \times n}$, operators U_k specify the probability vectors $U_k(\tilde{p}) = (\tilde{p}_{1|k}, \tilde{p}_{2|k}, \dots, \tilde{p}_{n|k})$ as follows [4]:

$$\tilde{p}_{i|k} = \frac{1}{1 - \psi_k \tilde{p}_k} \left(\sum_{j=1}^n \tilde{p}_j \rho_{ji} - \psi_k \tilde{p}_k \rho_{ki} \right). \quad (3.29)$$

It is clear that for $n = 2$ Equation 3.29 is equal to Equation 3.27, which corresponds to the operators U_1 and U_2 , and for the unit matrix ρ , Equation 3.29 specifies the probabilities (Equation 3.25) and operator U_S , as required for the static target search.

Similar to the two-point case, we can formulate the minimization problem of the expected cost $E_{\text{Ross}}(C|\tilde{p})$, while an observation of the single-point area $\{x_k\}$ has a positive cost $c_k = c(x_k) \geq 0$, $k = 1, 2, \dots, n$. The minimum expected cost $E_{\text{Ross}}(C|\tilde{p})$ is defined by the following recursive equations [4]:

$$E_{\text{Ross}}(C|\tilde{p}) = \min_{k=1,\dots,n} \{c_k + (1 - \tilde{p}_k \psi_k) E_{\text{Ross}}(C|U_k(\tilde{p}))\}, \quad (3.30)$$

and $E_{\text{Ross}}(C|\tilde{p}) = 0$ if in the vectors $\tilde{p} = (\tilde{p}_1, \tilde{p}_2, \dots, \tilde{p}_n)$ for some i it holds true that $\tilde{p}_i = 1$.

The vectors $\tilde{p} = (\tilde{p}_1, \tilde{p}_2, \dots, \tilde{p}_n)$, in which $\tilde{p}_i = 1$ for some i and so $\tilde{p}_j = 0$, $j \neq i$, $i, j = 1, 2, \dots, n$, correspond to the absorbing states for the MDP of search and represent a certain knowledge regarding the target location at the point x_i .

The actions of the Ross model are illustrated by the following example. In this example, we implement the same step-by-step techniques as presented by Equation 3.24 for the Pollock model and used in Example 3.3.

Example 3.5 Similar to Example 3.1, let the sample space consist of four points $X = \{x_1, x_2, x_3, x_4\}$ with initial location probabilities

$$p^0(x_1) = 0.1, p^0(x_2) = 0.2, p^0(x_3) = 0.3, \text{ and } p^0(x_4) = 0.4,$$

and, as above, let $\tilde{p}^0(x_i) = p^0(x_i)$, $i = 1, \dots, 4$. As above, we assume that the movements of the target are governed by a Markov process with the transition probability matrix $\rho = \|\rho_{ij}\|_{4 \times 4}$, and consider two matrices:

$$\rho_{\text{Static}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \rho_{\text{Brown}} = \begin{pmatrix} 0 & 0.4 & 0.4 & 0.2 \\ 0.4 & 0 & 0.2 & 0.4 \\ 0.4 & 0.2 & 0 & 0.4 \\ 0.2 & 0.4 & 0.4 & 0 \end{pmatrix}.$$

The first matrix corresponds to the static target and the second to the Brownian target.

Following the Ross model, the searcher is allowed to check single-point areas; in terms of Example 3.1 this means that the observed areas are $A_1 = \{x_1\}$, $A_2 = \{x_2\}$, $A_3 = \{x_3\}$, and $A_4 = \{x_4\}$, or, essentially the same, the points x_i , $i = 1, \dots, 4$. For each point x_i we define the detection probability ψ_i , $i = 1, \dots, 4$, and assume that the probabilities are equal. In the example, we consider two cases: the case of perfect detection with $\psi_i = 1$; and the case of imperfect detection with $\psi_i = 0.7$, $i = 1, \dots, 4$. The check of each point x_i is characterized by the cost $c_i = 1$, $i = 1, \dots, 4$, so the minimal expected number cost $E_{\text{Ross}}(C|\tilde{p})$ represents the minimal expected number of checks.

The calculations of the minimal expected cost $E_{\text{Ross}}(C|\tilde{p})$ are similar to the calculations conducted in Example 3.3 according to Equation 3.30. In the Ross model, this equation is written as follows:

$$E_{\text{Ross}}(C|\tilde{p}, t) = \min_{k=1,\dots,n} \{c_k + (1 - \tilde{p}_k \psi_k) E_{\text{Ross}}(C|U_k(\tilde{p}), t-1)\},$$

$$E_{\text{Ross}}(C|\tilde{p}, 0) = 1.$$

Assume that the target is static, that is, $\rho = \rho_{\text{Static}}$, and that the detection is perfect, that is, $\psi_i = 1$, $i = 1, \dots, 4$. Then, direct calculation of the minimal expected cost $E_{\text{Ross}}(C|\tilde{p}, t)$ results in the following values:

ρ_{Static}	$\psi_i = 1$	t	1	2	3	4	5	6	7	8
		$E_{Ross}(C \tilde{p}, t)$	1.6	1.9	2.0	1.9	1.9	1.9	1.9	1.9

It can be seen that the process quickly converges to the minimal expected cost $E_{Ross}(C|\tilde{p}, t) = 1.9$, and that for the costs $c_i = 1$ and detection probabilities $\psi_i = 1$, $i = 1, \dots, 4$, this value represents the minimal expected number of checks.

Let us compare the obtained value $E_{Ross}(C|\tilde{p}, t) = 1.9$ to the minimal expected number of checks, which is obtained by using the Huffman–Zimmermann procedure (see Theorem 2.7). An optimal binary search tree, which is obtained by use of the Huffman–Zimmermann procedure over the location probabilities $p^0(x_i)$, $i = 1, \dots, 4$, is shown in Figure 3.5.

According to the tree, an average number of the lengths $l(x_i)$ from the root node to the leaves x_i , $i = 1, \dots, 4$, gives the following value:

$$L = \sum_{i=1}^n p_i l(x_i) = 0.1 \times 3 + 0.2 \times 3 + 0.3 \times 2 + 0.4 \times 1 = 1.9.$$

As expected, this value is equal to the obtained value $E_{Ross}(C|\tilde{p}, t) = 1.9$ for the minimal expected cost.

Let us obtain values of the minimal expected cost $E_{Ross}(C|\tilde{p}, t)$ for the static target $\rho = \rho_{Static}$, while detection is imperfect with detection probabilities $\psi_i = 0.7$, $i = 1, \dots, 4$. Calculations of $E_{Ross}(C|\tilde{p}, t)$ for the series of t give the following results:

ρ_{Static}	$\psi_i = 0.7$	t	1	2	3	4	5	6	7	8
		$E_{Ross}(C \tilde{p}, t)$	1.72	2.23	2.60	2.89	3.10	3.26	3.37	3.45

The value of $E_{Ross}(C|\tilde{p}, t)$ increases with t and converges to a value ~ 4.0 ; in particular $E_{Ross}(C|\tilde{p}, 9) = 3.517$, $E_{Ross}(C|\tilde{p}, 10) = 3.563$, and $E_{Ross}(C|\tilde{p}, 11) = 3.596$. However, the time and space required for calculations increases exponentially. Note that in the case of imperfect detections, the Huffman–Zimmermann procedure is not applicable.

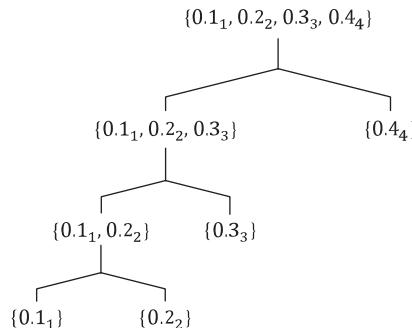


Figure 3.5 Optimal binary search tree for initial target location probabilities.

Now let us calculate the minimal expected costs for the perfect and imperfect detections, while the target is moving. For perfect detections with detection probabilities $\psi_i = 1$, $i = 1, \dots, 4$, the results are as follows:

ρ_{Brown}	$\psi_i = 1$	t	1	2	3	4	5	6	7	8
	$E_{Ross}(C \tilde{p}, t)$		1.60	1.98	2.25	2.42	2.54	2.62	2.68	2.72

As above, the value of $E_{Ross}(C|\tilde{p}, t)$ increases with t and converges to a value ~ 3.0 . In particular, $E_{Ross}(C|\tilde{p}, 9) = 2.740$, $E_{Ross}(C|\tilde{p}, 10) = 2.757$, and $E_{Ross}(C|\tilde{p}, 11) = 2.768$.

Similarly, for the imperfect detections with detection probabilities $\psi_i = 0.7$, $i = 1, \dots, 4$, the following values are obtained for the expected costs $E_{Ross}(C|\tilde{p}, t)$:

ρ_{Brown}	$\psi_i = 0.7$	t	1	2	3	4	5	6	7	8
	$E_{Ross}(C \tilde{p}, t)$		1.72	2.28	2.74	3.10	3.38	3.61	3.79	3.93

Further calculations demonstrate the increasing value of $E_{Ross}(C|\tilde{p}, t)$ with t and its convergence to a value ~ 5.0 . In particular, $E_{Ross}(C|\tilde{p}, 9) = 4.049$, $E_{Ross}(C|\tilde{p}, 10) = 4.140$, and $E_{Ross}(C|\tilde{p}, 11) = 4.212$. As in the case of the static target, for both perfect and imperfect detections the required time and space for the calculations increases exponentially with t .

A comparison of the obtained results demonstrates that, as expected, the minimal expected cost for the search with imperfect detections is greater than the cost for the search with perfect detections. However, as can be seen from the results, imperfections in the detections in the case of a static target search have a greater influence on the expected cost than in the case of MTS. In particular, at $t = 11$ the proportion between the costs for the static target search is $3.596/1.9 = 1.893$, while the proportion between the costs for MTS is $4.212/2.768 = 1.522$; the difference between the values increases with t and converges to the value provided by the exact values of the minimal expected costs. ■

In the Ross model of search, the probabilities are updated in similar manner as defined by Theorem 2.1 for the search and screening methods (see Section 2.1.2), and the model provides an optimal solution for the search problem while the searcher is allowed to observe the single-point areas $A = \{x\}$, $x \in X$. However, as in any other model that implements a dynamic programming approach, the Ross model requires exponential computation time and space. Below, we consider certain models that in particular cases allow a search by the use of arbitrary observed areas and require fewer computations.

3.3 Models of moving target search with constrained paths

Two models of search were considered above. The first general MDP model allows observation of the arbitrary areas of the sample space and implements heuristic information theory

measures for decision making. The second POMDP model implements the dynamic programming method for creating an unconstrained sequence of single-point observed areas. Now let us consider the models of search when the searcher's path over the search space is constrained and the observed areas can include an arbitrary number of points. These models generalize the algorithms of search, which were introduced in Section 2.1.3.

3.3.1 Eagle model with finite and infinite horizons

The Ross model presented above generalizes the Pollock model of search to the sample space with an arbitrary number of points and considers the optimization problem regarding minimal expected cost. Below, we present the model that deals with a similar generalization of the Pollock model with respect to the maximum probability of detecting the target. Such a generalization was provided by Eagle [25].

As in the previous considerations, let $X = \{x_1, x_2, \dots, x_n\}$ be a sample space and $\tilde{p}^t(x_i)$, $t = 0, 1, 2, \dots$, be estimated location probabilities such that $\tilde{p}^{t=0}(x_i) = p^{t=0}(x_i)$, $i = 1, 2, \dots, n$, where $p^t(x_i)$ are the target location probabilities. In other words, $\tilde{p}^t(x_i)$ is the probability that the target is located at the point x_i at time t before the observation at this time (see Figure 3.2). The detections are allowed to be uncertain and ψ_i , $i = 1, 2, \dots, n$, stands for the probability of detecting the target at the point x_i , while it is located at this point and the searcher observes it. Also, recall that in the considered search, the target is governed by a Markov process with transition probability matrix $\rho = [\rho_{ij}]_{n \times n}$.

Similar to the Ross model, let U_k be an operator that specifies the updated probabilities after an unsuccessful search at the point k , $k = 1, 2, \dots, n$. Similar to Equation 3.29, the operators U_k act over the probability vectors $\tilde{p} = (\tilde{p}_1, \tilde{p}_2, \dots, \tilde{p}_n)$ and result in the probability vectors $U_k(\tilde{p}) = (\tilde{p}_{1|k}, \tilde{p}_{2|k}, \dots, \tilde{p}_{n|k})$, where $\tilde{p}_{i|k}$, $i, k = 1, 2, \dots, n$, are defined by Equation 3.29. For brevity, such operators are specified as follows [25]:

$$U_k(\tilde{p}) = \frac{1}{1 - \psi_k \tilde{p}_k} \tilde{p} \times \rho_{(k)}, \quad \psi_k \tilde{p}_k < 1, \quad (3.31)$$

where $\rho_{(k)}$ stands for the transition probability matrix ρ , in which the k th row is multiplied by $(1 - \psi_k)$, $k = 1, 2, \dots, n$. Similar to the Ross model, it is assumed that if for some k it follows that $\psi_k \tilde{p}_k = 1$, then the search terminates, and the target is found at the point x_k .

The optimization problem regarding maximum detection probability in the Eagle model is formulated as follows. Let $P_{\text{Eagle}}(\tilde{p}|t)$ be the maximum probability of detecting the target up to time t , $t = 0, 1, 2, \dots$. Then the recurrent equations regarding $P_{\text{Eagle}}(\tilde{p}|t)$ are formulated in the usual manner for dynamic programming [25]:

$$\begin{aligned} P_{\text{Eagle}}(\tilde{p}|t) &= \max_{k=1, \dots, n} \{\psi_k \tilde{p}_k + (1 - \psi_k \tilde{p}_k) P_{\text{Eagle}}(U_k(\tilde{p})|t-1)\}, \\ P_{\text{Eagle}}(\tilde{p}|0) &= 0. \end{aligned} \quad (3.32)$$

These equations are simple generalizations of Equation 3.22 considered by the Pollock model and follow the line of the Ross model equations, which were presented in Example 3.5.

According to Equation 3.32, at each step a chosen operator U_k updates the probability vector \tilde{p}^{t-1} so that a new vector \tilde{p}^t is obtained. Consequently, the process can be considered as an optimization problem over the set \mathcal{P} of probability vectors $\tilde{p} = (\tilde{p}_1, \tilde{p}_2, \dots, \tilde{p}_n)$.

Moreover, since operator U_k decreases the probability at the point x_k and the next step operator is chosen according to the maximization criterion, the set of probability vectors, which is considered at the step t , depends on the previous choice. Hence, the choice of the point x_k with further probability updating *constrains* the set \mathcal{P} so that, at the next step, the probability vectors are chosen from the set $\mathcal{P}(t) \subset \mathcal{P}$. Such an observation allows the optimization Equation 3.32 to be reformulated, as given by the next lemma.

Lemma 3.6 [25]. *For a finite t , $t = 0, 1, 2, \dots$, it follows that*

$$P_{\text{Eagle}}(\tilde{p}|t) = \max_{b \in \mathcal{B}(t)} \{\tilde{p} \times b\}, \quad (3.33)$$

where $\mathcal{B}(t)$ is a finite set of real vectors $b = (b_1, b_2, \dots, b_n)$ and $\mathcal{B}(0) = \{(0, 0, \dots, 0)\}$ such that $(\tilde{p} \times b) \in \mathcal{P}(t)$ while $b \in \mathcal{B}(t)$.

Proof. If $t = 0$, then $b = (0, 0, \dots, 0)$ and $P_{\text{Eagle}}(\tilde{p}|0) = 0$, as required by the second equation in Equation 3.32.

Let $t = 1$. Then from Equation 3.32 it follows that

$$\begin{aligned} P_{\text{Eagle}}(\tilde{p}|t) &= \max_{k=1,\dots,n} \{\psi_k \tilde{p}_k + (1 - \psi_k \tilde{p}_k) P_{\text{Eagle}}(U_k(\tilde{p})|t-1)\} \\ &= \max_{k=1,\dots,n} \{\psi_k \tilde{p}_k\} = \max_{b \in \mathcal{B}(1)} \{\tilde{p} \times b\}, \end{aligned}$$

where $\mathcal{B}(1) = \{\psi_k b_{(k)}\}$, $k = 1, 2, \dots, n$, and $b_{(k)} = (b_1, b_2, \dots, b_k, \dots, b_n)$ is a vector such that $b_k = 1$ and $b_i = 0$, $i \neq k$, $i = 1, 2, \dots, n$.

By induction, assume that Equation 3.33 holds up to time $t - 1$ and consider the moment t . By the use of Equation 3.32 and the updating rule (Equation 3.31) one obtains

$$\begin{aligned} P_{\text{Eagle}}(\tilde{p}|t) &= \max_{k=1,\dots,n} \{\psi_k \tilde{p}_k + (1 - \psi_k \tilde{p}_k) P_{\text{Eagle}}(U_k(\tilde{p})|t-1)\} \\ &= \max_{k=1,\dots,n} \{\psi_k \tilde{p}_k + (1 - \psi_k \tilde{p}_k) \max_{b' \in \mathcal{B}(t-1)} \{U_k(\tilde{p}) \times b'\}\} \\ &= \max_{k=1,\dots,n} \left\{ \psi_k \tilde{p}_k + (1 - \psi_k \tilde{p}_k) \max_{b' \in \mathcal{B}(t-1)} \left\{ \frac{\tilde{p} \times \rho_{(k)}}{1 - \psi_k \tilde{p}_k} b' \right\} \right\} \\ &= \max_{k=1,\dots,n, b' \in \mathcal{B}(t-1)} \{\psi_k \tilde{p}_k + \tilde{p} \times \rho_{(k)} \times b'\} \\ &= \max_{b \in \mathcal{B}(t)} \{\tilde{p} \times b\}, \end{aligned}$$

where $\mathcal{B}(t) = \{b | b = \psi_k b_{(k)} + \rho_{(k)} \times b'\}$ while $b' \in \mathcal{B}(t-1)$, $k = 1, 2, \dots, n$, and, as above, $\rho_{(k)}$ stands for the transition probability matrix ρ with the k th row multiplied by $(1 - \psi_k)$ and $b_{(k)}$ is a vector with unity in the k th place and zeros elsewhere. ■

With the use of Lemma 3.6 the optimization problem of choosing the point that maximizes the probability $P_{\text{Eagle}}(\tilde{p}|t)$ of detecting the target is formulated as a problem of finding sets $\mathcal{B}(t)$, $t = 0, 1, 2, \dots$, such that they include a minimum of possible vectors, which provide an optimal solution. Such a minimization problem is formulated in the form of a linear programming problem as follows [25].

The vector $b' \in \mathcal{B}(t)$ is said to be dominated in $\mathcal{B}(t)$ if for every probability vector \tilde{p} it follows that $\max_{b \in \mathcal{B}(t) \setminus \{b'\}} \{\tilde{p} \times b\} = \max_{b \in \mathcal{B}(t)} \{\tilde{p} \times b\}$; that is, if the vector b' does not change as a result of the maximization problem. Thus, such dominated vectors can

be removed from the set $\mathcal{B}(t)$. To find these vectors, at each step t the following linear programming problem is solved:

$$(\boldsymbol{r} - \tilde{\boldsymbol{p}} \times \boldsymbol{b}') \rightarrow \min \text{ over all } \tilde{\boldsymbol{p}} \in \mathcal{P} \text{ and some positive constant } \boldsymbol{r}, \quad (3.34)$$

$$\text{subject to } \boldsymbol{r} - \tilde{\boldsymbol{p}} \times \boldsymbol{b} \geq 0 \text{ for all } \boldsymbol{b} \in \mathcal{B}(t) \setminus \{\boldsymbol{b}'\}. \quad (3.35)$$

As a result, the problem of search is formulated as a sequence of linear programming problems with the maximization over the resulting finite sets of vectors. Note that the consideration of the Eagle model above deals with observations of the points $x \in X$, or, essentially the same, of the single-point observed areas $A = \{x\} \subset X$. However, because of the minimization (Equations 3.34 and 3.35) there are no restrictions regarding the searcher's abilities, and observed areas with an arbitrary number of points with a suitable formulation of the updating operators U are also allowed.

Let us consider a model of search with constrained multi-point observed areas and formulate an infinite horizon optimization problem, as suggested by Singh and Krishnamurthy [26] on the basis of the Eagle model above [25]. This consideration follows the notation that was introduced for the general MDP and POMDP models (see Sections 3.1.1 and 3.2.1) with certain additions.

Let $\hat{X} = \{x_1, x_2, \dots, x_n\} \cup \{x_{n+1}\}$ be a sample space which includes the target locations x_1, x_2, \dots, x_n and, in addition, equipped with a fictive absorbing point x_{n+1} which corresponds to the termination of the search upon detection of the target. The target starts with the initial probabilities $p^{t=0}(x_i) = p^{t=0}(x_i) = \Pr\{x^{t=0} = x_i\}$, $i = 1, 2, \dots, n$, $p^{t=0}(x_{n+1}) = 0$, and moves over \hat{X} according to the Markov process with transition probability matrix $\rho = [\rho_{ij}]_{(n+1) \times (n+1)}$, where $\rho_{ij} = \Pr\{x^{t+1} = x_j | x^t = x_i\}$ while $i, j = 1, 2, \dots, n$, $\rho_{i(n+1)} = 0$ for $i < n + 1$, $\rho_{(n+1)j} = 0$ for $j < n + 1$, and $\rho_{(n+1)(n+1)} = 1$, $t = 0, 1, 2, \dots$. In other words, if $\rho = [\rho_{ij}]_{n \times n}$ is a target transition probability matrix over the usual sample space X , then $\rho = \begin{pmatrix} \rho & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix}$, where $\mathbf{0}$ stands for the column and row vectors of zeros. As a result, the target can move over the states x_1, x_2, \dots, x_n and cannot enter the absorbing point x_{n+1} , while the transition to the point x_{n+1} occurs if the target is detected by the searcher.

As above, assume that the actions of the searcher are represented by the choices and observations of the observed areas $A \subset X$, which are selected from the power set $\mathcal{X} = 2^X$ or its certain subset. Note that in such a case, at each time $t = 0, 1, 2, \dots$ the searcher is allowed to choose an area $A^t \in \mathcal{X}$ which includes a single point or a number of points, that is, $1 \leq |A^t| \leq n$. If an observation of the chosen area A^t is possible and it results in finding the target, then the target moves to the absorbing point x_{n+1} and the search *terminates*. If, in contrast, the target is not found by observation of the area A^t , then the target moves to its next location $x^{t+1} \in X \setminus \{x_{n+1}\}$ and the search *continues*. In addition, the situation is defined where, in the current time t , the area A^t chosen by the searcher cannot be observed or the probabilities cannot be updated according to the observation. In such a case the area is specified as *blocked*. Following the notation of group-testing search (see Section 2.2.1), the indicated situations are combined into the set of controls $\mathfrak{C} = \{\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_3\}$, where $\mathfrak{c}_1 = \text{terminate}$, $\mathfrak{c}_2 = \text{continue}$, and $\mathfrak{c}_3 = \text{block}$, so at each time t , $t = 0, 1, 2, \dots$, the value of control is $\mathfrak{c}^t \in \mathfrak{C}$.

In contrast to the POMDP model, assume that each available area $A_k \in \mathcal{X}$, $k = 1, 2, \dots, N \leq 2^n$, is chosen with a constant probability $\mathfrak{d}_k = \mathfrak{d}(A_k) = \Pr\{A = A_k\}$,

and let $\psi_k = \psi(A_k) = \Pr\{\text{target is detected} | A = A_k\}$ be the probability of detecting the target by observation of the area A_k . Then, the search process is conducted on the basis of the following probabilities [26]:

- probability of search termination at time t , $t = 0, 1, 2, \dots$:

$$p_{ik}^t(\mathbf{c}_1) = \Pr\{\mathbf{c}^t = \mathbf{c}_1 | x^t = x_i, A^t = A_k\} = \begin{cases} \mathfrak{d}(A_k) \times \psi(A_k) & \text{if } x_i \in A_k, \\ 0 & \text{otherwise,} \end{cases}$$

$$p_{(n+1)k}^t(\mathbf{c}_1) = \Pr\{\mathbf{c}^t = \mathbf{c}_1 | x^t = x_{n+1}, A^t = A_k\} = 1,$$

where $\mathbf{c}_1 = \text{terminate}$, x_{n+1} is an absorbing point, x^t is the target location, and A^t is an observed area chosen by the searcher;

- probability of continuing the search after an observation at time t :

$$p_{ik}^t(\mathbf{c}_2) = \Pr\{\mathbf{c}^t = \mathbf{c}_2 | x^t = x_i, A^t = A_k\} = \begin{cases} \mathfrak{d}(A_k) \times (1 - \psi(A_k)) & \text{if } x_i \in A_k, \\ 1 - \mathfrak{d}(A_k) & \text{otherwise,} \end{cases}$$

where $\mathbf{c}_2 = \text{continue}$, x^t is the target location from the set $X \setminus \{x_{n+1}\}$, and A^t is an observed area chosen by the searcher;

- probability of obtaining a blocked observed area at time t :

$$p_{ik}^t(\mathbf{c}_3) = \Pr\{\mathbf{c}^t = \mathbf{c}_3 | x^t = x_i, A^t = A_k\} = 1 - \mathfrak{d}(A_k),$$

where $\mathbf{c}_3 = \text{block}$; this probability does not depend on the target location x^t .

Note that if for some area $A \in \mathcal{X}$ it follows that $\mathfrak{d}(A) = 0$ for any time t , then it is assumed that this area is always blocked and can be removed from consideration. Similarly, if for an observed area $A \in \mathcal{X}$ it always holds true that $\psi(A) = 0$, then it does not contribute to the search process and also can be excluded from the set \mathcal{X} of available areas. Thus, for the considered areas it is specified that $\mathfrak{d}(A) > 0$ and $\psi(A) > 0$; that is, for each observed area $A \in \mathcal{X}$ there is at least one time such that the area can be selected, and an observation of each area is uncertain.

Let $A^t \circ \mathbf{c}^t = \langle A^{t-1}, \mathbf{c}^{t-1}, A^{t-2}, \mathbf{c}^{t-2}, \dots, A^2, \mathbf{c}^2, A^1, \mathbf{c}^1 \rangle$, $t = 1, 2, \dots$, be an observation history which is defined with respect to controls $\mathbf{c}^t \in \mathcal{C}$. Note that, in contrast to previous considerations, such a definition implies that the states of the system are specified by the controls rather than by target locations or exact observation results. Denote

$$\pi_i^t = \pi(x_i | \overrightarrow{A^t \circ \mathbf{c}^t}) = \Pr\{x^t = x_i | A^{t-1}, \mathbf{c}^{t-1}, A^{t-2}, \mathbf{c}^{t-2}, \dots, A^2, \mathbf{c}^2, A^1, \mathbf{c}^1\} \quad (3.36)$$

as the probability of the target location at the point $x_i \in X$, $i = 1, 2, \dots, n+1$, given a complete observation history $\overrightarrow{A^t \circ \mathbf{c}^t}$ up to time t . Then, a function $\pi^t : X \rightarrow [0, 1]$ such that $\sum_{i=1}^{n+1} \pi_i^t = 1$ for any t represents the target location probabilities with respect to the previous searcher's observations.

Finally, in contrast to the models considered above, define a search policy as the sequence $\overrightarrow{\mathfrak{d}}(A^t \circ \mathbf{c}^t) = \mathfrak{d}^t, \mathfrak{d}^{t-1}, \dots, \mathfrak{d}^1, \mathfrak{d}^0$ of *deterministic* rules of choice for the observed areas $A^t = \mathfrak{d}^t(\pi^t, A^{t-1})$ given the conditional target location probabilities π^t and previously chosen area A^{t-1} . As a result, in the Singh and Krishnamurthy model [26], a probabilistic

choice of the observed areas, as implemented by the previously considered models, is substituted by the deterministic choice among probability mass functions π^t available at time t , which define the target location probabilities with respect to the observations history.

The solution of the Singh and Krishnamurthy model [26] follows the general MDP framework (see Section 3.1.1). Let $R(x^t, A^t) \in [0, \infty)$ be an instantaneous reward which is obtained by the searcher at time t , $t = 0, 1, 2, \dots$, from the choice of the observed area $A^t \in \mathcal{X}$ while the target is located at the point $x^t \in X$. Since x_{n+1} is a termination point, it is assumed that $R(x_{n+1}, A^t) = 0$ for any observed area $A^t \in \mathcal{X}$. Similar to Equation 3.3, for any vector π^0 of initial target location probabilities $\pi_i^0 = p_i^0$, $i = 1, 2, \dots, n + 1$, and initial observed area A^0 , there exists an infinite horizon reward

$$E(R|\vec{\delta}, \pi^0, A^0) = \lim_{t \rightarrow \infty} E\left(\sum_{\tau=0}^t R(s^\tau, a^\tau) | \vec{\delta}, \pi^0, A^0\right),$$

which is obtained by use of the policy $\vec{\delta}$. Then, an optimal policy $\vec{\delta}^*$ is such a policy, which provides a maximal reward

$$E(R|\vec{\delta}^*) = \sup_{\vec{\delta}} \{E(R|\vec{\delta}, \pi^0, A^0)\}$$

for all possible initial location probabilities π^0 and initial observed areas A^0 .

As in the models of search above, an optimal policy $\vec{\delta}^*$ is obtained by using the dynamic programming approach with recursive equations similar to the Ross model equations (Equation 3.30) and guarantees the existence of an optimal solution. Since suitable specification of the observed areas, which are available to the searcher at each stage of search, reduces the Singh and Krishnamurthy model to certain algorithms of search and screening and path planning (see Section 2.1.3), such a model provides a general framework for a wide range of search problems. However, computation of both optimal strategy $\vec{\delta}^*$ and particular strategies, which are optimal with respect to the given initial location probability vector π^0 and initial observed area A^0 , requires exponential time and space. Note also that assuming that the detection is imperfect is crucial to the model. In Section 5.2, we will present an algorithm of search and screening that follows the direction of the Singh and Krishnamurthy model and implements heuristic informational policies.

3.3.2 Branch-and-bound procedure of constrained search with single searcher

The above models of search with constrained paths follow the general approach of MDP models and, as indicated in Section 2.1.3, also continue the line of search and screening methods, especially of Brown's Algorithm 2.2 and Washburn's Algorithm 2.3 (see Section 2.1.3). Below, we present a branch-and-bound search procedure [27, 28] for the constrained search, which demonstrates a link between the MDP model and search and screening methods and provides a basis for the game theory model of search that is considered below in Section 3.4.

The branch-and-bound search procedure follows the Stewart [28] method and implements a certain MDP and optimization techniques for the data, which are considered by search and screening methods. Let us recall the notation from search and screening (see

Section 2.1), in which, in contrast to the previous models, the searcher's behavior is defined by the use of the search effort dynamics.

Let, as above, $X = \{x_1, x_2, \dots, x_n\}$ be a sample space with initial target location probabilities $p_i = p(x_i) = p^{t=0}(x_i)$, $i = 1, 2, \dots, n$, and, following Section 2.1, let $\kappa^t(x_i) = \kappa(x_i, t) \geq 0$ be a search effort that is applied to the point x_i , $i = 1, 2, \dots, n$, at time t , $t = 0, 1, 2, \dots$. Similar to the Ross and Eagle models (see Sections 3.2.3 and 3.3.1, correspondingly) and in contrast to Stone's Algorithm 2.1 (Section 2.1.3), assume that the relocation of the search efforts over the points $x \in X$ is constrained so that for each point x_i , $i = 1, 2, \dots, n$, a neighborhood $N(x_i) \subset X$ is defined and consists of the points to which the search effort can be relocated at the end of the search at the current time.

Let $k^t(x_i, x_j)$ be the amount of search effort that is relocated from the point x_i to the point x_j , $i, j = 1, 2, \dots, n$, at the end of time t , $t = 0, 1, 2, \dots$, such that $k^t(x_i, x_j) \geq 0$ while $x_j \in N(x_i)$ and $k^t(x_i, x_j) = 0$ for $x_j \notin N(x_i)$. Then the search effort at the point x_j at time t is defined as the sum of the search efforts relocated from its neighboring points, that is [27, 28],

$$\kappa^t(x_j) = \sum_{\{x_l | x_l \in N(x_j)\}} k^{t-1}(x_l, x_j) = \sum_{x_l \in N(x_j)} k^t(x_j, x_l). \quad (3.37)$$

In addition, for a search by a single searcher it is assumed that $\kappa^{t=0}(x_j) = 1$ if the searcher starts at the point x_j , that is, the searcher's initial observed area is $A^{t=0} = \{x_j\}$, and that $\kappa^{t=0}(x_j) = 0$ otherwise. The scheme of search effort relocation is illustrated in Figure 3.6.

Note that Equation 3.37 implies that the 'flow' of the search efforts is constant in the sense that the amount of search effort coming into the point is equivalent to the amount of search effort going out from this point. The search effort with such a property differs from the additive search effort implemented in Section 2.1 for non-constrained search and screening algorithms.

Following the search and screening approach (see Section 2.1.1), given a search effort $\kappa^t(x_i)$ which is applied to the point x_i , $i = 1, 2, \dots, n$, up to time t , $t = 0, 1, 2, \dots$, the probability $\varphi(x_i, \kappa^t)$ of target detection at the point x_i is specified by a detection function $\varphi_\kappa : X \rightarrow [0, 1]$ such that $\varphi(x_i, 0) = 0$ and $0 < \lim_{\kappa^t(x_i) \rightarrow \infty} \varphi(x_i, \kappa^t) \leq 1$. According to the Koopman random-search equation (Equation 2.1), the probability $\varphi(x_i, \kappa^t)$ is defined as follows [27, 28]:

$$\varphi(x_i, \kappa^t) = 1 - e^{-\sigma(x_i) \kappa^t(x_i)}, \quad (3.38)$$

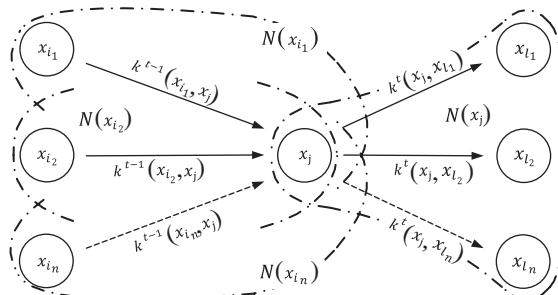


Figure 3.6 Relocation of the search efforts in the branch-and-bound procedure.

where the weighting coefficients $\sigma(x_i)$ are chosen in such a manner that $1 - e^{-\sigma(x_i)} < 1$ for any $i = 1, 2, \dots, n$. Since, according to Equation 3.37, the search effort flow preserves an amount of search efforts, the value $\varphi(x_i, \kappa^t)$, which specifies a detection probability *up to* time t , is equivalent to the detection probability $\psi(x_i, t)$ at time t given a search effort $\kappa^t(x_i)$. Thus, the probability $\varphi(x_i, \kappa^t) = \psi(x_i, t)$ in the branch-and-bound procedure plays the same role as the probability $\psi(x_i, t)$ in the Ross and Eagle models considered above.

Finally, similar to Brown's Algorithm 2.2, let $\overrightarrow{o}(t) = \langle x^{t=0}, \dots, x^{t-2}, x^{t-1}, x^t \rangle$ be a target trajectory up to time t , and let $p(\overrightarrow{o}(t)) \in [0, 1]$ be a probability that the target chooses this trajectory $\overrightarrow{o}(t)$. For convenience, we denote the points from the trajectory $\overrightarrow{o}(t)$ by $o(t)$, that is, $o(0) = x^{t=0}$, $o(1) = x^{t=1}$, and so on up to $o(t) = x^t$. As above, the set of all possible trajectories of the target up to time t is denoted by $\mathcal{O} = \{\overrightarrow{o} \in X^{(t)} : p(\overrightarrow{o}(t)) > 0\}$, where $X^{(t)}$ stands for the Cartesian product of t copies of X .

According to the search and screening approach, the problem of optimal allocation of search efforts over the points of the sample space X requires minimization of the probability $Q[\overrightarrow{o}(t)]$ of not detecting the target up to time t subject to the target's trajectory $\overrightarrow{o}(t)$ [28]. By using the search effort flows $k^t(x_i, x_j)$ such a problem is formulated as follows [27, 28]:

$$Q(\overrightarrow{o}(t)) = \sum_{\overrightarrow{o} \in \mathcal{O}} p(\overrightarrow{o}) \exp \left(- \sum_{\tau=1}^t \sigma(x_i) \sum_{\{x_i | o(\tau) \in N(x_i)\}} k^{\tau-1}(x_i, o(\tau)) \right) \rightarrow \min, \quad (3.39)$$

subject to

$$k_\tau = (x_i, x_j) \geq 0 \text{ for all } \tau = 1, 2, \dots, t-1 \text{ and } i, j = 1, 2, \dots, n, \quad (3.40)$$

$$\kappa^0(x_i) = \sum_{x_j \in N(x_i)} k^0(x_i, x_j) \quad \text{for all } i = 1, 2, \dots, n, \quad (3.41)$$

$$\begin{aligned} \sum_{\{x_i | x_j \in N(x_i)\}} k^{\tau-1}(x_i, x_j) &= \sum_{x_l \in N(x_j)} k^\tau(x_j, x_l) \quad \text{for each } j = 1, 2, \dots, n \text{ and all} \\ &\tau = 1, 2, \dots, t-1. \end{aligned} \quad (3.42)$$

Equations 3.41 and 3.42 represent the previously defined Equation 3.37 for preserving the search effort flow. For the Markovian target with a transition matrix $\rho = \|\rho_{ij}\|_{n \times n}$, Equation 3.39 obtains the simple form [27]

$$p \times \rho_{(0,j)} \times \rho_{(1,j)} \times \dots \times \rho_{(t,j)} \times \mathbf{1} \rightarrow \min,$$

where $p = (p_1, p_2, \dots, p_n)$ is a vector of initial target location probabilities, $\mathbf{1}$ is a column vector of ones and, similar to the Eagle model (see the proof of Lemma 3.6), $\rho_{(\tau,j)}$ stands for the transition probability matrix ρ , in which each j th row, $j = 1, 2, \dots, n$, is multiplied by $e^{-\sigma(x_j)\kappa^\tau(x_j)}$. The multiplier $e^{-\sigma(x_j)\kappa^\tau(x_j)}$ has the same meaning for the probability of not detecting the target as the multiplier $(1 - \psi_j)$ in the Eagle model.

Now let us introduce the searcher's trajectory into the optimization, Equations 3.39–3.42. A resulting optimization problem will provide a minimal probability of not detecting the target up to time t subject to the searcher's trajectory. Let us return to the detection function φ_κ with weighting coefficients $\sigma^\tau(x_i) = \sigma(x_i, \tau) > 0$ varying in time, $\tau = 0, 1, 2, \dots$, where $i = 1, 2, \dots, n$. In addition, assume that the weighted search effort

is additive, while the search effort flows are integer. Formally, these assumptions are represented as follows.

Let $\vec{a}(t) = \langle a(0), a(1), \dots, a(t-1), a(t) \rangle$ be a searcher's trajectory up to time t over the points $a(\tau) \in X$, $\tau = 0, 1, \dots, t$, of the sample space $X = \{x_1, x_2, \dots, x_n\}$. Such a trajectory implies that the searcher is allowed to observe the single-point observed areas only; that is, in the sequence $\vec{a}(t) = A^{t=0}, A^{t=1}, \dots, A^{t-1}, A^t$ (see Section 2.1.3) it follows that $A^\tau = \{a(\tau)\}$, $\tau = 0, 1, \dots, t$. In addition, let, as above, $\vec{o}(t) = \langle o(0), o(1), \dots, o(t-1), o(t) \rangle$ be the target's trajectory. Then, the probability $P(t)$ of detecting the target up to time t is defined by the Koopman's random-search formula, as follows:

$$P(t) = 1 - \exp \left(- \sum_{\tau=0}^t \sigma^\tau(a(\tau)) \kappa^\tau(a(\tau)) \right) = 1 - Q(t), \quad (3.43)$$

where a search effort $\kappa^\tau(a(\tau))$ is defined as an indicator function such that $\kappa^\tau(a(\tau)) = 1$ if $a(\tau) = o(\tau)$ and $\kappa^\tau(a(\tau)) = 0$ otherwise. Regarding the search effort flows $k^\tau(x_i, x_j)$, it is assumed that for any points x_i and x_j , $i, j = 1, 2, \dots, n$, it follows that $k^\tau(x_i, x_j) \in \{0, 1\}$, $\tau = 0, 1, \dots, t$, and Equation 3.37 holds.

As in the optimization problem, Equations 3.39–3.42, denote by $Q(\vec{a}(t))$ the probability of not detecting the target up to time t subject to the searcher's trajectory $\vec{a}(t)$. Then, the minimization problem for the probability $Q(\vec{a}(t))$ is formulated as follows [29]:

$$\begin{aligned} Q(\vec{a}(t)) &= \sum_{\vec{o} \in \mathcal{O}} p(\vec{o}) \\ &\times \exp \left(- \sum_{\tau=0}^{t-1} \sigma^\tau(a(\tau)) \sum_{\{a(\tau) | a(\tau+1) \in N(a(\tau))\}} k^\tau(a(\tau), a(\tau+1)) \right) \rightarrow \min, \end{aligned} \quad (3.44)$$

subject to:

$$\kappa^\tau(x_i) \in \{0, 1\}, k^\tau(x_i, x_j) \in \{0, 1\}, \tau = 0, 1, \dots, t, \text{ and } i, j = 1, 2, \dots, n, \quad (3.45)$$

$$\sum_{x \in N(a(\tau))} \kappa^{\tau+1}(x) = 1 \text{ and } \kappa^{\tau+1}(x) = 0 \text{ while } x \notin N(a(\tau)), \quad (3.46)$$

$$\sum_{x_j \in N(x_i)} k^\tau(x_i, x_j) = \kappa^\tau(x_j) \text{ for } i = 1, 2, \dots, n \text{ and } \tau = 0, 1, \dots, t, \quad (3.47)$$

$$\sum_{\{x_i | x_j \in N(x_i)\}} k^{\tau-1}(x_i, x_j) = \kappa^{\tau-1}(x_i) \text{ for } j = 1, 2, \dots, n \text{ and } \tau = 1, 2, \dots, t, \quad (3.48)$$

where Equation 3.46 requires the searcher to start a search at some initial point, and Equations 3.47 and 3.48 provide the searcher's movement from one point to another.

On the basis of the optimization problem, Equations 3.44–3.48, the branch-and-bound search procedure is outlined as follows.

Procedure 3.1 (branch-and-bound search) [29]. Given sample space X , search effort function κ^τ , and search effort flow k^τ such that Equation 3.37 holds, the Koopman detection probability function (Equation 3.38), starting point a_0 of the searcher, and final time $t \geq 0$, do:

1. Initialize searcher's trajectory $\vec{a}(t)$ as empty sequence.
2. Set $\tau = 0$ and specify some value $Q^* > 1$.
3. Specify searcher's starting point: Set $a(\tau) = a_0$.
4. Include $a(\tau)$ in the trajectory $\vec{a}(\tau)$: Set $\vec{a}(\tau) = \langle a(\tau) \rangle$.

5. Do:

- 5.1. Specify minimal probability $Q(\tau)$ of not detecting the target up to time τ by solving Equations 3.44–3.48 subject to the points in the searcher's trajectory $\vec{a}(\tau)$.
- 5.2. If $Q(\tau) < Q^*$ and $\tau < t$ then:
 - 5.2.1. Save the set $N'(\tau) = N(a(\tau))$ of neighbors, to which the search effort can be relocated from the point $a(\tau)$.
 - 5.2.2. Select a point $a(\tau + 1) \in N'(\tau)$ and add it to the trajectory $\vec{a}(\tau)$.
 - 5.2.3. Set $\tau = \tau + 1$.
- 5.3 If $Q(\tau) < Q^*$ and $\tau = t$ then:
 - 5.3.1 Set $Q(\tau) = Q^*$.
 - 5.3.2 Save the resulting trajectory $\vec{a}^*(\tau) = \vec{a}(\tau)$, where $\tau = t$.
- 5.4 If $\tau = 0$ then: Break.
- 5.5 Exclude the point $a(\tau)$ from $N'(\tau - 1)$ and from the trajectory $\vec{a}(\tau)$.
- 5.6 If $N'(\tau - 1) \neq \emptyset$ then:
 - 5.6.1 Select a new point $a(\tau) \in N'(\tau - 1)$ and add it to the trajectory $\vec{a}(\tau)$ instead of the excluded point. Else:
 - 5.6.2 Set $\tau = \tau - 1$.
 - 5.6.3 If $\tau = 0$ then: Break. Else:
 - a. Exclude the point $a(\tau)$ from the trajectory $\vec{a}(\tau)$.
 - b. Select a new point $a(\tau) \in N'(\tau - 1)$ and add it to the trajectory $\vec{a}(\tau)$ instead of the excluded point. While $Q(\tau) < Q^*$ and $\tau < t$.
6. Return the trajectory $\vec{a}^*(\tau)$ and the probability Q^* of not detecting the target. ■

The procedure starts with the initial point $a(\tau) = a_0$, which is included in the searcher's trajectory as a first point. Then, according to Lines 5.1 and 5.2, some available trajectory $\vec{a}(\tau)$ for relocation of the search efforts up to time $\tau = t$ is created and saved (Line 5.3). Starting from Line 5.5, the procedure follows a backward direction over the obtained trajectory. The last point is excluded from the neighborhood and from the trajectory (Line 5.5). If the neighborhood is still not empty, a new point is selected and added to the trajectory (Line 5.6.1), and the procedure returns to Lines 5.1 and 5.2 and checks the probability of not detecting the target by using a new trajectory. Otherwise, if the neighborhood is empty, the procedure moves one step backward along the trajectory, and excludes the previous point (Lines 5.6.2 and 5.6.3). Then the procedure returns to Lines 5.1 and 5.2 and rebuilds the excluded part of the trajectory.

This procedure presents a general model for the search and screening and path-planning methods (see Section 2.1). It results in an optimal path of the searcher such that the

probability of not detecting the target reaches its minimum. Since the procedure implements an optimization (Equations 3.44–3.48) for finding an optimal trajectory, the requirement regarding Koopman's definition of the detection probability as well as the requirement regarding the single-point observed areas are crucial.

However, note that in its actions, the branch-and-bound Procedure 3.1 follows the general direction of the (best-first) BF* algorithm (Procedure 2.5), which is applied for the search and path-planning over graphs (see Section 2.3). Such an observation provides a basis for the implementation of methods used in BF*-type algorithms for the search and screening procedures. Below we will use this observation within the framework of the Singh and Krishnamurthy model with heuristic informational policies.

Since the branch-and-bound Procedure 3.1 deals with the distribution of the search efforts and requires preservation of the search effort flows, it can be easily generalized for a search by multiple searchers. Below, we briefly consider such a generalization.

3.3.3 Constrained path search with multiple searchers

As indicated above, the branch-and-bound Procedure 3.1 can be implemented for solving path-planning problems while the search is conducted by a number of searchers. Formally, such a generalization implements a common search effort, which the searchers apply to a certain point of the sample space, while the optimization problem remains similar to Equations 3.44–3.48. Below, following Dell *et al.* [30], we consider a formalization of the problem and present two heuristic procedures of search and screening by multiple searchers.

Assume that there is a single target moving over the sample space $X = \{x_1, x_2, \dots, x_n\}$, and that the trajectory of the target up to time t , $t = 0, 1, 2, \dots$, is specified by the sequence $\overrightarrow{o}(t) = \langle o(0), o(1), \dots, o(t-1), o(t) \rangle \in \mathcal{O}$ of points $o(\tau) \in X$, $\tau = 0, 1, 2, \dots, t$, where \mathcal{O} is a set of possible target trajectories over the sample space X up to time t . Assume also that there are $m \geq 1$ searchers looking for the target, and denote by $\overline{a}_j(t) = \langle a_j(0), a_j(1), \dots, a_j(t-1), a_j(t) \rangle$ a trajectory of the j th searcher, $j = 1, \dots, m$. As above, at each moment of time each searcher is allowed to check a single point of the sample space, so $a_j(\tau) \in X$ for any $\tau = 0, 1, 2, \dots, t$.

Let $\kappa_j^\tau(x_i)$ be a search effort, which is applied to the point x_i by the j th searcher at time τ , and assume that the detection probability function φ is specified by the Koopman random-search equation (Equation 2.1). In addition, regarding the detection function φ , assume that the search efforts over the searchers are additive. That is, if there are $m' \leq m$ searchers at time τ , check the same point x_i ; then the detection probability $\psi(x_i, \tau)$ is defined by the sum of search efforts as follows [30]:

$$\psi(x_i, \tau) = 1 - \exp \left(- \sum_{j=1}^{m'} \kappa_j^\tau(x_i) \right), \quad i = 1, 2, \dots, n. \quad (3.49)$$

Note that for a single searcher, Equation 3.49 is equivalent to Equation 3.38 with the weights $\sigma(x_i) = 1$.

Finally, the search effort flows are defined for each searcher j , $j = 1, \dots, m$, by an amount $k_j^\tau(x_i, x_{i'})$ of search effort that is relocated by the searcher j from the point x_i to the point $x_{i'}$, $i, i' = 1, 2, \dots, n$, at time τ , $\tau = 0, 1, 2, \dots, t$. Similar to Equations 3.44–3.48, it is assumed that $k_j^\tau(x_i, x_{i'}) = 1$ if the j th searcher relocates the search effort from the point x_i to the point $x_{i'}$, and $k_j^\tau(x_i, x_{i'}) = 0$, otherwise.

As above, denote by $N(x) \subset X$ a neighborhood the point $x \in X$ that consists of the points to which the search effort can be relocated from point x at the end of the current time. Then, by using these assumptions, the problem of search is formulated as a minimization problem for the probability $Q(\bar{a}_{j=1,\dots,m}(t))$ of not detecting the target by use of the searchers' trajectories $\bar{a}_j(t)$, $j = 1, \dots, m$, as follows [30]:

$$\begin{aligned} Q(\bar{a}_{j=1,\dots,m}(t)) &= \sum_{\bar{o} \in O} p(\bar{o}) \\ &\times \exp \left(-\sum_{\tau=0}^t \sum_{j=0}^m \kappa_j^\tau(o(\tau)) \sum_{\{x_i | o(\tau) \in N(x_i)\}} k_j^\tau(x_i, o(\tau)) \right) \rightarrow \min, \end{aligned} \quad (3.50)$$

subject to:

$$\kappa_j^\tau(x_i) \in \{0, 1\}, \kappa_j^\tau(x_i x_{i'}) \in \{0, 1\}, \tau = 0, 1, \dots, t, i, i' = 1, 2, \dots, n, \text{ and } j = 1, \dots, m, \quad (3.51)$$

$$\sum_{x_{i'} \in N(x_i)} k_j^0(x_i, x_{i'}) = \kappa_j^0(x_{i'}), i, i' = 1, 2, \dots, n, \text{ and } j = 1, \dots, m, \quad (3.52)$$

where $\kappa_j^0(x_{i'}) = 1$ if the j th searcher starts with the point $x_{i'}$ and $\kappa_j^0(x_{i'}) = 0$ otherwise;

$$\sum_{\{x_{i'} | x_i \in N(x_i)\}} k_j^{\tau-1}(x_i, x_{i'}) = \sum_{x_i \in N(x_{i'})} k_j^\tau(x_i, x_{i'}), \tau = 1, 2, \dots, t, i, i' = 1, 2, \dots, n, \quad (3.53)$$

and $j = 1, \dots, m$.

Like Equations 3.44–3.48, Equation 3.52 requires the searchers to start at some initial points, and Equation 3.53 provides the searchers' movements.

From Equation 3.51 it follows that the value which appears as an argument of the exponential function in Equation 3.50 represents the number of times that at least one of the searchers and the target were located at the same point [29]. Thus, instead of the minimization (Equation 3.50), the problem can be formulated as a maximization problem of the expected number $E_{BB}(\bar{a}_{j=1,\dots,m}(t))$ of such a rendezvous [30]:

$$\begin{aligned} E_{BB}(\bar{a}_{j=1,\dots,m}(t)) &= \sum_{\bar{o} \in O} p(\bar{o}) \\ &\times \sum_{\tau=0}^t \sum_{j=0}^m \kappa_j^\tau(o(\tau)) \sum_{\{x_i | o(\tau) \in N(x_i)\}} \kappa_j^\tau(x_i, o(\tau)) \rightarrow \max, \end{aligned} \quad (3.54)$$

subject similarly to Equations 3.51–3.53.

Note that since the probability of detecting the target up to time t is $P(\bar{a}_{j=1,\dots,m}(t)) = 1 - Q(\bar{a}_{j=1,\dots,m}(t))$, Equations 3.50–3.53 are equivalent to the maximization problem regarding the probability $P(\bar{a}_{j=1,\dots,m}(t))$. Below, in the formulations of the search procedures, we will use this equivalence, as well as a maximal expected number $E_{BB}(\bar{a}_{j=1,\dots,m}(t))$ for the rendezvous.

On the basis of the formulated optimization problems regarding $Q(\bar{a}_{j=1,\dots,m}(t))$ or $E_{BB}(\bar{a}_{j=1,\dots,m}(t))$, optimal trajectories of the searchers can be obtained by using direct generalization of the branch-and-bound Procedure 3.1 to the search by multiple searchers. Now, let us briefly consider a simple heuristic procedure which applies the maximal probability $P(\bar{a}_{j=1,\dots,m}(t))$ of the maximal expected number of rendezvous $E_{BB}(\bar{a}_{j=1,\dots,m}(t))$ in a different manner to Procedure 3.1.

Procedure 3.2 (myopic heuristic search) [30]. Given sample space X , search effort function κ_j^τ , and search effort flow κ_j^τ , $j = 1, \dots, m$, as defined for Equations 3.50–3.53, the

Koopman detection probability function (Equation 3.49), starting points a_{0j} of the searchers, and final time $t \geq 0$, do:

1. Initialize searchers' trajectories $\overrightarrow{a}_j(t)$, $j = 1, \dots, m$, as empty sequences.
2. Set $\tau = 0$.
3. Specify searchers' starting points: Set $a_j(\tau) = a_{0j}$, $j = 1, \dots, m$.
4. Include $a_j(\tau)$ in the trajectory $\overrightarrow{a}_j(t)$, $j = 1, \dots, m$.
5. For $\tau = 1, \dots, t$, do:
 - 5.1. Specify current searchers' locations $a_j(\tau - 1)$ as initial points for estimated trajectories $\overrightarrow{b}_j(\tau - 1, \dots, t)$, which follow up to the final time t : Set $b_j(\tau - 1) = a_j(\tau - 1)$, $j = 1, \dots, m$.
 - 5.2. Find the continuations of the estimated trajectories $\overrightarrow{b}_j(\tau, \dots, t) = b_j(\tau), \dots, b_j(t)$ such that an expected number $E_{BB}(\overrightarrow{b}_{j=1,\dots,m})$ of rendezvous (Equation 3.54) reaches its maximum subject to Equations 3.51–3.53.
 - 5.3. Specify next searchers' locations according to the trajectories found: Set $a_j(\tau) = b_j(\tau)$, $j = 1, \dots, m$.
 - 5.4. Update the target location probabilities assuming that they have not been detected.
6. Calculate $P(\overrightarrow{a}_{j=1,\dots,m}(t)) = 1 - Q(\overrightarrow{a}_{j=1,\dots,m}(t))$ according to Equation 3.50, assuming that the searchers follow their obtained paths $\overrightarrow{a}_j(t)$, $j = 1, \dots, m$.
7. Return $P(\overrightarrow{a}_{j=1,\dots,m}(t))$ and searchers' paths $\overrightarrow{a}_j(t)$, $j = 1, \dots, m$. ■

The procedure follows a simple ‘looking forward’ heuristic, which represents one direction of the Washburn’s Forward-and-Backward (FAB) algorithm, Algorithm 2.3. Following such a heuristic, the searchers’ next locations are specified on the basis of estimated trajectories, which start from the current searchers’ locations. Hence, in contrast to the FAB algorithm, which uses backward recursion, Procedure 3.2 can be applied for online search given the current searchers’ locations.

A slight modification of Procedure 3.2 can be obtained by use of the neighborhoods $N(x) \subset X$ for the points $x \in X$ [30]. Assume that the current searchers’ locations, as specified in Line 5.1, are $a_j(\tau - 1)$, $j = 1, \dots, m$. Then, for each point $a_j(\tau - 1)$, a neighborhood $N(a_j(\tau - 1))$ is defined, which can be used in Line 5.2 so that a single action in this line is replaced by several actions [30]:

- 5.2.1. For each combination of the points $x_j \in N(a_j(\tau - 1))$ with respect to the searchers $j = 1, \dots, m$ do:
 - 5.2.1.1. Specify points x as initial points for estimated trajectories $\overrightarrow{b}'_j(\tau, \dots, t)$, which follow up to the final time t : Set $b'_j(\tau) = x_j$, $j = 1, \dots, m$.

- 5.2.1.2. Find the continuations of the estimated trajectories $\vec{b}'_j(\tau+1, \dots, t) = b'_j(\tau+1), \dots, b'_j(t)$ such that an expected number $E_{BB}(\vec{b}'_{j=1, \dots, m})$ of rendezvous (Equation 3.54) reaches its maximum subject to Equations 3.51–3.53.
- 5.2.1.3. Calculate $P(\vec{b}'_{j=1, \dots, m}) = 1 - Q(\vec{b}'_{j=1, \dots, m})$ according to Equation 3.50.
- 5.2.2. Specify the searchers' trajectories $\vec{b}_j(\tau, \dots, t) = b_j(\tau), \dots, b_j(t)$ such that $\vec{b}_j(\tau+1, \dots, t) = \text{argmax } P(\vec{b}'_{j=1, \dots, m})$.

The next actions follow the lines of Procedure 3.2.

These procedures finalize our consideration of the models of search, which are based on the dynamic programming approach. They provide existing optimal solutions of the search problem; however, direct application of the optimization methods requires exponential time and space, and for practical applications certain heuristics are required. As indicated above, in Chapter 5 we will present practical algorithms which follow the ideas of these procedures and illustrate their actions.

3.4 Game theory models of search

In the previous models, we considered the search problem in which the target's behavior does not depend on the searcher's actions up to terminating the search. In this section, we present another approach and consider the game theory models of search so that the target's actions depend on the searcher's strategy. Such models essentially deal with the problems of search for a moving target and relate to the *chase and escape* problems [31] (see Section 1.1). Within the framework of search such models are called *games of search* [32] or *pursuit-evasion games* [33–35]. Below, we consider a special class of search games, namely, PEGGs that provide a basis for the consideration of different kinds of search games in discrete space and time.

The game theory models of search have appeared in several different approaches to the search problem. Below, we consider three models of search games which correspond to the methods considered above. The first model follows a search and screening approach [36] and continues the line of the path-planning tasks (see Section 2.1.3) and of the Eagle and Washburn models of search (Section 3.3); the second model [37, 38] implements an MDP approach (see Sections 3.1 and 3.2) and considers the existence of search game solutions; and the third model addresses the search game over graphs [39], which in its probabilistic formulation [40, 41] follows the line of A*-type algorithms (see Section 2.3). Note that the indicated models do not cover all possible game theory formulations of the search problem; for additional information regarding search games and applications of game theory methods to the search problems see, for example, the books by Alpern and Gal [33], Gal [34], and Garnaev [35].

3.4.1 Game theory model of search and screening

Let us start with formulation of the game theory model of the search and screening problem, called the *cumulative search-evasion game* (CSEG) [36]. In this model, it is assumed that

the search is conducted by a single searcher, which can be considered as either a single agent or a society of cooperating agents with common search density function. However, at each time each agent is allowed to observe a single point of the sample space. Below, we will follow the notation of Section 3.3.2 and consider a search game with a single searcher; formulation of the game for multiple searchers can be obtained in a similar manner to that in Section 3.3.3 for the constrained search.

Let, as above, $X = \{x_1, x_2, \dots, x_n\}$ be a sample space, and denote by $\overrightarrow{o}(t) = \langle o(0), o(1), \dots, o(t-1), o(t) \rangle$ a trajectory of the target and by $\overrightarrow{a}(t) = \langle a(0), a(1), \dots, a(t-1), a(t) \rangle$ a trajectory of the searcher up to time t , $t = 0, 1, 2, \dots$. As above, the times up to moment t are denoted by $\tau = 0, 1, \dots, t$, so at each moment τ it follows that $o(\tau) \in X$ and $a(\tau) \in X$. Note, again, that in terms of Section 2.1.3 such an assumption implies that the searcher is allowed to observe the single-point observed areas only; that is, an observed area A^τ chosen by the searcher at time τ is $A^\tau = \{a(\tau)\}$, $\tau = 0, 1, \dots, t$.

Following the notation of group-testing search (see Section 2.2.1), let $z^\tau = z(A^\tau, x^\tau)$ be the result of observation of area A^τ obtained by using observation function z , where A^τ is a searcher's observed area and x^τ is a target location at time τ , $\tau = 0, 1, \dots, t$. Since all available search areas $A^\tau = \{a(\tau)\}$ are single point, below we will write $z^\tau = z(a(\tau), o(\tau))$ or $z^\tau = z(a_i, o_j, \tau)$, where a_i and o_j , $a_i, o_j \in X$, are points which are occupied at time τ by the searcher and the target, respectively, and we will use the last notation interchangeably.

Similar to Equation 3.54, let us consider a number of rendezvous between the searcher and the target, that is, the number of times at which $a(\tau) = o(\tau)$. Assume that observation function z is defined as an indicator function, so $z(a(\tau), o(\tau)) = 1$ while $a(\tau) = o(\tau)$ and $z(a(\tau), o(\tau)) = 0$ otherwise. Such a definition is similar to the definitions of the search effort function κ , which was implemented in Section 3.3.2; however, since functions z and κ have a different nature, we apply a different notation. By using the observation function, given the target's trajectory $\overrightarrow{o}(t)$ and the searcher's trajectory $\overrightarrow{a}(t)$, the number of rendezvous is specified by the sum

$$\text{rendezvous number} = \sum_{\tau=0}^t z(a(\tau), o(\tau)), \quad a(\tau) \in \overrightarrow{a}(t), \quad o(\tau) \in \overrightarrow{o}(t).$$

Then, in the game theory situation, the goal of the searcher is to find such a trajectory $\overrightarrow{a}^*(t)$ that maximizes the rendezvous number, while the goal of the target is to follow the trajectory $\overrightarrow{o}^*(t)$ that minimizes this number of rendezvous. The pair of trajectories $(\overrightarrow{a}^*(t), \overrightarrow{o}^*(t))$ which satisfy these goals is called the *solution of the game in pure strategies*. According to the terminology of Section 3.1.1, the strategy stands for a sequence of locations $a^*(\tau) \in \overrightarrow{a}^*(t)$ and $o^*(\tau) \in \overrightarrow{o}^*(t)$ that are chosen following purely deterministic rules.

In the case of the probabilistic choice of locations, the game addresses an *expected number of rendezvous*; the goal of the searcher is to maximize this number and the goal of the target is to minimize it. Denote by $u^\tau(a_i)$ the probability that at time τ , $\tau = 0, 1, \dots, t$, the searcher is located at the point $a_i \in X$. This probability is similar to the probability $\delta^\tau(a)$ of choice of action as used in the MDP models of search (see Section 3.1.1). In addition, denote by $v^\tau(o_i)$ the probability that at time τ , $\tau = 0, 1, \dots, t$, the target is located at the point $o_i \in X$. If the target moves independently of the searcher's actions, the probability $v^\tau(o_i)$ is equivalent to the expected location probability $\tilde{p}^\tau(o_i)$, but if the target follows its own choices, these probabilities are different. The notation introduced for probability mass functions u and v stresses the link between the considered problem and the search and screening procedures (see Section 2.1), which implement search density function u and target location density function v .

Then, given an observation function z such that $z(a_i, o_j, \tau) = z(a(\tau), o(\tau))$ is an observation result, obtained at time τ , $\tau = 0, 1, \dots, t$, while the searcher and target locations are $a_i, o_j \in X$, the expected number of rendezvous is defined as follows [36]:

$$E_{\text{CSEG}}(u, v|t) = \sum_{\tau=0}^t \sum_{i=1}^n \sum_{j=1}^n z(a_i, o_j, \tau) u^\tau(a_i) v^\tau(o_j),$$

where abbreviation CSEG stands for ‘Cumulative Search-Evasion Game’ [36] and stresses that the expected number of rendezvous accumulates over the time period. The pair of probability mass functions (u^*, v^*) such that

$$E_{\text{CSEG}}(u^*, v^*|t) = \max_u E_{\text{CSEG}}(u, v|t) = \min_v E_{\text{CSEG}}(u, v|t) \quad (3.55)$$

provides the *solution of the game in mixed strategies*. In other words, such a solution is formed by the searcher’s and the target’s *probabilistic policies*, which specify the choices of the searcher and target locations, respectively. The expected number of rendezvous $E_{\text{CSEG}}(u^*, v^*|t)$ for the pair (u^*, v^*) is called the *value of the search-evasion game*, and policies u^* and v^* are called *optimal policies* of the searcher and of the target, correspondingly.

The formulated game is finite, so the existence of its solution in mixed strategies is guaranteed by von Neumann’s min–max theorem, while for finding its solution general game theory theorems (see, e.g., [35]) can be implemented. Let us consider a linear programming method of finding such a solution, which is obtained by implementing a game theory approach to the search-evasion game.

The policies u and v are called *feasible* with respect to the searcher’s and the target’s movement abilities if for any time τ , $\tau = 0, 1, \dots, t$, the searcher and the target are able to move according to the probabilities $u^\tau(a)$ and $v^\tau(o)$, $a, o \in X$. In other words, feasible policies u and v specify zero probabilities for such points that cannot be reached from the current searcher and target locations. The feasibility of the probability mass functions u and v in the search-evasion game can be considered in the same sense as consistency of the costs in the A* algorithm (see Section 2.3.1) as used in Lemmas 2.10 and 2.11.

Let $0 < t' < t$, and denote by $u^{*|t'}_0$ and $v^{*|t'}_0$ optimal policies in the game up to time t' . Note that, in general, from the optimality of the policies over a certain time interval $[0, t']$, it does not follow that these policies are optimal over the complete time interval $[0, t]$. Let u^* and v^* be optimal policies in the game up to the final time t . Denote by $V_{\text{CSEG}}(u^*, v^*|t')$ a value of the game up to time t' and by $v_{\text{CSEG}}(u^*, v^*|\tau)$ a value of the game at time τ , while the searcher and the target use their optimal policies u^* and v^* , respectively, $\tau = 0, 1, \dots, t$. It is clear that, given policies u^* and v^* , it follows that $V_{\text{CSEG}}(u^*, v^*|t') = \sum_{\tau=0}^{t'} v_{\text{CSEG}}(u^*, v^*|\tau)$ for any $t' \leq t$.

Theorem 3.1 [36] *Assume that, regarding the policies u and v , it follows that if $0 \leq \tau \leq t'$, then $u^\tau(a) = u^{*\tau}(a)|_{0}^{t'}$ and $v^\tau = v^{*\tau}(o)|_{0}^{t'}$, if $t' \leq \tau \leq t$, then $u^\tau(a) = u^{*\tau}(a)$, and $v^\tau = v^{*\tau}(o)$, $a, o \in X$, where $u^{*|t'}_0$ and $v^{*|t'}_0$ are optimal policies over the time interval $[0, t']$ and u^* and v^* are optimal policies over the complete time interval $[0, t]$. Then, if policies u and v are feasible for the search-evasion game up to time t , they are also optimal.*

Proof. Assume that the searcher uses policy u and the target acts according to some feasible policy \tilde{v} . Since $u = u^{*|t'}_0$ is optimal for the game over interval $[0, t']$, it follows that

$E_{\text{CSEG}}(u, \tilde{v}|0 \dots t') \geq V_{\text{CSEG}}(u^*, v^*|t')$. Thus

$$E_{\text{CSEG}}(u, \tilde{v}|0 \dots t) \geq V_{\text{CSEG}}(u^*, v^*|t') + \sum_{\tau=t'+1}^t v_{\text{CSEG}}(u^*, v^*|\tau).$$

Similarly, if the searcher uses some policy \tilde{u} and the target acts according to the policy $v = v^*|t'$, then $E_{\text{CSEG}}(\tilde{u}, v|0 \dots t') \leq V_{\text{CSEG}}(u^*, v^*|t')$, and

$$E_{\text{CSEG}}(\tilde{u}, v|0 \dots t) \leq V_{\text{CSEG}}(u^*, v^*|t') + \sum_{\tau=t'+1}^t v_{\text{CSEG}}(u^*, v^*|\tau).$$

Since

$$\begin{aligned} & V_{\text{CSEG}}(u^*, v^*|t') + \sum_{\tau=t'+1}^t v_{\text{CSEG}}(u^*, v^*|\tau) \\ &= \sum_{\tau=0}^{t'} v_{\text{CSEG}}(u^*, v^*|\tau) + \sum_{\tau=t'+1}^t v_{\text{CSEG}}(u^*, v^*|\tau) = V_{\text{CSEG}}(u^*, v^*|t), \end{aligned}$$

one obtains

$$\begin{aligned} E_{\text{CSEG}}(u, \tilde{v}|0 \dots t) &\geq V_{\text{CSEG}}(u^*, v^*|t), \\ E_{\text{CSEG}}(\tilde{u}, v|0 \dots t) &\leq V_{\text{CSEG}}(u^*, v^*|t). \end{aligned}$$

Thus, the searcher's policy u and the target's policy v are optimal for a complete time interval $[0, t]$, that is, $u^\tau(a) = u^{*\tau}(a)$ and $v^\tau = v^{*\tau}(o)$ for all $a, o \in X$ and $\tau = 0, 1, \dots, t$. ■

Let us present the linear programming techniques for obtaining the feasible searcher's and target's policies [36]. Following the general search and screening approach (see Section 2.1.3), let us define a cost function $c : X \times [0, t] \rightarrow [0, \infty)$, which is interpreted as a payoff to the target for its evasion, and say that $c(o, \tau)$ is a *minimal cost* that is accumulated over the times $\tau, \tau+1, \dots, t$, while the searcher acts according to a policy u and the target is located at the point $o = o(\tau)$ at time τ . Note that since we implement a cumulative cost here, we do not need the cumulative search effort as used in the basic search and screening problem, and we follow the line of cost specification as in the Ross model of search (see Section 3.2.3).

Similar to the branch-and-bound procedure (see Section 3.3.2), denote by $N(o(\tau)) \subset X$, $o(\tau) \in X$, $\tau = 0, 1, \dots, t$, a set neighboring points, to which the target can move from the point $o(\tau)$. Then, given a searcher's policy u , minimal cumulative cost is defined as follows [36]:

$$c(o_j, \tau) = \sum_{i=1}^n z(a_i, o_j, \tau) u^\tau(a_i) + \min_{o \in N(o(\tau))} \{c(o, \tau+1)\}, \quad (3.56a)$$

$$c(o, t+1) = 0 \text{ for any } o \in X, \quad (3.56b)$$

where $j = 1, 2, \dots, n$ and $\tau = 0, 1, \dots, t$.

Denote by $C(t)$ the resulting minimal expected cumulative cost at final time t . As indicated above, the goal of the searcher is to maximize an expected number of rendezvous $E_{\text{CSEG}}(u, v|t)$ and the goal of the target is to minimize this number. Hence, given the searcher's policy u , the obtained minimal expected cumulative cost $C(t)$, which the target

has to pay for its evasion, meets the target's goal. Then, the *goal of the searcher* is to find such a policy u that maximizes the minimal expected cost $C(t)$.

Similarly, let us define a reward function $r : X \times [0, t] \rightarrow [0, \infty)$, which is interpreted as a reward that the searcher obtains for finding the target, and say that $r(a, \tau)$ is a *maximal reward* accumulated over times $\tau, \tau + 1, \dots, t$, while the target acts according to a policy v and the searcher at time τ checks the point $a = a(\tau)$. As above, denote by $N(a(\tau)) \subset X$, $a(\tau) \in X$, $\tau = 0, 1, \dots, t$, a set neighboring points, to which the search effort can be relocated after checking the point $a(\tau)$ at time τ . Then, given the target's policy v , maximal cumulative reward is defined as follows:

$$r(a_i, \tau) = \sum_{j=1}^n z(a_i, o_j, \tau) v^\tau(a_j) + \max_{a \in N(a(\tau))} \{r(a, \tau + 1)\}, \quad (3.57a)$$

$$r(a, t + 1) = 0 \text{ for any } a \in X, \quad (3.57b)$$

where $i = 1, 2, \dots, n$ and $\tau = 0, 1, \dots, t$.

By the same reasoning, denote by $R(t)$ the resulting maximal expected cumulative reward that is obtained by the searcher given the target's policy v . Then, the *goal of the target* is to find such a policy v that minimizes the maximal expected reward $R(t)$.

Finally, let us formulate the probabilistic version of the search effort relocations. In the case of the branch-and-bound procedure (see Section 3.3.2) such relocations are specified by the search effort flow and are restricted by Equation 3.37. Let $u^t(a_i, a_j)$ be the probability that the searcher is located at the point a_i at time t and at the point a_j at the next time $t + 1$. Then, the probabilistic version of Equation 3.37 is written as follows [36]:

$$\begin{aligned} u^t(a_i) &= \sum_{\{a_i | a_j \in N(a_i)\}} u^{t-1}(a_i, a_j) = \sum_{a_l \in N(a_j)} u^t(a_j, a_l), \\ i &= 1, 2, \dots, n, \quad t = 0, 1, 2, \dots \end{aligned}$$

Similarly, let $v^t(o_i, o_j)$ be the probability that the target is located at the point o_i at time t and at the point o_j at the next time $t + 1$. Then, regarding the probability $v^t(o_i)$, it is assumed that

$$\begin{aligned} v^t(o_i) &= \sum_{\{o_i | o_j \in N(o_i)\}} v^{t-1}(o_i, o_j) = \sum_{o_l \in N(o_j)} v^t(o_j, o_l), \\ i, j &= 1, 2, \dots, n, \quad t = 0, 1, 2, \dots \end{aligned}$$

On the basis of the introduced values and restrictions, linear programming problems for the searcher and the target are formulated as follows [36].

Searcher's side: Find a policy u such that it maximizes an expected cost of the target's evasion

$$C(t) \rightarrow \max, \quad (3.58a)$$

subject to:

$$\sum_{a_j \in N(a^0)} u^0(a^0, a_j) = 1, \quad (3.58b)$$

where a^0 is a starting point of the searcher,

$$\sum_{a_j \in N(a_i)} u^1(a_i, a_j) = u^0(a^0, a_i), \quad a_i \in N(a^0), \quad (3.58c)$$

$$\sum_{\{a_i | a_j \in N(a_i)\}} u^{\tau-1}(a_i, a_j) = \sum_{a_l \in N(a_j)} u^\tau(a_j, a_l), \tau = 2, \dots, t-1, \quad (3.58d)$$

$$u^t(a_i) = \sum_{\{a_i | a_j \in N(a_i)\}} u^{t-1}(a_i, a_j), i = 1, 2, \dots, n, \quad (3.58e)$$

$$c(o_j, t) = \sum_{i=1}^n z(a_i, o_j, t) u^t(a_i), j = 1, 2, \dots, n, \quad (3.58f)$$

$$c(o_j, \tau) \leq \sum_{i=1}^n z(a_i, o_j, \tau) \sum_{\{a_l \in N(a_i)\}} u^\tau(a_i, a_l) + c(o_k, \tau+1), \\ o_k \in N(o_j), j = 1, 2, \dots, n, \tau = 1, 2, \dots, t-1, \quad (3.58g)$$

$$C(t) \leq c(o_j, t), o_j \in N(o^0), \quad (3.58h)$$

where o^0 is a starting point of the target,

$$u^\tau(a_i, a_j) \geq 0, i, j = 1, 2, \dots, n, \tau = 0, 1, \dots, t. \quad (3.58i)$$

The meanings of the constraints (Equations 3.58b–3.58i) are similar to the meanings of the constraints in the branch-and-bound procedure (see Section 3.3.2). That is, the constraint in Equation 3.58b is required to start the search at the initial point a^0 ; constraints in Equations 3.58c and 3.58d specify the preserving of search effort flow; constraints in Equations 3.58e and 3.58f provide a termination of the search at the final time t ; and constraints in Equations 3.58g and 3.58h represent the definition of the minimal cumulative cost specified by Equations 3.56.

Target's side: Find a policy v such that it minimizes an expected searcher's reward

$$R(t) \rightarrow \min, \quad (3.59a)$$

subject to:

$$\sum_{o_j \in N(o^0)} v^0(o^0, o_j) = 1, \quad (3.59b)$$

where o^0 is a starting point of the target,

$$\sum_{o_j \in N(o_i)} v^1(o_i, o_j) = v^0(o^0, o_i), o_i \in N(o^0), \quad (3.59c)$$

$$\sum_{\{o_i | o_j \in N(o_i)\}} v^{\tau-1}(o_i, o_j) = \sum_{o_l \in N(o_j)} v^\tau(o_j, o_l), \tau = 2, \dots, t-1, \quad (3.59d)$$

$$v^t(o_i) = \sum_{\{o_i | o_j \in N(o_i)\}} v^{t-1}(o_i, o_j), i = 1, 2, \dots, n, \quad (3.59e)$$

$$r(a_i, t) = \sum_{j=1}^n z(a_i, o_j, t) v^t(o_j), i = 1, 2, \dots, n, \quad (3.59f)$$

$$r(a_i, \tau) \leq \sum_{j=1}^n z(a_i, o_j, \tau) \sum_{\{o_l \in N(o_j)\}} v^\tau(o_j, o_l) + r(a_k, \tau+1), \\ a_k \in N(a_i), i = 1, 2, \dots, n, \tau = 1, 2, \dots, t-1, \quad (3.59g)$$

$$R(t) \leq r(a_i, t), a_i \in N(a^0), \quad (3.59h)$$

where a^0 is a starting point of the searcher,

$$\sigma^\tau(o_i, o_j) \geq 0, i, j = 1, 2, \dots, n, \tau = 0, 1, \dots, t. \quad (3.59i)$$

The optimization in Equation 3.59 that specifies the target's policy is parallel to the optimization in Equation 3.58 regarding the searcher's policy, and the meanings of the constraints in Equations 3.59b–3.59i are similar to those in Equations 3.58b–3.58i.

Joint solution of the dual linear programming (Equations 3.58a and 3.59a) provides a solution (u^*, v^*) of the formulated CSEG, and the obtained optimal searcher's policy u^* and target's policy v^* result in the game's value $E_{\text{CSEG}}(u^*, v^*|t)$. If either a searcher or a target follows a certain constant policy, then the search-evasion game is reduced to the search and screening problem and can be solved by using the branch-and-bound procedure (see Section 3.3.2) or by a suitable search and screening algorithm (see Section 2.1). The next example illustrates the dynamics of the searcher's and target's policies.

Example 3.6 This example is based on the iterative method of solving matrix games, which was suggested by Brown [42] and improved by Robinson [43] (for a practical explanation of the method and an example see, e.g., [44]). Following this method, each player chooses a pure strategy such that it is the best against the averaged mixed strategies or probabilistic policies of the opponent as accumulated up to the current turn.

Since the goal of the example is to clarify the dynamics of the searcher's and target's policies, rather than to demonstrate the method of calculating the value of the game, we implement additional restrictions regarding the searcher's and the target's behavior. In the example, we assume that the searcher chooses the next point such that it maximizes the averaged probability of finding the target, and the target chooses its next point such that it minimizes this averaged probability.

As in the examples considered in Section 2.1, let the sample space $X = \{x_1, x_2, \dots, x_n\}$ be a gridded domain of the size $n = 10 \times 10$, and assume that at each time both the searcher and the target are allowed to move one step in one of the four directions 'north,' 'south,' 'east,' or 'west,' or to stay in the current position. Recall that $u^t(a_i, a_j)$ is the probability that the searcher is located at the point a_i at time t and at the point a_j at the next time $t + 1$, and similarly $\sigma^t(o_i, o_j)$ is the probability that the target is located at the point o_i at time t and at the point o_j at the next time $t + 1$, $i, j = 1, 2, \dots, n, t = 0, 1, 2, \dots$. Then, for the gridded domain with Manhattan distance $\text{distance}(x, x')$ between the points, this assumption specifies that for any $t, t = 0, 1, 2, \dots$, it follows that

$$u^t(a_i, a_j) \geq 0 \quad \text{if} \quad \text{distance}(a_i, a_j) \leq 1, \sigma^t(o_i, o_j) \geq 0 \quad \text{if} \quad \text{distance}(o_i, o_j) \leq 1,$$

$$u^t(a_i, a_j) = 0 \quad \text{if} \quad \text{distance}(a_i, a_j) > 1, \sigma^t(o_i, o_j) = 0 \quad \text{if} \quad \text{distance}(o_i, o_j) > 1,$$

$$\sum_{i=1}^n \sum_{j=1}^n u^t(a_i, a_j) = 1, \quad \sum_{i=1}^n \sum_{j=1}^n \sigma^t(o_i, o_j) = 1.$$

In addition, assume that both the searcher and the target are informed about the initial locations $a^{t=0}$ and $o^{t=0}$ of each other.

Now, let us consider the first steps of the game.

Step $t=0$. Assume that at initial time $t = 0$ the target is located at the point $o^0 = (7, 3)$ and the searcher starts the search at the point $a^0 = (7, 7)$. Thus, $\sigma^0(o^0) = 1$ and $u^0(a^0) = 1$. The initial situation is shown in Figure 3.7a.

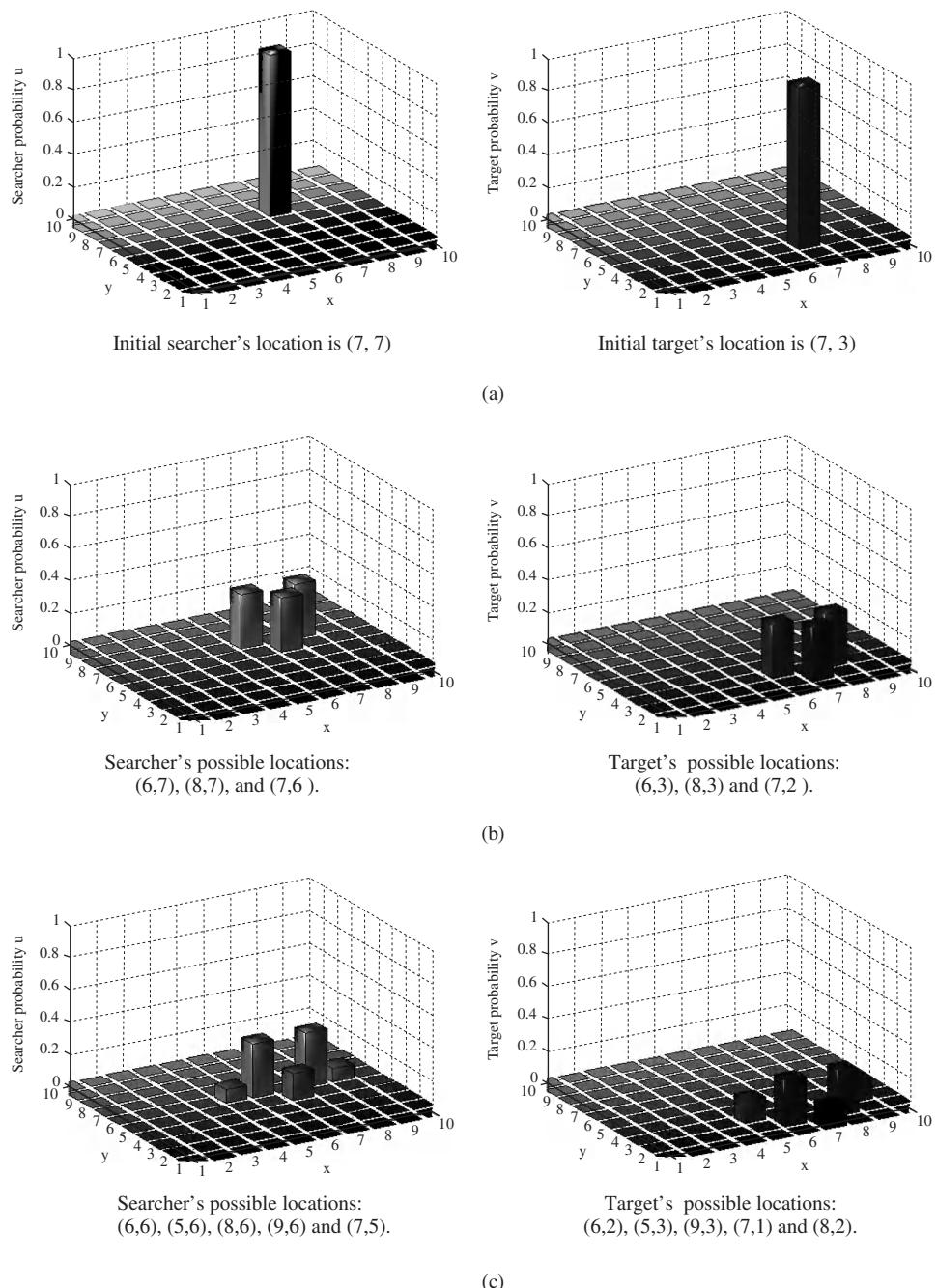


Figure 3.7 Dynamics of the searcher and target in the search-evasion game. (a) Initial searcher and target location probabilities, $t = 0$. (b) Searcher and target location probabilities, $t = 1$. (c) Searcher and target location probabilities, $t = 2$.

Step t = 1. The searcher is informed that the target is located at the point $o^0 = (7, 3)$ and that it can move to one of its neighboring points or stay in the current location. In addition, the searcher knows that the target is informed about the searcher's location and available movements. Thus, from the searcher's point of view, the target should move away from the searcher; that is, at time $t = 1$ the target can occupy one of the points $o_1^1 = (6, 3)$, $o_2^1 = (8, 3)$, $o_3^1 = (7, 2)$ with equal probabilities $v^{t=0}(o^0, o_1^1) = v^{t=0}(o^0, o_2^1) = v^{t=0}(o^0, o_3^1) = 1/3$. Hence, $v^{t=1}(o_1^1) = v^{t=1}(o_2^1) = v^{t=1}(o_3^1) = 1/3$. On the other hand, the target is informed about the searcher's initial location $a^0 = (7, 7)$ and on possible movements of the searcher, and also about the searcher's knowledge and goal. Thus, from the target's point of view, the searcher should move closer to the estimated target's location; that is, at time $t = 1$ the searcher can occupy one of the points $a_1^1 = (6, 7)$, $a_2^1 = (8, 7)$, $a_3^1 = (7, 6)$ with equal probabilities $u^{t=0}(a^0, a_1^1) = u^{t=0}(a^0, a_2^1) = u^{t=0}(a^0, a_3^1) = 1/3$. Thus, $u^{t=1}(a_1^1) = u^{t=1}(a_2^1) = u^{t=1}(a_3^1) = 1/3$. The searcher and target location probabilities are shown in Figure 3.7b.

Step t = 2. According to the searcher's knowledge, at the end of time $t = 1$ the target can be located at one of the points $o_1 = (6, 3)$, $o_2 = (8, 3)$, $o_3 = (7, 2)$ with probabilities $v^{t=1}(o_1^1) = v^{t=1}(o_2^1) = v^{t=1}(o_3^1) = 1/3$. Hence, at the current time $t = 2$, the target can occupy one of the neighboring points $o_1^2 = (6, 2)$, $o_2^2 = (5, 3)$, $o_3^2 = (9, 3)$, $o_4^2 = (8, 2)$, and $o_5^2 = (7, 1)$. The probabilities of the target's movements to these points are

$$\begin{aligned} v^{t=1}(o_1^1, o_1^2) &= v^{t=1}(o_1^1, o_2^2) = \frac{1}{2}, \\ v^{t=1}(o_2^1, o_3^2) &= v^{t=1}(o_2^1, o_4^2) = \frac{1}{2}, \\ v^{t=1}(o_3^1, o_5^2) &= v^{t=1}(o_3^1, o_1^2) = v^{t=1}(o_3^1, o_4^2) = \frac{1}{3}. \end{aligned}$$

Thus, the probabilities for the target's location at the neighboring points are as follows:

$$v^{t=2}(o_2^2) = v^{t=2}(o_3^2) = v^{t=2}(o_5^2) = \frac{1}{7} \quad \text{and} \quad v^{t=2}(o_1^2) = v^{t=2}(o_4^2) = \frac{2}{7}.$$

Similarly, according to the target's knowledge, the searcher can occupy one of the neighboring points $a_1^2 = (6, 6)$, $a_2^2 = (5, 6)$, $a_3^2 = (8, 6)$, $a_4^2 = (9, 6)$, and $a_5^2 = (7, 5)$. The probabilities of the searcher's movements to these points according to the target location probabilities are

$$\begin{aligned} u^{t=1}(a_1^1, a_1^2) &= \frac{1}{3}, & u^{t=1}(a_1^1, a_2^2) &= \frac{2}{3}, \\ u^{t=1}(a_2^1, a_4^2) &= \frac{1}{3}, & u^{t=1}(a_2^1, a_3^2) &= \frac{2}{3}, \\ u^{t=1}(a_3^1, a_5^2) &= u^{t=1}(a_3^1, a_1^2) = u^{t=1}(a_3^1, a_3^2) = \frac{1}{3}. \end{aligned}$$

Thus, the probabilities of the searcher's location at the neighboring points are as follows:

$$u^{t=2}(a_2^2) = u^{t=2}(a_4^2) = \frac{1}{12}, \quad u^{t=2}(a_1^2) = u^{t=2}(a_3^2) = \frac{1}{3}, \quad \text{and} \quad u^{t=2}(a_5^2) = \frac{1}{6}.$$

The searcher and target location probabilities at time $t = 2$ are shown in Figure 3.7c.

Note, again, that in these iterations the decisions of the searcher and the target were obtained on the basis of the policies which were built at the previous time. To obtain the value of the game, the policies have to be averaged and the decision regarding the next turn has to be obtained on the basis of these averaged policies. ■

This search-evasion game provides an example of the implementation of a general game theory approach to search and screening problems. As indicated above, if the target's actions do not depend on the searcher's behavior, the search-evasion game is reduced to a certain search and screening procedure.

3.4.2 Probabilistic pursuit-evasion games

In the previous section we considered a game theory extension of the search and screening approach. A similar consideration can be done regarding the MDP and POMDP models of search, which were considered in Sections 3.1 and 3.2. The resulting construction is known as *competitive Markov decision processes* or *Markov games* [45], while their implementation within the framework of search problems is known as *probabilistic pursuit-evasion games* (PPEGs) [37, 38]. In such games, in addition to the goal and actions of the searcher, the MDP or POMDP models take into account the goal and actions of the target. Below, we briefly consider one such model.

Let us return to the general MDP model, presented in Section 3.1. In this model, it is assumed that there exists a set of states and the dynamics of the system is provided by transitions between these states. In the MDP model of search the searcher (or a group of search agents) is considered as a single decision-maker and transitions between the states are defined by the searcher's actions specified by the searcher's decision policy. In contrast, in the game theory model, decisions are conducted both by the searcher and by the target, so transitions between the states depend on the searcher's and the target's policies. Formally, such a situation is specified as follows.

Let $\mathcal{S} = \{\mathcal{s}_1, \mathcal{s}_2, \dots\}$ be a finite or countable set of states, and, like the MDP model, let $\mathcal{A}_{[\text{ser}]} = \{a_{[\text{ser}]1}, a_{[\text{ser}]2}, \dots\}$ be a set of actions, which can be applied by the searcher. In addition, let $\mathcal{A}_{[\text{tar}]} = \{a_{[\text{tar}]1}, a_{[\text{tar}]2}, \dots\}$ be a set of actions, which are available to the target. In the pursuit-evasion game over a sample space $X = \{x_1, x_2, \dots, x_n\}$, the searcher's actions $a_{[\text{ser}]} \in \mathcal{A}_{[\text{ser}]}$ represent the choice of a certain observed area $A \subset X$ and obtaining observation result $z(A, x)$, while the target's actions $a_{[\text{tar}]} \in \mathcal{A}_{[\text{tar}]}$ correspond to the choice of a certain point $x \in X$ in the sample space and hiding at this point.

Following the assumption regarding Markovian dynamics of the system, the transition probabilities over the states are defined as follows [45]:

$$\rho_{ij}(a_{[\text{ser}]k}, a_{[\text{tar}]l}) = \Pr\{\mathcal{s}^{t+1} = \mathcal{s}_j | a_{[\text{ser}]}^t = a_{[\text{ser}]k}, a_{[\text{tar}]}^t = a_{[\text{tar}]l}, \mathcal{s}^t = \mathcal{s}_i\}, \quad (3.60)$$

where $\mathcal{s}^t, \mathcal{s}^{t+1} \in \mathcal{S}$ are the states of the system at times t and $t + 1$, $t = 0, 1, 2, \dots$; $a_{[\text{ser}]}^t \in \mathcal{A}_{[\text{ser}]}$ and $a_{[\text{tar}]}^t \in \mathcal{A}_{[\text{tar}]}$ are the actions of the searcher and the target that are chosen at this time; and $\mathcal{s}_i, \mathcal{s}_j \in \mathcal{S}$, $i, j = 1, 2, \dots$, $a_{[\text{ser}]k} \in \mathcal{A}_{[\text{ser}]}$, $k = 1, 2, \dots$, and $a_{[\text{tar}]l} \in \mathcal{A}_{[\text{tar}]}$, $l = 1, 2, \dots$. Such a definition is a straightforward generalization of the definition (Equation 3.1) of transition probabilities as implemented for the MDP model.

In the same manner, for the definition in Equation 3.2, let

$$\delta_{[\text{ser}]i}(a_{[\text{ser}]k}) = \Pr\{a_{[\text{ser}]} = a_{[\text{ser}]k} | \mathcal{s} = \mathcal{s}_i\} \quad (3.61)$$

be the probability that, given the system's state $s_i \in \mathcal{S}$, $i = 1, 2, \dots$, the searcher chooses an action $a_{[ser]k}$, $k = 1, 2, \dots$, and let

$$\mathfrak{d}_{[tar]i}(a_{[tar]k}) = \Pr\{a_{[tar]} = a_{[tar]l} | s = s_i\} \quad (3.62)$$

be the probability that, given the system's state $s_i \in \mathcal{S}$, $i = 1, 2, \dots$, the target chooses an action $a_{[tar]l}$, $l = 1, 2, \dots$. Then probability mass functions $\mathfrak{d}_{[ser]} : \mathcal{A}_{[ser]} \times \mathcal{S} \rightarrow [0, 1]$ and $\mathfrak{d}_{[tar]} : \mathcal{A}_{[tar]} \times \mathcal{S} \rightarrow [0, 1]$ specify *stationary probabilistic policies* of the searcher and the target, respectively. Note that in the general case, the policies $\mathfrak{d}_{[ser]}$ and $\mathfrak{d}_{[tar]}$ vary in time and depend on the history of the process, as implemented in the POMDP model in Section 3.2.1 using the information vector.

In parallel to the MDP model (see Section 3.1.1), let $R(s, a_{[ser]}, a_{[tar]}) \in [0, \infty)$ be an *immediate reward*, which is obtained by the searcher while the system is in the state $s \in \mathcal{S}$, the searcher chooses an action $a_{[ser]} \in \mathcal{A}_{[ser]}$, and the target chooses the action $a_{[tar]} \in \mathcal{A}_{[tar]}$. Such a reward has the same meaning as the reward in Equation 3.58a, which is defined in Section 3.4.1. In addition, following the line of the definition (Equations 3.57) of the cost, let $C(s, a_{[ser]}, a_{[tar]}) \in [0, \infty)$ be an *immediate cost*, which pays the target while the system is in the state s and the chosen actions are $a_{[ser]}$ and $a_{[tar]}$.

Finally, denote by $E_{PPEG}(R|\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]})$ an expected reward, which can be obtained by the searcher during the pursuit, and by $E_{PPEG}(C|\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]})$ an expected cost, which can be paid by the target while evading the searcher, where PPEG stands for ‘Probabilistic Pursuit-Evasion Game’ [37, 38]. Then, as in the optimization in Equation 3.58a, the goal of the searcher is to maximize an expected cost $E_{PPEG}(C|\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]})$, which has to be paid by the target; that is, to find a policy $\mathfrak{d}_{[ser]}^*$ that, for any other policy $\mathfrak{d}_{[ser]}$ and any target’s policy $\mathfrak{d}_{[tar]}$, it follows that

$$E_{PPEG}(C|\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]}) \leq E_{PPEG}(C|\mathfrak{d}_{[ser]}^*, \mathfrak{d}_{[tar]}).$$

Similarly, in parallel to Equation 3.58a, the goal of the target is to minimize an expected reward $E_{PPEG}(R|\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]})$, which can be obtained by the searcher; that is, to find a policy $\mathfrak{d}_{[tar]}^*$ that, for any other policy $\mathfrak{d}_{[tar]}$ and any searcher’s policy $\mathfrak{d}_{[ser]}$, it follows that

$$E_{PPEG}(R|\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]}^*) \leq E_{PPEG}(R|\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]}).$$

Assume that for any state $s \in \mathcal{S}$, searcher’s action $a_{[ser]} \in \mathcal{A}_{[ser]}$, and target’s action $a_{[tar]} \in \mathcal{A}_{[tar]}$ an immediate reward which is obtained by the searcher is equivalent to the cost paid by the target, that is, $R(s, a_{[ser]}, a_{[tar]}) = C(s, a_{[ser]}, a_{[tar]})$. Such games are called *zero-sum games*. In this case, for any pair of policies $\mathfrak{d}_{[ser]}$ and $\mathfrak{d}_{[tar]}$ it follows that $E_{PPEG}(C|\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]}) = E_{PPEG}(R|\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]})$. Denote this value by $E_{PPEG}(\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]})$. Substitution of this value into the presented inequalities results in

$$E_{PPEG}(\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]}^*) \leq E_{PPEG}(\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]}) \quad \text{and} \quad E_{PPEG}(\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]}) \leq E_{PPEG}(\mathfrak{d}_{[ser]}^*, \mathfrak{d}_{[tar]}).$$

Then, for the pair of policies $(\mathfrak{d}_{[ser]}^*, \mathfrak{d}_{[tar]}^*)$, for which both inequalities hold [45],

$$E_{PPEG}(\mathfrak{d}_{[ser]}, \mathfrak{d}_{[tar]}^*) \leq E_{PPEG}(\mathfrak{d}_{[ser]}^*, \mathfrak{d}_{[tar]}^*) \leq E_{PPEG}(\mathfrak{d}_{[ser]}^*, \mathfrak{d}_{[tar]}), \quad (3.63)$$

is called the *Nash equilibrium* of the Markov game. Similar to Equation 3.55, such equilibrium provides a max–min (min–max) solution so that, given the target’s policy $\mathfrak{d}_{[tar]}^*$,

the searcher's most profitable policy is $\delta_{[ser]}^*$, and, backwards, given the searcher's policy $\delta_{[tar]}^*$, the most profitable policy of the target is $\delta_{[tar]}^*$.

This model specifies a general Markov game between the searcher and the target. Let us implement this game in the particular form of a pursuit-evasion game [37, 38]. Below, we consider the simplest case of the pursuit-evasion game, which follows the MDP model of search with perfect observations.

Let $X = \{x_1, x_2, \dots, x_n\}$ be a finite sample space and $\mathcal{X} = 2^X$ be a power set of the set X , which consists of all possible subsets of X . The set of available observed areas is denoted by $\mathcal{X}_{[ser]} \subset \mathcal{X}$ and the set of possible target locations is denoted by $\mathcal{X}_{[tar]} \subset \mathcal{X}$. In the considered case, the set $\mathcal{X}_{[tar]}$ consists of the single-point sets $\mathcal{X}_{[tar]} = \{\{x_1\}, \{x_2\}, \dots, \{x_n\}\}$, and we will say that the target chooses points $x \in X$.

As indicated above, in the pursuit-evasion game the searcher's actions $a_{[ser]}$ are related with the choices of the observed areas $A \subset X$ and obtaining an observation result, while an observation of the chosen area A is a common part of the actions and does not depend on the choice of the area. Thus, similar to the MDP model (see Section 3.1.2) and the underlying group-testing search algorithms (see Section 2.2.1), it is assumed that the choice of the action $a_{[ser]} \in \mathcal{A}_{[ser]}$ is equivalent to the choice of the observed area $A \in \mathcal{X}_{[ser]}$. In the same manner, the choice of the target's action $a_{[tar]}$ is equivalent to the choice of the single-point set $\{x\} \in \mathcal{X}_{[tar]}$, or, the same thing, of the point $x \in X$.

To stress the symmetry between the searcher's and the target's decision, the states s in the game theory model of search are defined differently from the states in the MDP and POMDP models (see Sections 3.1 and 3.2) and include the searcher's and target's choices. That is [37, 38], it is assumed that the set of states $\mathcal{S} \subseteq \mathcal{X}_{[ser]} \times \mathcal{X}_{[tar]}$ and the states are specified by the pairs $s = (A, \{x\})$. Because of the symmetry between the searcher and the target in their knowledge regarding the obtained probabilistic information, it is assumed that the observation result $z(A, x)$ is available both for the searcher and for the target.

Note that since the sample space $X = \{x_1, x_2, \dots, x_n\}$ is finite, the set $\mathcal{X}_{[ser]}$ includes at most 2^n observed areas and the set $\mathcal{X}_{[tar]}$ includes at most n single-point sets. Thus, the set of states \mathcal{S} is also finite with at most $2^n \times n$ elements.

Following this definition of the states and actions, transition probabilities (Equation 3.60) are specified as follows:

$$\rho(A', x' | A, x) = \Pr\{A^{t+1} = A', \{x^{t+1}\} = \{x'\} | A^t = A, \{x^t\} = \{x\}\},$$

where $A', A \in \mathcal{X}_{[ser]}$ and $\{x'\}, \{x\} \in \mathcal{X}_{[tar]}$. Since the searcher and the target conduct independent decision making, it is assumed that [37]

$$\begin{aligned} \rho(A', x' | A, x) &= \Pr\{A^{t+1} = A' | A^t = A, \{x^t\} = \{x\}\} \times \Pr\{\{x^{t+1}\} = \{x'\} | A^t = A, \{x^t\} = \{x\}\} \\ &= \rho(A' | A, x) \times \rho(x' | A, x). \end{aligned}$$

As indicated above, the transition probabilities do not depend on the observation result. The probabilistic policies (Equations 3.61 and 3.62) of the searcher and the target are specified as follows:

$$\delta_{[ser]}(A' | A, x, z) = \Pr\{A^{t+1} = A' | A^t = A, \{x^t\} = \{x\}, z(A, x)\},$$

$$\delta_{[tar]}(x' | A, x, z) = \Pr\{x^{t+1} = x' | A^t = A, \{x^t\} = \{x\}, z(A, x)\},$$

and the probabilities $\mathfrak{d}_{[\text{ser}]}(A'|A, x, z)$ and $\mathfrak{d}_{[\text{tar}]}(x'|A, x, z)$, which are defined by these policies over the sets $\mathcal{X}_{[\text{ser}]}$ and $\mathcal{X}_{[\text{tar}]}$, correspondingly, depend on the observation results $z(A, x)$. Note that in the case of perfect observations it follows that $z(A, x) = 1$ when $x \in A$ and $z(A, x) = 0$ when $x \notin A$. Then,

$$\begin{aligned}\rho(A'|A, x) &= \mathfrak{d}_{[\text{ser}]}(A'|A, x, 0) + \mathfrak{d}_{[\text{ser}]}(A'|A, x, 1), \\ \rho(x'|A, x) &= \mathfrak{d}_{[\text{tar}]}(x'|A, x, 0) + \mathfrak{d}_{[\text{tar}]}(x'|A, x, 1).\end{aligned}$$

Finally, since states $s \in \mathcal{S}$ are specified by the pairs $s = (A, \{x\})$, for immediate reward and immediate cost one obtains $R(s, a_{[\text{ser}]}, a_{[\text{tar}]}) \equiv R(A', x'|A, x)$ and $C(s, a_{[\text{ser}]}, a_{[\text{tar}]}) \equiv C(A', x'|A, x)$. As indicated above, in the considered zero-sum game, the reward which is obtained by the searcher is equivalent to the cost paid by the target; hence it follows that $R(A', x'|A, x) = C(A', x'|A, x)$.

On the basis of these policies, the dynamics of the game can be specified in the form of two parallel MDPs or, essentially the same, as a controlled Markov process [46, 47] with feedback control. In general, the dynamics of the pursuit-evasion game is specified as follows:

1. The searcher starts with initial target location probabilities over the set $\mathcal{X}_{[\text{tar}]}$:

$$p_{[\text{tar}]}^{t=0}(\{x\}) = \Pr\{\{x^{t=0}\} = \{x\}\}, \quad \{x\} \in \mathcal{X}_{[\text{tar}]}, \quad \sum_{\{x\} \in \mathcal{X}_{[\text{tar}]}} p_{[\text{tar}]}^{t=0}(\{x\}) = 1.$$

As indicated above, such probabilities represent the knowledge of the searcher regarding the initial target location. In contrast to the search procedures, in the game theory approach the target acts similarly to the searcher; hence it is informed about the searcher's initial location and such knowledge is represented by the initial searcher location probabilities over the set $\mathcal{X}_{[\text{ser}]}$:

$$p_{[\text{ser}]}^{t=0}(A) = \Pr\{A^{t=0} = A\}, \quad A \in \mathcal{X}_{[\text{ser}]}, \quad \sum_{A \in \mathcal{X}_{[\text{ser}]}} p_{[\text{ser}]}^{t=0}(A) = 1.$$

Note that the probabilities $p_{[\text{ser}]}(A)$ differ from the probabilities $p(A)$, which are used in the MDP model of search in Section 3.1.2: probabilities $p_{[\text{ser}]}(A)$ specify the target's knowledge regarding the searcher's choice of the observed area $A \in \mathcal{X}_{[\text{ser}]}$, while probabilities $p(A)$ represent the searcher's knowledge regarding the target location in the area A .

2. As in the models above, at each time t , $t = 0, 1, 2, \dots$, the searcher is not informed about the target location $\{x^t\} \in \mathcal{X}_{[\text{tar}]}$ and the target is not informed about the searcher location (chosen observed area) $A^t \in \mathcal{X}_{[\text{ser}]}$. However, it is assumed that after obtaining observation result $z^t = z(A^t, x^t)$, it is available both for the searcher and for the target.
3. Given the searcher and the target location probabilities $p_{[\text{ser}]}(A^t)$, $A^t \in \mathcal{X}_{[\text{ser}]}$, and $p_{[\text{tar}]}(\{x^t\})$, $\{x^t\} \in \mathcal{X}_{[\text{tar}]}$, and observation result $z^t = z(A^t, x^t)$, the searcher and the target implement their policies $\mathfrak{d}_{[\text{ser}]}$ and $\mathfrak{d}_{[\text{tar}]}$.

Similar to the searcher's knowledge of the target's transition probability matrix ρ , as implemented in the MDP model of search (see Section 3.1.2), it is assumed that both the searcher and the target are informed about the policies $\mathfrak{d}_{[\text{ser}]}$ and $\mathfrak{d}_{[\text{tar}]}$. In other

words, given the searcher and the target location probabilities $p_{[\text{ser}]}(A^t)$, $A^t \in \mathcal{X}_{[\text{ser}]}$, and $p_{[\text{tar}]}(\{x^t\})$, $\{x^t\} \in \mathcal{X}_{[\text{tar}]}$, and observation result $z^t = z(A^t, x^t)$, the searcher is informed about the probability $\delta_{[\text{tar}]}(x^{t+1}|A^t, x^t, z^t)$ that the target will move to the location $\{x^{t+1}\}$, and the target is informed about the probability $\delta_{[\text{ser}]}(A^{t+1}|A^t, x^t, z^t)$ that the searcher chooses observed area A^{t+1} .

4. The game terminates if it follows that $A^t = \{x^t\}$ and observation result $z^t \geq z^*$, where z^* is a predefined threshold; in the case of perfect detection $z^* = 1$.

The outlined dynamics continues the direction of the MDP model of search (see Section 3.1.2) with additional decision making that is conducted by the target. The actions of the game are illustrated in Figure 3.8.

As indicated above, the searcher's and the target's policies $\delta_{[\text{ser}]}$ and $\delta_{[\text{tar}]}$ are defined by using the Nash equilibrium, which specifies the most profitable behavior of the search and target. Following the definitions presented above, the expected immediate reward and, equivalently, the expected immediate cost are defined by

$$\begin{aligned} E(\delta_{[\text{ser}]}, \delta_{[\text{tar}]}) &= E(R|\delta_{[\text{ser}]}, \delta_{[\text{tar}]}) = \rho(A', x'|A, x) \times R(A', x'|A, x) \\ &= E(C|\delta_{[\text{ser}]}, \delta_{[\text{tar}]}) = \rho(A', x'|A, x) \times C(A', x'|A, x) \\ &= (\delta_{[\text{ser}]}(A'|A, x, 0) + \delta_{[\text{ser}]}(A'|A, x, 1)) \times (\delta_{[\text{tar}]}(x'|A, x, 0) + \delta_{[\text{tar}]}(x'|A, x, 1)). \end{aligned}$$

The searcher's and target's policies $\delta_{[\text{ser}]}^*$ and $\delta_{[\text{tar}]}^*$, for which it holds true that

$$E(\delta_{[\text{ser}]}, \delta_{[\text{tar}]}^*) \leq E(\delta_{[\text{ser}]}^*, \delta_{[\text{tar}]}^*) \leq E(\delta_{[\text{ser}]}^*, \delta_{[\text{tar}]}),$$

provide the *one-step Nash equilibrium* of the pursuit-evasion game [37]. This equilibrium specifies the most profitable one-step probabilistic rules for the searcher's choice of the observed area $A \in \mathcal{X}_{[\text{ser}]}$ and for the target's choice of its location $\{x\} \in \mathcal{X}_{[\text{tar}]}$. According to the von Neumann min–max theorem (see, e.g., [35]), since the sets $\mathcal{X}_{[\text{ser}]}$ and $\mathcal{X}_{[\text{tar}]}$ are finite, such an equilibrium always exists.

Note that in the considered case it is assumed that the policies of the searcher and the target do not depend on time. Hence, policies $\delta_{[\text{ser}]}^*$ and $\delta_{[\text{tar}]}^*$, which provide the one-step Nash equilibrium for all possible searcher's and target's choices, also provide a Nash equilibrium (Equation 3.63) in the game. However, in the general case with

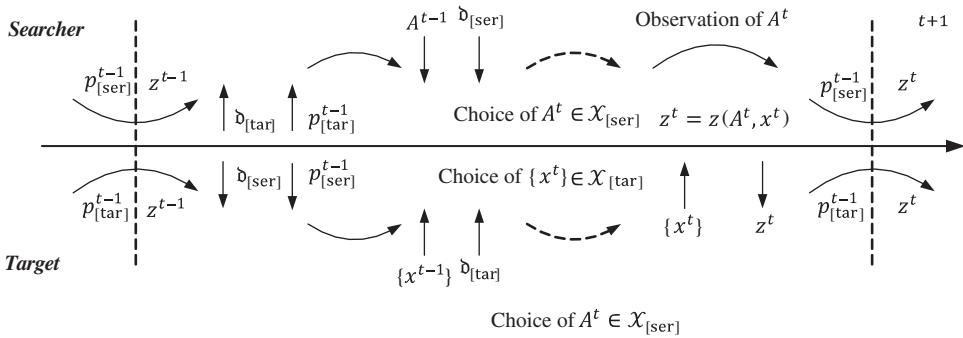


Figure 3.8 Dynamics of pursuit-evasion game.

imperfect observations, while the policies vary according to the history of the game, the relation between one-step equilibrium and equilibrium (Equation 3.63) does not hold, and an expected reward (or expected cost) $E_{\text{PEGG}}(\vartheta_{[\text{ser}]}, \vartheta_{[\text{tar}]})$ over the game has to be defined specifically. In the next section, we consider a certain A*-type algorithm which implements the probabilistic search over a graph and can be applied for an approximate solution of the pursuit-evasion game over graphs.

3.4.3 Pursuit-evasion games on graphs

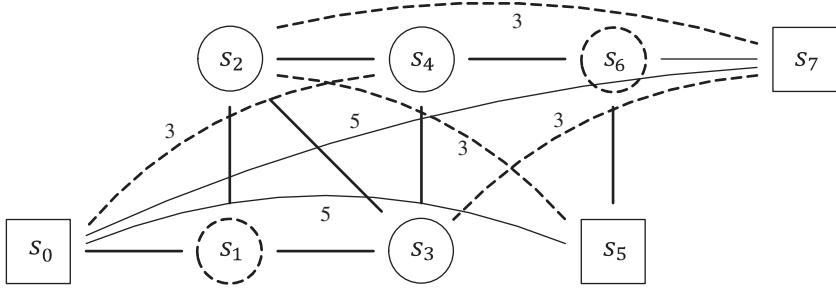
Finally, let us consider a special case of the pursuit-evasion games, namely PEGGs. We present general facts regarding these games [39] and also present an A*-type algorithm [40, 41] which implements a search over graphs with changing probabilities of target locations. In the consideration below, we follow the notation introduced in Section 2.3 while addressing algorithms for a search over graphs.

Let $\mathcal{G} = (S, E)$ be a finite connected graph with a set S of vertices (or nodes) and a set of edges $E \subset S \times S$. As in Section 2.3, we assume that in the graph there are no loops or multiple edges. The vertices $s \in S$ of the graph represent locations of the searcher and the target, and the edges $(s, s') \in E$ specify possible paths for one-step movements. If at each time the target can occupy a single point x of the sample space X and the searcher deals with single-point observed areas $A = \{x\} \in \mathcal{X}$, then the set of vertices $S = \{s_1, s_2, \dots, s_n\}$ is equivalent to the sample space $X = \{x_1, x_2, \dots, x_n\}$, and the set of edges E determines a topology of X .

For the next consideration, we need several values defined over the graph \mathcal{G} . Let $s, s' \in S$ be two vertices. The *distance* $d(s, s')$ between the vertices s and s' is defined by a number of edges in the shortest path, which starts at the vertex s and ends at the vertex s' . If (s, s') is an edge, then $d(s, s') = 1$, and if $s = s'$, then $d(s, s') = 0$. *Diameter* $\text{diam}(\mathcal{G})$ of the graph \mathcal{G} is the length of the longest path in the graph, that is, $\text{diam}(\mathcal{G}) = \max_{s, s' \in S} d(s, s')$. *Radius* $\text{rad}(\mathcal{G})$ of the graph \mathcal{G} is the minimum of the longest paths taken over the vertices, that is, $\text{rad}(\mathcal{G}) = \min_{s \in S} \max_{s' \in S} d(s, s')$. The values of the graph are illustrated in Figure 3.9. In the figure, the most remote vertices s_0, s_5 , and s_7 are denoted by squares, and the vertices s_2, s_3 , and s_4 with minimal eccentricity, called *central vertices*, are denoted by solid circles. Dashed curves with values specify the distances between the vertices, which result in the radius of the graph, and solid curves with values specify the distances between the vertices, which result in the diameter of the graph.

The pursuit-evasion game on the graph \mathcal{G} is conducted as follows. At each time the searcher chooses a vertex s and the target chooses a vertex y . Then, the searcher pays an amount $d(s, y)$ to the target, and the target obtains this amount, and the choice of the vertices continues. The *goal of the searcher* is to minimize the amount paid, while the *goal of the target* is to maximize it. Note that the specified goals are inverses of the goals of the searcher and the target, which were considered for the previous games. To insure that these goals of the game on graphs are equivalent to the goals of general search-evasion and pursuit-evasion games, recall that the diameter $\text{diam}(\mathcal{G})$ of the finite graph is also finite. Thus, the goal of the searcher is equivalent to maximizing the cost $C(s, y) = \text{diam}(\mathcal{G}) - d(s, y) \geq 0$, which is paid by the target, and the goal of the target is equivalent to minimizing the same reward $R(s, y) = \text{diam}(\mathcal{G}) - d(s, y) \geq 0$, which is obtained by the searcher.

Let $S = \{s_1, s_2, \dots, s_n\}$ and, as above, let $p_{[\text{ser}]}(s_i)$ be the probability that the searcher checks the vertex s_i , and let $p_{[\text{tar}]}(s_i)$ be the probability that the target is located at the vertex



$$\text{diam}(\mathcal{G}) = d(s_0, s_5) = d(s_0, s_7) = 5,$$

$$\text{rad}(\mathcal{G}) = d(s_0, s_4) = d(s_2, s_5) = d(s_2, s_7) = d(s_3, s_5) = d(s_3, s_7) = 3.$$

Figure 3.9 Diameter and radius of the graph.

s_i , $i = 1, 2, \dots, n$. The probability vectors for the searcher and the target are denoted by $\overrightarrow{p}_{[\text{ser}]} = (p_{[\text{ser}]}(s_1), \dots, p_{[\text{ser}]}(s_n))$ and $\overrightarrow{p}_{[\text{tar}]} = (p_{[\text{tar}]}(s_1), \dots, p_{[\text{tar}]}(s_n))$, correspondingly. Denote by $\Delta(\mathcal{G}) = d(s_i, s_j)_{n \times n}$, $s_i, s_j \in S$, a distance matrix of the graph \mathcal{G} . Then, as in games above, the value of the PEGG is defined as follows [39]:

$$\begin{aligned} V_{\text{PEGG}}(\overrightarrow{p}_{[\text{ser}]}, \overrightarrow{p}_{[\text{tar}]}) &= \max_{\overrightarrow{p}_{[\text{tar}]}} \min_{\overrightarrow{p}_{[\text{ser}]}} \left\{ \overrightarrow{p}_{[\text{tar}]}^T \times \Delta(\mathcal{G}) \times \overrightarrow{p}_{[\text{ser}]} \right\} \\ &= \min_{\overrightarrow{p}_{[\text{ser}]}} \max_{\overrightarrow{p}_{[\text{tar}]}} \left\{ \overrightarrow{p}_{[\text{tar}]}^T \times \Delta(\mathcal{G}) \times \overrightarrow{p}_{[\text{ser}]} \right\}, \end{aligned} \quad (3.64)$$

where $(\cdot)^T$ stands for the transposition of the vector. The pair $(\overrightarrow{p}_{[\text{ser}]}, \overrightarrow{p}_{[\text{tar}]})$ of the searcher's and the target's probability vectors provides a solution of the game in the mixed strategies, and the vectors $\overrightarrow{p}_{[\text{ser}]}$ and $\overrightarrow{p}_{[\text{tar}]}$ specify optimal probabilistic policies of the searcher and the target, respectively.

Theorem 3.2 [36] *For any pursuit-evasion game on the graph \mathcal{G} it follows that $V_{\text{PEGG}}(\overrightarrow{p}_{[\text{ser}]}, \overrightarrow{p}_{[\text{tar}]}) < \text{diam}(\mathcal{G})$.*

Proof. For optimal policies $\overrightarrow{p}_{[\text{ser}]}$ and $\overrightarrow{p}_{[\text{tar}]}$ it follows that

$$V_{\text{PEGG}}(\overrightarrow{p}_{[\text{ser}]}, \overrightarrow{p}_{[\text{tar}]}) = \max_i (\Delta(\mathcal{G}) \times \overrightarrow{p}_{[\text{ser}]}^*)_i = \min_j (\overrightarrow{p}_{[\text{tar}]}^{*T} \times \Delta(\mathcal{G}))_j.$$

Since $\overrightarrow{p}_{[\text{tar}]}^*$ is a probability vector, at least one element $p_{[\text{tar}]}^*(s_j)$, $j = 1, 2, \dots, n$, is greater than 0. Then,

$$\begin{aligned} (\overrightarrow{p}_{[\text{tar}]}^{*T} \times \Delta(\mathcal{G}))_j &= p_{[\text{tar}]}^*(s_j) \times d(s_j, s_j) + \sum_{\substack{i=1 \\ i \neq j}}^n p_{[\text{tar}]}^*(s_i) \times d(s_i, s_j) \\ &\leq (1 - p_{[\text{tar}]}^*(s_j)) \text{diam}(\mathcal{G}) \\ &< \text{diam}(\mathcal{G}) = \max_{i,j} \{d(s_i, s_j)\}. \end{aligned}$$

Therefore,

$$V_{\text{PEGG}}(\vec{p}_{[\text{ser}]}^*, \vec{p}_{[\text{tar}]}^*) = \min_j (\vec{p}_{[\text{tar}]}^{*T} \times \Delta(\mathcal{G}))_j \leq (\vec{p}_{[\text{tar}]}^{*T} \times \Delta(\mathcal{G}))_j < \text{diam}(\mathcal{G}),$$

as required. \blacksquare

Now, let us specify a particular type of the graphs. Graph $\mathcal{G} = (S, E)$ is called vertex-transitive if for any two vertices $s, s' \in S$ there exists an automorphism $f : S \rightarrow S$ such that $s' = f(s)$. In other words, the graph is vertex-transitive if the pairwise permutation of its vertices does not change the structure of the graph. In particular, complete graph \mathcal{K}_n and cycle graph \mathcal{C}_n , where n is the number of vertices, are vertex-transient; such graphs are shown in Figure 3.10a,b.

In contrast, a path graph \mathcal{P}_n (see Figure 3.10c) as well as the graph shown in Figure 3.9, and are not vertex-transitive.

Note that in the vertex-transient graphs all vertices have equal numbers of incident edges, vertex degrees $\deg(s)$; that is, the number of available movements does not depend on the vertex. For the pursuit-evasion game on such graphs, the following theorem holds.

Theorem 3.3 [39] *In the pursuit-evasion game on the vertex-transitive graph \mathcal{G} , optimal policies of the searcher and the target are specified by equal vectors $\vec{p}_{[\text{ser}]}^* = \vec{p}_{[\text{tar}]}^*$ such that $p^*(s_i) = p_{[\text{ser}]}^*(s_i) = p_{[\text{tar}]}^*(s_i) = 1/n$, $i = 1, 2, \dots, n$.*

Proof. Since \mathcal{G} is vertex-transitive, the set of optimal policies of both the searcher and the target is convex, and $\vec{p}^* = (p^*(s_1), p^*(s_2), \dots, p^*(s_n))$ is a convex combination of these optimal policies. \blacksquare

Note that Theorem 3.3 implies that the value of the game (Equation 3.64) on vertex-transitive graph \mathcal{G} is specified as follows:

$$V_{\text{PEGG}}(\mathcal{G}) = V_{\text{PEGG}}(\vec{p}_{[\text{ser}]}^*, \vec{p}_{[\text{tar}]}^*) = \vec{p}^{*T} \times \Delta(\mathcal{G}) \times \vec{p}^*.$$

In particular, Chung *et al.* [39] indicated that if graph \mathcal{G} is complete, $\mathcal{G} = \mathcal{K}_n$, then $V_{\text{PEGG}}(\mathcal{K}_n) = n - 1/n$, and if \mathcal{G} is a cycle graph, $\mathcal{G} = \mathcal{C}_n$, then $V_{\text{PEGG}}(\mathcal{C}_n) = n^2 - \varepsilon/4n$,

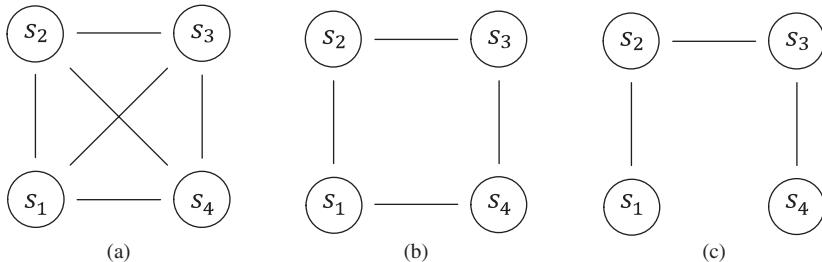


Figure 3.10 Examples of complete, cyclic, and path graphs. (a) Complete graph \mathcal{K}_4 . (b) Cycle graph \mathcal{C}_4 . (c) Path graph \mathcal{P}_4 .

where $\varepsilon = 0$ if n is even, and $\varepsilon = 1$ is n is odd. For example, for the graph \mathcal{K}_4 (see Figure 3.10a) we have

$$\Delta(\mathcal{K}_4) = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \text{and} \quad \overrightarrow{p}^* = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix},$$

and $V_{\text{PEGG}}(\mathcal{K}_4) = \overrightarrow{p}^{*T} \times \Delta(\mathcal{K}_4) \times \overrightarrow{p}^* = 3/4$. Similarly, for the cyclic graph \mathcal{C}_4 (see Figure 3.10b) it follows that

$$\Delta(\mathcal{C}_4) = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{pmatrix} \quad \text{and} \quad \overrightarrow{p}^* = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix}.$$

Hence, $V_{\text{PEGG}}(\mathcal{C}_4) = \overrightarrow{p}^{*T} \times \Delta(\mathcal{C}_4) \times \overrightarrow{p}^* = 1$.

Finally, let us consider the values of the pursuit-evasion game on unnecessary vertex-transitive graphs, and especially on the path graph \mathcal{P}_n (see Figure 3.10c).

Theorem 3.4 [39] Let \mathcal{G} be a finite connected graph with n vertices. Then, for the value $V_{\text{PEGG}}(\overrightarrow{p}_{[\text{ser}]}^*, \overrightarrow{p}_{[\text{tar}]}^*) = V_{\text{PEGG}}(\mathcal{G})$ of the pursuit-evasion game on the graph \mathcal{G} the following holds:

1. If \mathcal{G} is not a complete graph, $\mathcal{G} \neq \mathcal{K}_n$, then $1 \leq V_{\text{PEGG}}(\mathcal{G}) \leq \text{rad}(\mathcal{G})$.
2. If $\mathcal{G} = \mathcal{P}_n$ is a path graph, then $V_{\text{PEGG}}(\mathcal{P}_n) \leq (n-1)/2$, otherwise $V_{\text{PEGG}}(\mathcal{G}) \leq (n-2)/2$.

Proof. Let us consider the first statement. Since the searcher and the target can choose one of the central vertices, the right part of the inequality holds.

Let us demonstrate the left part of the inequality. Let $y, y' \in S$ be two non-adjacent vertices of the graphs. One of the target's policies $\overrightarrow{p}'_{[\text{tar}]}$ specifies equal probabilities $p'_{[\text{tar}]}(y) = p'_{[\text{tar}]}(y') = 1/2$ for these vertices and zero probabilities for the others. Assume that the best searcher's policy, which corresponds to the target's policy $\overrightarrow{p}'_{[\text{tar}]}$, is $\overrightarrow{p}_{[\text{ser}]}$. Then, the value $\overrightarrow{p}'_{[\text{tar}]}^T \times \Delta(\mathcal{G}) \times \overrightarrow{p}_{[\text{ser}]}$ of the game for this pair of policies is at least $1/2 \sum_{s \in S} (d(s, y) + d(s, y')) p_{[\text{ser}]}(s) \geq 1$. Hence,

$$\begin{aligned} V_{\text{PEGG}}(\mathcal{G}) &= \max_{\overrightarrow{p}_{[\text{tar}]}} \min_{\overrightarrow{p}_{[\text{ser}]}} \left\{ \overrightarrow{p}'_{[\text{tar}]}^T \times \Delta(\mathcal{G}) \times \overrightarrow{p}_{[\text{ser}]} \right\} \\ &\geq \min_{\overrightarrow{p}_{[\text{ser}]}} \left\{ \overrightarrow{p}'_{[\text{tar}]}^T \times \Delta(\mathcal{G}) \times \overrightarrow{p}_{[\text{ser}]} \right\} \geq 1, \end{aligned}$$

as required by the left part of the inequality, and thus the first inequality is proved.

Now let us demonstrate the second inequality. Assume that $\mathcal{G} \neq \mathcal{P}_n$. Then graph \mathcal{G} contains a spanning tree \mathcal{T} with n vertices, and includes at least one vertex s such that $\deg(s) \geq 3$. Let \mathcal{T}' be a subtree of the tree \mathcal{T} , which is obtained by deleting all leaves (end-vertexes) of \mathcal{T} with incident edges. Then, the subtree \mathcal{T}' has $n' \leq n - 3$ vertices. Hence, $\text{rad}(\mathcal{T}) = \text{rad}(\mathcal{T}') + 1$. Since $\text{rad}(\mathcal{G}) \leq (n-1)/2$ for any graph $\mathcal{G} \neq \mathcal{K}_n$ and $n' \leq n - 3$,

it follows that $\text{rad}(\mathcal{T}') + 1 \leq (n' - 1)/2 + 1 \leq (n - 2)/2$. Consequently, recalling that for any graph \mathcal{G} and its spanning tree \mathcal{T} it follows that $\text{rad}(\mathcal{G}) \leq \text{rad}(\mathcal{T})$, and using the right side of the first inequality, one obtains

$$V_{\text{PEGG}}(\mathcal{G}) \leq \text{rad}(\mathcal{G}) \leq \text{rad}(\mathcal{T}) \leq n - 2/2.$$

Now let $\mathcal{G} = \mathcal{P}_n$. Then its spanning tree $\mathcal{T} = \mathcal{P}_n$ is also a path graph and a subtree $\mathcal{T}' = \mathcal{P}_{n'}$ includes exactly $n' = n - 2$ vertices. Hence, regarding the radius of \mathcal{T}' , it follows that

$$\text{rad}(\mathcal{T}') + 1 \leq \frac{n' - 1}{2} + 1 = \frac{n - 1}{2}$$

and by the same reasoning, one obtains

$$V_{\text{PEGG}}(\mathcal{P}_n) \leq \text{rad}(\mathcal{P}_n) \leq \frac{n - 1}{2},$$

as required. ■

Note that, following Chung *et al.* [39], the upper bound of the value of the game on the path graph \mathcal{P}_n , which appears in the second inequality in Theorem 3.4, can be considered as an exact value; that is, in parallel to the above-given values of the game on the graphs \mathcal{K}_n and \mathcal{C}_n , it can be specified that $V_{\text{PEGG}}(\mathcal{P}_n) = (n - 1)/2$.

Like the game theory model of search and screening (see Section 3.4.1) and PPEG (see Section 3.4.2), the considered PEGGs guarantee the existence of optimal searcher's and target's policies and provides the value of the game for such policies. If the target is not informed about the searcher's action, the search procedures can be considered as pursuit-evasion games with nature [48], while the target's movements are defined by random walks over graphs [49–51]. In such a case, an optimal policy of the searcher guarantees finding the target in all probability; however, the values of the game are specified similarly to the MDP and POMDP models of search (see Sections 3.1 and 3.2).

Now, based on the existence of the searcher's policy, let us consider the heuristic A*-type algorithm (see Section 2.3.1), which is called the Min–Max LRTA* algorithm [40, 41] and provides a method for real-time search, creating a near-optimal searcher's path on the graph. The Min–Max LRTA* algorithm generalizes the LRTA* algorithm (Algorithm 2.13), which was considered in Section 2.3.2 within the framework of real-time search over graphs, and can be considered as a probabilistic extension of the MTS algorithm (Algorithm 2.14) presented in Section 2.3.3.

Let us start with a definition of the pursuit-evasion game in the form of search for a static target, while the searcher's movements are determined by the target policy. Let $\mathcal{G} = (S, E)$ be a vertex-transitive graph with at least five vertices. Assume that the target starts at a vertex $y_{\text{init}} \in S$ and follows a path over the graph \mathcal{G} according to optimal policy $\overrightarrow{p}_{[\text{tar}]}$, while the searcher starts at a vertex $s_{\text{init}} \in S$ and follows optimal policy $\overrightarrow{p}_{[\text{ser}]}$. Let $d(s_{\text{init}}, y_{\text{init}}) \geq 2$, and assume that between s_{init} and y_{init} there exists a path $a[s_{\text{init}}, y_{\text{init}}]$ with a length of at least three. Then, since graph \mathcal{G} is vertex-transitive, the goals of the searcher and the target are opposite, and the game continues infinitely, the distance $d(s_{\text{curr}}, y_{\text{curr}})$ between the current location of the searcher s_{curr} and that of the target y_{curr} is preserved, $d(s_{\text{curr}}, y_{\text{curr}}) \geq d(s_{\text{init}}, y_{\text{init}}) - 1$, over all the game.

The same situation can be described as follows. Assume that the target chooses a vertex $y_{fin} \in S$ and does not change its location during the game, while the searcher starts at a vertex $s_{init} \in S$ and moves over the graph. Similar to the above, we assume that $d(s_{init}, y_{init}) \geq 2$ and that a path $a[s_{init}, y_{init}]$ with a length of at least three exists. As indicated at the beginning of the section, for each searcher's location $s_{curr} \in S$, the edges $(s_{curr}, s) \in E$ specify possible one-step movements. Then, the target's policy can be represented by restricting the searcher's possible steps such that the distance $d(s_{curr}, y_{fin})$ between the searcher's current location s_{curr} and the target location y_{fin} will be preserved as $d(s_{curr}, y_{fin}) \geq d(s_{init}, y_{fin}) - 1$. For a cycle graph C_5 the indicated duality between the policies is illustrated Figure 3.11. In the figure, the evolution of the game is from left to right and from top to bottom.

In Figure 3.11a, the searcher starts at the vertex $s_{init} = s_1$ and the target starts at the vertex $y_{init} = s_4$. The distance between these locations is $d(s_{init}, y_{init}) = 2$. At first, the searcher moves to the vertex $s_{curr} = s_5$; thus $d(s_{curr}, y_{init}) = 1$. Then the target moves to the vertex $y_{curr} = s_3$ and the distance becomes $d(s_{curr}, y_{curr}) = 2$. Finally, the searcher moves to the vertex $s_{curr} = s_4$ and the target moves to the vertex $y_{curr} = s_2$, so the distance is still $d(s_{curr}, y_{curr}) = 2$. In parallel, Figure 3.11b illustrates the game dynamics while the target restricts the searcher's possible movements; since the target is static, we denote its location by y_{fin} . As in the previous case, the searcher starts at the vertex $s_{init} = s_1$ and the target is located at the vertex $y_{fin} = s_4$, and $d(s_{init}, y_{fin}) = 2$. The first movement of the searcher is to the vertex $s_{curr} = s_5$, so, as above, $d(s_{curr}, y_{fin}) = 1$. After this step, following its policy the target restricts the searcher's neighborhood and prohibits movement from $s_{curr} = s_5$ to $y_{fin} = s_4$; then the searcher moves to the vertex $s_{curr} = s_1$. The distance in this case is $d(s_{curr}, y_{fin}) = 3$. At the third step, the target restricts the searcher's movement from $s_{curr} = s_1$ to the vertex s_5 (note that since the target restricts a current searcher's movement only, the previously deleted edge (s_4, s_5) appears in its place). Thus the searcher moves to the vertex $s_{curr} = s_2$ and the distance is $d(s_{curr}, y_{fin}) = 2$.

Following such duality in the definition of the searcher's and the target's policies, the pursuit-evasion game can be specified as search for a static target, while at each step the neighborhood of the searcher is determined by the target's policy. The Min–Max LRTA* algorithm [40, 41] deals with such a specification of the pursuit-evasion game and provides near-optimal policy of the searcher in real time. Before considering the actions of this algorithm, let us recall the concepts used in real-time search algorithms (see Section 2.3.2) and MTS algorithms (see Section 2.3.3).

Let $\mathcal{G} = (S, E)$ be a finite connected graph, and assume that the target is located at the vertex $y_{fin} \in S$ and the searcher's current location is $s_{curr} \in S$, which may be equivalent to the initial state s_{init} . Similar to the indicated algorithms, for any vertex $s \in S$, denote by $a[s_{curr}, s]$ a path which starts at the vertex s_{curr} and by $a[s, y_{fin}]$ a path which starts at the vertex s and ends at the vertex y_{fin} . Recall that for the paths $a[s_{curr}, s]$ and $a[s, y_{fin}]$, the cost $\bar{c}(a[s_{curr}, s])$ and estimated cost $\tilde{c}(a[s, y_{fin}])$ were defined and characterize the chosen paths. Following Equation 2.50, in the real-time search algorithms on graphs, the evaluation cost $c_{RT}(s)$ of the state $s \in S$ is defined as the value

$$c_{RT}(s) = \bar{c}(a[s_{curr}, s]) + \tilde{c}(a^*[s, y_{fin}]),$$

where $a^*[s, y_{fin}]$ stands for the optimal path from the vertex s to vertex y_{fin} , and all cost functions are such that the optimal path has minimal cost. As above, denote by $\tilde{c}(s) = \tilde{c}(a^*[s, y_{fin}])$ an estimated cost of the optimal path from the vertex s to the final vertex y_{fin} .

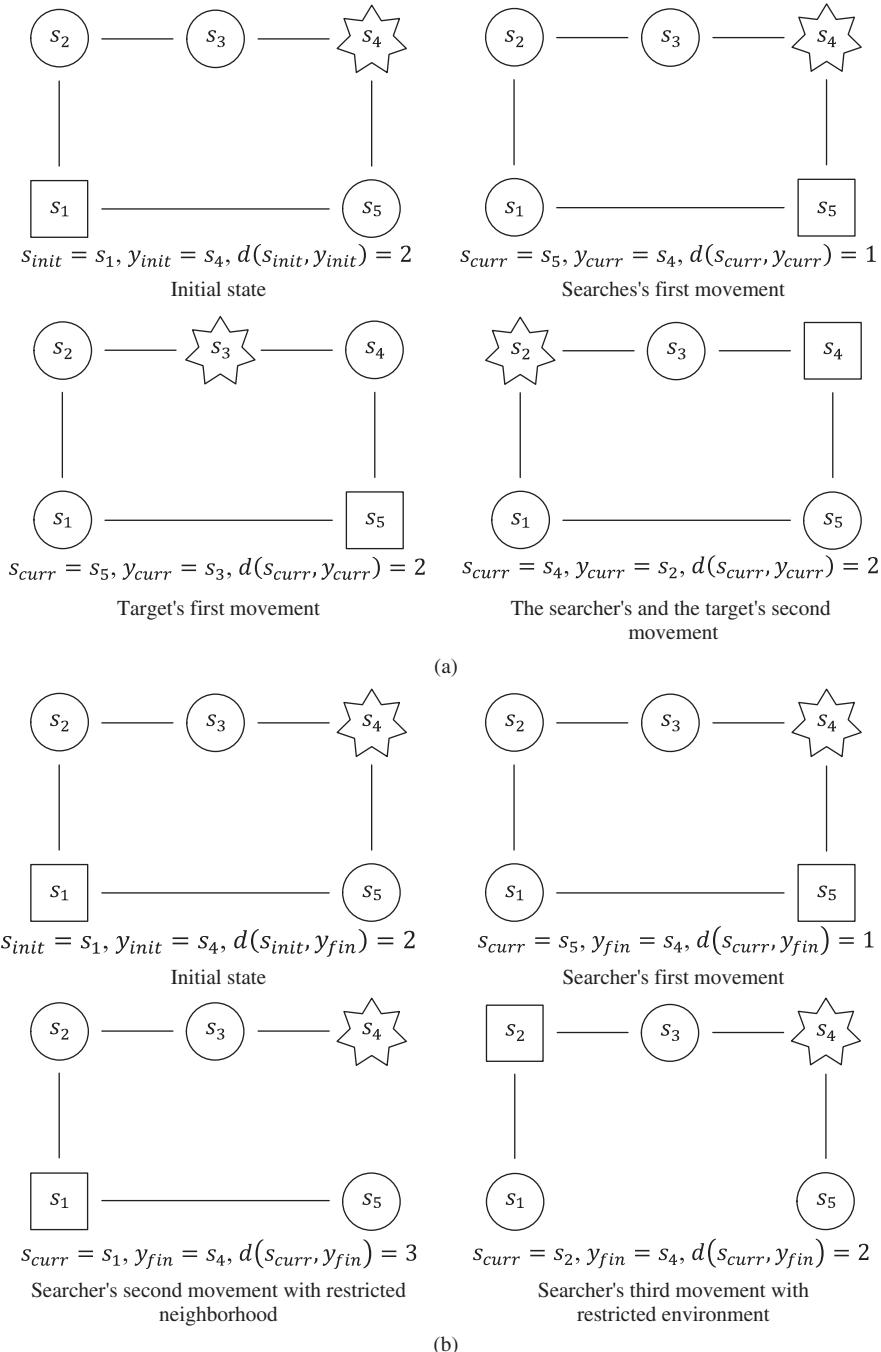


Figure 3.11 Dual definitions of the target's policies. (a) Moving target and unrestricted searcher's movements. (b) Static target and restricted searcher's movements.

In search for a static target, the choice of the path and, consequently, the value of the cost depend on the searcher's policy, which is provided by the RTA* algorithm (Algorithm 2.12) and LRTA* algorithm (Algorithm 2.13). Now, let us define the costs while the target specifies the available searcher's movements, as specified by the considered pursuit-evasion game. Assume that the searcher's current vertex is s_{curr} .

In the pursuit-evasion game, which is addressed by the Min–Max LRTA* algorithm [40, 41], the choice of next vertex $s_{next} \in N(s_{curr})$ from the set $N(s_{curr})$ of neighbors of the current vertex s_{curr} depends on the target's policy. To stress this dependence, let us denote the neighborhood of the vertex s_{curr} by $N(s_{curr} | \mathfrak{d}_{[tar]})$, where $\mathfrak{d}_{[tar]}$ is a target's policy of choosing the vertices, which are available for the searcher from the vertex s_{curr} . Equivalently, it can be specified that, given a neighborhood $N(s_{curr})$ of the vertex s_{curr} , the target's policy $\mathfrak{d}_{[tar]}$ results in a probability distribution on the set edges $\{(s_{curr}, s) | s \in N(s_{curr})\}$, which are adjacent to the vertex s_{curr} .

The actions of the Min–Max LRTA* algorithm are similar to the actions of its basic LRTA* algorithm (Algorithm 2.13); however, the updating of the estimated cost and the criterion for choosing the next vertex follow a game theory approach. As indicated above, the goal of the searcher in the PEGG is to minimize the cost of the search, and the goal of the target is to maximize this cost. Then, according to the min–max definition of the value of the game, the Min–Max LRTA* algorithm is outlined as follows.

Algorithm 3.1 (Min–Max LRTA* algorithm) [40, 41]. Given graph $\mathcal{G} = (S, E)$, states s_{init} and y_{fin} , $s_{init}, y_{fin} \in S$, cost function c_{RT} , and initial costs $c_0(s)$, $s \in S$, do:

1. Define $ResPath$ as a list of states.
2. Set $s_{curr} = s_{init}$.
3. While $s_{curr} \neq y_{fin}$ do:
 - 3.1. Expand s_{curr} to obtain the set $N(s_{curr})$ of its successors.
 - 3.2. Set $\tilde{c}(s_{curr}) = \max \left\{ \tilde{c}(s_{curr}), \max_{s \in N(s_{curr} | \mathfrak{d}_{[tar]})} \{c_{RT}(s)\} \right\}$.
 - 3.3. Set $s_{next} = \operatorname{argmin}_{s \in N(s_{curr} | \mathfrak{d}_{[tar]})} \max_{s \in N(s_{curr} | \mathfrak{d}_{[tar]})} \{c_{RT}(s)\}$; ties are broken randomly.
 - 3.4. $ResPath.include(s_{curr})$.
 - 3.5. Set $s_{curr} = s_{next}$.
4. Return $ResPath$. ■

The actions of the Min–Max LRTA* algorithm are clear from the previous consideration. Starting from the initial state $s_{curr} = s_{init}$, the algorithm updates the estimated cost $\tilde{c}(s_{curr})$ following the pessimistic policy. In particular, if, according to its policy, the target restricts the available movements of the searcher, as shown in Figure 3.11b, while the weights of the remaining edges are equal to unity, then

$$c_{RT}(s) = 1 + \max_{s \in N(s_{curr} | \mathfrak{d}_{[tar]})} \{\tilde{c}(s)\},$$

and, following Line 3.2, the cost $\tilde{c}(s_{curr})$ is updated to the maximal estimated cost from the neighboring states to the final state y_{fin} with the addition of the unit cost $\bar{c}(a[s_{curr}, s]) = 1$.

After this updating, the next state s_{next} in Line 3.3 is chosen according to min–max criteria; that is, from the states with maximal estimated cost a minimal one is chosen. As above, if the target acts so that $\bar{c}(a[s_{curr}, s]) = 1$, then

$$s_{next} = \operatorname{argmin}_{s \in N(s_{curr} | \mathfrak{d}_{[tar]})} \max_{s \in N(s_{curr} | \mathfrak{d}_{[tar]})} \{\tilde{c}(s)\}.$$

When the final state y_{fin} is reached, the algorithm returns the resulting path $ResPath = a[s_{init}, y_{fin}]$.

Note that the path $a[s_{init}, y_{fin}]$ is not necessarily optimal. Moreover, if there exists such a policy of the target, which prevents its finding, termination of the Min–Max LRTA* algorithm is not guaranteed. However, if there exists such a searcher’s policy, which guarantees the existence of the path $a[s_{init}, y_{fin}]$ for any target’s policy $\mathfrak{d}_{[tar]}$, and for the estimated costs, it follows that:

1. $\tilde{c}(s_{init}) = \tilde{c}(a^*[s_{init}, y_{fin}]) \leq \bar{c}(a^*[s_{init}, y_{fin}])$, where $\bar{c}(a^*[s_{init}, y_{fin}])$ is the actual cost of the optimal path searcher’s $a^*[s_{init}, y_{fin}]$ given the target’s policy $\mathfrak{d}_{[tar]}$.
2. $\tilde{c}(s_{curr}) \leq \min_{s \in N(s_{curr} | \mathfrak{d}_{[tar]})} \max_{s \in N(s_{curr} | \mathfrak{d}_{[tar]})} \{c_{RT}(s)\}$ for any $s_{curr} \in S$.

Then the Min–Max LRTA* algorithm results with near-optimal path $a[s_{init}, y_{fin}]$. The first requirement is parallel to the admissibility assumption (Equation 2.48) and the second assumption is a min–max version of the consistency assumption (Equation 2.49), which are implemented in the real-time search algorithms in Section 2.3.2. For additional information regarding the Min–Max algorithm see [40, 41]; the particular proofs of its properties are given in Koenig [40].

This algorithm finalizes our consideration of the PEGGs and, in general, of the game theory models of search. As indicated at the beginning of the section, we have presented a narrow class of search games that follow the models of search. Complete information regarding search games and their application for different tasks can be found in the books by Alpern and Gal [33], Gal [34], and Garnaev [35].

3.5 Summary

In this chapter we presented several models of search and decision making. The first is the basic MDP that supports sequential decision making and provides unified solutions for a wide range of search problems. In MDP the search process is defined over the set of states, each of which is associated with the observed area along with the possible observation result and possible set of search actions (search policies). The model can be designed for special cases such as partially observed MDP, HMMs, and for a dynamic programming search scheme that often tackles computational complexity aspects in practical search problems. The described general framework can also model classical search problems, such as *search and screening* and the *pursuit-evasion game* if the target is informed about the searcher’s actions and tries to escape from the searcher. The use of MDP is well described by search decision trees that are often based on information-seeking and entropy criteria. An important class of decision trees is obtained by applying the Huffman–Zimmermann procedure that links the search problems to well-known information theory and coding procedures over the location probabilities. When more than one location is searched simultaneously, the group-testing approach can be applied. The relation of search problems to coding procedures is

well known for the case of a static target but is less trivial in the case of a moving target that is considered here as well. Few optimal solutions have been proposed over the years using an MDP framework for relatively small search problems. Some examples are the Pollock model for an optimal search over a space with only two points; the continuous time Dobbie two-cell model of search; the Ross model that generalizes the Pollock model of search to a sample space with an arbitrary number of points and a minimal expected cost optimization; and the Eagle model with both finite and infinite horizons. These models are very limited in their practical aspects, yet they are important for gaining insights into the nature of the search solutions as well as for benchmarking purposes with respect to proposed heuristics that can then be applied to larger and more practical search scenarios. Branch-and-bound search procedures are also considered as a general framework for decision making and path planning. Finally, the chapter briefly addresses models for the search of a target that is trying to escape the searcher. This is proposed in a game theory framework, such as the PPEGs.

References

1. A. Wald (1950). *Statistical Decision Functions*. John Wiley & Sons, Ltd: Chichester.
2. R. Bellman (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, **6** (5), 679–684.
3. C. Derman (1970). *Finite State Markov Decision Processes*. Academic Press: New York.
4. D.J. White, (1993). *Markov Decision Processes*. John Wiley & Sons, Ltd: Chichester.
5. S.M. Ross (2003). *Introduction to Probability Models*. Academic Press: San Diego, London.
6. W. Feller (1970). *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, Ltd: New York.
7. H. M. Taha (1997). *Operations Research: An Introduction*. Prentice-Hall International Inc.: Upper Saddle River, NJ.
8. W.L. Winston (2004). *Operations research Applications and Algorithms*. Thomson Learning: Belmont, CA.
9. T. M. Cover, J. A. Thomas (1991). *Elements of Information Theory*. John Wiley & Sons Inc.: New York.
10. G. E. Monahan (1982). A survey of partially observable Markov decision processes: theory, models, and algorithms. *Management Science*, **28** (1), 1–16.
11. L.R. Rabiner (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, **77** (2), 257–286.
12. D. P. Bertsekas (1995). *Dynamic Programming and Optimal Control*. Athena Scientific Publishers: Boston.
13. Y. Ephraim, N. Merhav (2002). Hidden Markov processes. *IEEE Transactions on Information Theory*, **48** (6), 1518–1569.
14. M. Aoki (1967). *Optimization of Stochastic Systems*. Academic Press: New York, London.
15. K.J. Åström (1970). *Introduction to Stochastic Control Theory*. Academic Press: New York.
16. A. Cassandra, Littman, M. L., Zhang, N. L. (1997). Incremental pruning: a simple, fast, exact method for partially observable Markov decision processes. Proceedings of 13-th Annual Conference on Uncertainty in Artificial Intelligence ‘UAI-97’, Morgan Kaufmann, San Francisco, CA, pp. 54–61.
17. S.M. Ross (1970). *Applied Probability Models with Optimization Applications*. Holden-Day: San Francisco, CA.

18. M. B. Hurley (2003, 2005). An extension of statistical decision theory with information theoretic cost functions to decision fusion: Part I. *Information Fusion*, **4**, 319–330; Part II. *Information Fusion*, **6**, 165–174.
19. R.S. Sutton, A.G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press: Cambridge, MA.
20. S.M. Pollock (1970). A simple model of search for a moving target. *Operations Research*, **18**, 883–903.
21. P.J. Schweitzer (1971). Threshold probabilities when searching for a moving target. *Operations Research*, **19**(3), 707–709.
22. J. M. Dobbie (1974). A two-cell model of search for a moving target. *Operations Research*, **22** (1), 79–92.
23. S.M. Ross (1969). A problem in optimal search and stop. *Operations Research*, **17**(6), 984–992.
24. S.M. Ross (1983). *Introduction to Stochastic Dynamic Programming*. Academic Press: New York.
25. J. N. Eagle (1984). The optimal search for a moving target when the search path is constrained. *Operations Research*, **32**, 1107–1115.
26. S. Singh, V. Krishnamurthy (2003). The optimal search for a moving target when the search path is constrained: the infinite-horizon case. *IEEE Transactions on Automatic Control*, **48** (3), 493–497.
27. J. N. Eagle, J. R. Yee (1990). An optimal branch-and-bound procedure for the constrained path, moving target search problem. *Operations Research*, **38** (1), 110–114.
28. T.J. Stewart (1979). Search for a moving target when searcher motion is restricted. *Computers and Operation Research*, **6**, 129–140.
29. A.R. Washburn (1998). Branch and bound methods for a search problem. *Naval Research Logistics*, **45**, 243–257.
30. R. F. Dell, J. N. Eagle, G. H. A. Martins, A. G. Santos (1996). Using multiple searchers in constrained-path, moving-target search problems. *Naval Research Logistics*, **43**, 463–480.
31. P. J. Nahin (2007). *Chases and Escapes: The Mathematics of Pursuit and Evasion*. Princeton University Press: Princeton, NJ.
32. R. Isaacs (1965). *Differential Games: a Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley and Sons, Inc.: New York, London, Sydney.
33. S. Alpern, S. Gal (2003). *The Theory of Search Games and Rendezvous*. Kluwer Academic Publishers: New York.
34. S. Gal. *Search Games*. Academic Press: New York, 1988.
35. A. Y. Garnaev (2000). *Search Games and Other Applications of Game Theory*. Springer–Verlag: Berlin.
36. J. N. Eagle, A. R. Washburn (1991). Cumulative search-evasion games. *Naval Research Logistics*, **38**, 495–510.
37. J. P. Hespanha, M. Prandini, S. Sastry (2000). Probabilistic pursuit–evasion games: a one-step Nash approach. Proceedings of 39-th IEEE Conference on Decision Making and Control, pp. 2272–2277.
38. J. P. Hespanha, M. Prandini, S. Sastry (1999). Multi-agent probabilistic pursuit–evasion games. Proceedings of 38-th IEEE Conference on Decision Making and Control, Vol. 3, pp. 2432–2437.
39. F.R.K. Chung, J. E. Cohen, R. L. Graham (1988). Pursuit–evasion games on graphs. *Journal of Graph Theory*, **12** (2), 159–167.
40. S. Koenig (2001). Mini–max real–time heuristic search. *Artificial Intelligence*, **129** (1–2), 165–197.

41. S. Koenig, R.G Simmons. (1995). Real-time search on non-deterministic domains. Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-95), pp. 1660–1667.
42. G.W Brown. (1949). Some Notes on Computation of Games Solutions. Technical Report P-78, RAND Corporation, Santa-Monica, CA.
43. J. Robinson (1951). An iterative method of solving a game. *Annals of Mathematics*, **54** (2), 296–301.
44. L.A. Pertosjan, N.A. Zenkevich (1996). *Game Theory*. World Scientific: Singapore.
45. J. Filar, K. Vrieze (1997). *Competitive Markov Decision Processes*. Springer–Verlag: New York.
46. A. Arapostathis, V.S. Borkar, E. Fernandez–Gaucherand, M. K. Ghosh, S. I. Markus (1993). Discrete-time controlled Markov processes with average cost criterion: a survey. *SIAM Journal on Control and Optimization*, **31** (2), 282–344.
47. E. B. Dynkin, A. A. Yushkevich (1979). *Controlled Markov Processes*. Springer–Verlag: New York.
48. P. D. Grünwald, A. P. David (2004). Game theory, maximum entropy, minimum discrepancy and robust Bayesian decision theory. *The Annals of Statistics*, **32** (4), 1367–1433.
49. R. Burioni and D Cassi (2005). Random Walks on Graphs: Ideas, Techniques and Results. *e–publ.*: arXiv:cond–mat/0507142 v1 6 Jul 2005.
50. D. Coppersmith, P. Doyle, P. Raghavan (1993). Random walks on weighted graphs, and applications to on-line algorithms. *Journal of the ACM*, **40** (3), 421–453.
51. L. Lovász (1993). Random walks on graphs: a survey. *Boya Society Mathematical Studies*, **2**, 1–46.

4

Methods of information theory search

In the previous chapters, we considered the methods and algorithms of search and presented a unified model of search that is based on Markov decision processes. In particular, the algorithms of search over weighted graphs were described and their relation with group-testing methods was stressed. In this chapter, we develop a search procedure that is based on the partitions space, as obtained by the searcher and the target moves. For this purpose we use the Rokhlin distance between the partitions as distance measures.

We formulate the algorithm of search for static targets and prove its main properties. The algorithm is based on the Learning Real-Time A* (LRTA*) algorithm, which is presented in Section 2.3.2, and since it implements informational distances, it is also called the Informational Learning Real-Time A* (ILRTA*) algorithm. This algorithm acts on the partitions space and therefore has proper relations with the group-testing algorithms, especially with the Huffman–Zimmerman search procedure and the Generalized Optimal Testing Algorithm (GOTA), which were considered in Section 2.2.4. In this chapter, we consider such relations and show that under certain conditions the algorithm acts as the optimal Huffman–Zimmerman search procedure or as the near-optimal GOTA.

We consider the informational algorithm of search for a moving target. This algorithm is based on the Moving Target Search (MTS) algorithm, which was considered in Section 2.3.3, and implements the same informational distances as the ILRTA* algorithm; it is therefore called the Informational Moving Target Search (IMTS) algorithm. As for the algorithm of search for static targets, the IMTS algorithm acts over the partitions space, thus it also can be related with the group-testing search procedures. In addition, we demonstrate the relation of the algorithm with the Pollock model of search, which is presented in Section 3.2.2, and show that under the conditions of the Pollock simple optimal model of search, it gives the same optimal solutions for both certain and uncertain detections.

Further, we present certain generalizations and, on the basis of the ILRTA* algorithm, demonstrate search with multiple searchers for both cooperative and non-cooperative behavior. The theoretical results are also illustrated by numerical examples.

4.1 Entropy and informational distances between partitions

Let us start with formal definitions of the structures and measures that specify the dynamics of the search procedure over the sample space $X = \{x_1, x_2, \dots, x_n\}$. As indicated in Section 2.1.1, such dynamics is represented by a change in the location probabilities of the target, which are specified by the probability mass function $p^t : X \rightarrow [0, 1]$, $t = 0, 1, 2, \dots$, and the observed and estimated probabilities, which are specified by the probability mass functions $\bar{p}^t : X \rightarrow [0, 1]$ and $\tilde{p}^t : X \rightarrow [0, 1]$, respectively. In addition, in the previous chapters, especially in the considerations of group-testing search in Section 2.2.4, the systems of partitions of the sample space X and entropy measures both for location probabilities and for partitions were implemented. Now let us consider these concepts more formally.

Let (Ω, \mathcal{B}, P) be a probability space, where Ω is a compact measurable set, \mathcal{B} is a σ -algebra of the subsets of Ω , and P is a probability measure on Ω , and let $x = x(\omega)$, $\omega \in \Omega$, be a random variable defined on the set $X = \{x_1, x_2, \dots, x_n\}$, which is considered as an alphabet for the variable x . Let $\eta = \{D_1, D_2, \dots, D_n\}$, where $D_i \in \mathcal{B}$, $D_i \cap D_j = \emptyset$ if $i \neq j$, $\cup_{D \in \eta} D = \Omega$, and $P(D_i) > 0$, for any $D_i \in \mathcal{B}$, $i = 1, \dots, n$, be a partition of Ω . Then, for any set $A \in \mathcal{B}$ from the σ -algebra \mathcal{B} it follows that [1]

$$x(\omega) = \sum_{i=1}^n P(A|D_i) \times \mathbf{1}(D_i, \omega),$$

where $P(A|D_i) = p(A \cap D_i)/p(D_i)$ is a conditional probability of A given D_i , and $\mathbf{1}(D_i, \omega)$, as above, is an indicator of the set D_i such that $\mathbf{1}(D_i, \omega) = 1$ if $\omega \in D_i$ and $\mathbf{1}(D_i, \omega) = 0$ otherwise. If $A = \Omega$, then the random variable x depends only on the partition η , thus

$$x(\omega) = \sum_{i=1}^n P(D_i) \times \mathbf{1}(D_i, \omega).$$

Such a representation of a random variable leads to a dual form of the system dynamics. On the one hand, the dynamics may be described by modifications of the probability measure P and, on the other hand, it may be considered as a choice of different partitions η , which define the random variable x . From a historical point of view, a dual form of the system dynamics definition gave rise to studies of the relations between ergodic theory, which is considered an important part of the modern theory of dynamical systems, and information theory. Such considerations were initiated in 1965 by Billingsley [2] and contributed to the development of information theory by a measure-theoretic point of view [3]. This view provides a basis for an information theory approach which differs from Shannon's original approach [4]. For an overview of the modern status of studies in the field, see, for example, [5].

Within the framework of the ergodic theory and information theory, a number of distance measures were defined between the states of dynamical systems or stochastic processes. In particular, two distance measures were suggested to meet the metric requirements: the first is the Rokhlin distance [6], which was introduced in 1967 on the basis of the Shannon

entropy of partition; and the second is the Ornstein distance [7, 8], which was introduced in 1971 as a probabilistic form of the Hamming distance.

In general, most of the applications of the Ornstein distance were related to studies of stochastic processes [8] and in recent years also to coding theory. For an overview of the results in the field, see, for example, [9]. The Rokhlin distance, in contrast, was mostly applied to the dynamical systems [10, 11] as a characteristic of the ‘complexity’ of the system dynamics by an entropy measure. In 2006, Ornstein and Weiss [12] showed that entropy can be viewed as a unique characteristic of stochastic processes that determines the type of processes and can be integrated within the framework of dynamical systems and information theory. Similar conclusions were suggested in Ruschin-Rimini *et al.* [13] who used entropy measures to find change points and for anomaly detection in various nonlinear stochastic processes.

Let us start with the definition of the Rokhlin distance [6]. Let (Ω, \mathcal{B}, P) be a probability space and let us fix a probability measure p on Ω . Let an exhaustive and mutually exclusive finite set $\alpha = \{A | A \in \mathcal{B}\}$, where $A_i \cap A_j = \emptyset$, $i \neq j$, and $\cup_{A \in \alpha} A = \Omega$ is a partition of Ω . Recall that in consideration of the group-testing search, the entropy and the conditional entropy of partitions play important roles (see Equations 2.41 and 2.42). Let us, for convenience, return to these definitions.

Given a probability measure p on Ω and a partition $\alpha = \{A | A \in \mathcal{B}\}$, the value $p(A) = \sum_{x \in A} p(x)$, $A \in \alpha$, stands for the probability of the set A . Then, the *entropy of the partition* α is defined as follows:

$$H(\alpha) = -\sum_{A \in \alpha} p(A) \log p(A), \quad (4.1)$$

where it is assumed that $p(\emptyset) \log p(\emptyset) = 0 \log 0 = 0$. In addition, let $\beta = \{B | B \in \mathcal{B}\}$, $B_i \cap B_j = \emptyset$, $i \neq j$, and $\cup_{B \in \beta} B = \Omega$ is another partition of Ω . Then, the *conditional entropy of partition* α given partition β is the following value:

$$H(\alpha|\beta) = -\sum_{B \in \beta} \sum_{A \in \alpha} p(A, B) \log p(A|B), \quad (4.2)$$

where $p(A, B) = p(A \cap B)$ and $p(A|B) = p(A \cap B)/p(B)$. As defined above, $0 \log 0 = 0$.

Let α and β be two partitions. Recall that the relation ‘ \prec ’ between the partitions α and β is called *refinement*, and if for every $A \in \alpha$ there exists a set $B \in \beta$ such that $A \subset B$, then one says that α refines β or that α is a *refinement* of β and writes $\beta \prec \alpha$.

The *multiplication* $\alpha \vee \beta$ of the partitions α and β is itself a partition such that it consists of all possible intersections $A \cap B$ of the sets $A \in \alpha$ and $B \in \beta$ from the partitions α and β . The multiplication of m partitions α_j , $j = 1, 2, \dots, m$, is denoted by $\vee_{j=1}^m \alpha_j$.

The next lemma summarizes the basic properties of the entropy and the conditional entropy of partitions. These properties are used in search algorithms that apply the metric properties of the information distances.

Lemma 4.1 [10, 11]. *Let α and β be two partitions of the same set. Then it follows that:*

1. *If $\alpha \prec \beta$ then $H(\alpha) \leq H(\beta)$.*
2. *$H(\alpha|\beta) = 0$ if and only if $H(\alpha) \leq H(\beta)$.*
3. *If $\alpha \prec \alpha'$ then $H(\alpha|\beta) \leq H(\alpha'|\beta)$, where α' is a partition of the same set.*
4. *If $\beta \prec \beta'$ then $H(\alpha|\beta') \leq H(\alpha|\beta)$, where β' is a partition of the same set.*

5. $H(\alpha \vee \beta) = H(\alpha|\beta) + H(\beta)$.
6. $H(\alpha|\beta) \leq H(\alpha)$. If $H(\alpha) < \infty$, then $H(\alpha|\beta) = H(\alpha)$ if and only if α and β are independent.
7. $H(\alpha|\beta) = H(\alpha) - H(\beta)$ if and only if $\beta \prec \alpha$.

For proofs of these properties and further details on the entropy and the conditional entropy of partitions, see, for example, [10, 11].

Now let us define the *Rokhlin distance*. Let α and β , as defined above, be two partitions of the probability space (Ω, \mathcal{B}, P) with fixed probability measure p . The Rokhlin distance between partitions α and β is defined as follows [6]:

$$d(\alpha, \beta) = H(\alpha|\beta) + H(\beta|\alpha), \quad (4.3)$$

where $H(\cdot|\cdot)$ stands for the conditional entropy of partitions as given in Equation 4.2.

In the following considerations, we will apply the metric properties of the Rokhlin distance, which are formulated in the usual form [10, 11]:

1. $d(\alpha, \beta) \geq 0$, where $d(\alpha, \beta) = 0$ if and only if $\alpha = \beta$.
2. $d(\alpha, \beta) = d(\beta, \alpha)$.
3. $d(\alpha, \beta) \leq d(\alpha, \xi) + d(\xi, \beta)$ for any partition ξ of the same space.

Additional information regarding the Rokhlin distance $d(\alpha, \beta)$ within the framework of entropy theory can be found in [14]; for applications of this distance within the framework of ergodic theory, see [10, 11]. Note that the original definition of the Rokhlin distance deals with partitions of a general probability space (Ω, \mathcal{B}, P) . Below, we restrict the discussion by implementing the Rokhlin distance over a finite sample space $X = \{x_1, x_2, \dots, x_n\}$ with probability mass function $p : X \rightarrow [0, 1]$, $0 \leq p(x_i) \leq 1$, and $\sum_{i=1}^n p(x_i) = 1$. A σ -algebra \mathcal{B} in this case is specified by the above-mentioned power set $\mathcal{X} = 2^X$ of the set X , that is, a set \mathcal{X} that contains all possible subsets of X . Following these remarks, we will use interchangeably the definition of the probability space (Ω, \mathcal{B}, P) and the sample space X , on which a probability mass function p is defined.

Note that the distance in the form of the Rokhlin metric, especially for finite spaces, appears in several publications. For example, in 1991 López De Mántaras [15], independently of Rokhlin's work, presented a definition of the same metric for partitions of the finite set X and applied it for purposes of attribute selection in machine learning. In 1993, Lloris-Ruiz *et al.* [16] used a similar form of

$$d(\mathbf{x}, \mathbf{y}) = H(\mathbf{x}|\mathbf{y}) + H(\mathbf{y}|\mathbf{x})$$

for the informational distance between two random variables \mathbf{x} and \mathbf{y} , which take their values from the same finite set X . On the basis of this definition, in 2003 Jaroszewicz [17] considered the Rokhlin distance between partitions of a finite set X with equiprobable points and applied it to the problems of data mining. Independent research in data mining was conducted in 2004 by Peltonen [18], using the distance between partitions of a finite set. In Section 5.1 we continue these works by applying the search algorithm to the construction of efficient decision trees.

The relation between the Rokhlin metric and mutual information is straightforward. Recall (see Section 3.1.2) that the mutual information $I(\mathbf{x}; \mathbf{y})$ between two random variables \mathbf{x} and \mathbf{y} , taking their values from the set X , is defined as follows [19]:

$$I(\mathbf{x}; \mathbf{y}) = H(\mathbf{x}) - H(\mathbf{x}|\mathbf{y}) = H(\mathbf{y}) - H(\mathbf{y}|\mathbf{x}).$$

Note that the mutual information between two partitions α and β of the sample space X is as follows:

$$I(\alpha; \beta) = H(\alpha) - H(\alpha|\beta) = \sum_{A \in \alpha} \sum_{B \in \beta} p(A, B) \log \frac{p(A, B)}{p(A)p(B)}.$$

Accordingly, the relation between the mutual information $I(\alpha; \beta)$ and the Rokhlin metric $d(\alpha, \beta)$ is given by the following lemma.

Lemma 4.2. $d(\alpha, \beta) = H(\alpha, \beta) - I(\alpha; \beta)$.

Proof. Consider the partitions α and β as random variables. Then, for the mutual information, it follows that (see, e.g., [19])

$$I(\alpha; \beta) = H(\alpha) + H(\beta) - H(\alpha, \beta),$$

and for joint entropy it follows that (see, e.g., [10, 19])

$$H(\alpha, \beta) = H(\alpha) + H(\beta|\alpha) = H(\beta) + H(\alpha|\beta).$$

Then, one obtains that $I(\alpha; \beta) = H(\alpha, \beta) - H(\alpha|\beta) - H(\beta|\alpha)$, which gives

$$H(\beta|\alpha) = H(\alpha, \beta) - H(\alpha|\beta) - I(\alpha; \beta).$$

Substitution of $H(\beta|\alpha)$ into the definition of the Rokhlin distance (Equation 4.3) results in the following:

$$\begin{aligned} d(\alpha, \beta) &= H(\alpha|\beta) + H(\beta|\alpha) = H(\alpha|\beta) + H(\alpha, \beta) - H(\alpha|\beta) - I(\alpha; \beta) \\ &= H(\alpha, \beta) - I(\alpha; \beta) \end{aligned}$$

as required. ■

According to the lemma, the Rokhlin distance is determined by the difference between joint entropy and mutual information; that is, this metric gives the amount of information required to remove the remaining uncertainty on partitions α and β , once the mutual information between these partitions has been obtained.

Now let us consider the *Ornstein distance* between partitions. The Ornstein distance between the partitions α and β is defined as follows [7, 8]:

$$d_{\text{Orn}}(\alpha, \beta) = p(X \setminus (\cup_{j=1}^k (A_j \cap B_j))), \quad A_j \in \alpha, \quad B_j \in \beta,$$

where $k = \max\{|\alpha|, |\beta|\}$, and $|\cdot|$ stands for the cardinality of the partition. If $|\alpha| > |\beta|$, then the partition β is completed by empty sets, and if $|\beta| > |\alpha|$, then empty sets are added to the partition α .

Note that, similar to the Rokhlin distance, the Ornstein distance meets the metric requirements [8] and may be considered as a probabilistic version of the Hamming distance [14].

The relation between the Rokhlin distance and the Ornstein distance is given by the next lemma. In the implementation of these distances for the informational search algorithms, the lemma supports the admissibility property of the distance and its estimation, as required by the LRTA* and MTS algorithms (Algorithms 2.13 and 2.14).

Lemma 4.3 [21]. *If α and β are partitions of the same sample space X with probability mass function p , then considering the Ornstein metric $d_{Orn}(\alpha, \beta)$ and the Rokhlin metric $d(\alpha, \beta)$ it follows that $d_{Orn}(\alpha, \beta) \leq d(\alpha, \beta)$.*

Proof. Using the definition (Equation 4.2) of the conditional entropy and the formulas of relative probabilities, one obtains that

$$d(\alpha, \beta) = \sum_{i=1}^k \sum_{j=1}^k p(A_i \cap B_j) \left(\log \frac{p(A_i \cap B_j)}{p(A_i)} + \log \frac{p(A_i \cap B_j)}{p(B_i)} \right).$$

Since the probabilistic measure is additive, it is true that

$$d_{Orn}(\alpha, \beta) = 1 - \sum_{i=1}^k p(A_i \cap B_i).$$

Thus, one needs to prove the correctness of the following inequality:

$$R \equiv \sum_{i=1}^k \left(p(A_i \cap B_i) + \sum_{j=1}^k p(A_i \cap B_j) \left(\log \frac{p(A_i \cap B_j)}{p(A_i)} + \log \frac{p(A_i \cap B_j)}{p(B_i)} \right) \right) \geq 1. \quad (4.4)$$

If $\alpha = \beta$, then

$$\sum_{j=1}^k p(A_i \cap B_j) \left(\log \frac{p(A_i \cap B_j)}{p(A_i)} + \log \frac{p(A_i \cap B_j)}{p(B_i)} \right) = 0.$$

Thus, since α and β are partitions, $R = \sum_{i=1}^k p(A_i \cap B_i) = 1$ holds, and the inequality (Equation 4.4) is proven.

Assume that $\alpha \neq \beta$, and let $\alpha = \{A_1, \dots, A_{k-1}, A_k\}$ and $\beta = \{B_1, \dots, B_{k-1}, B_k\}$ be partitions such that $A_i = B_i$, for $i = 1, 2, \dots, k-2$, and $A_{k-1} \neq B_{k-1}$ and $A_k \neq B_k$.

Denote by $\alpha' = \{A'_1, \dots, A'_{k-1}, A'_k\}$ and $\beta' = \{B'_1, \dots, B'_{k-1}, B'_k\}$ partitions such that $A'_i = A_i$, $B'_i = B_i$ for $i = 1, 2, \dots, k-2$, and $A'_{k-1} = B'_{k-1}$ and $A'_k = B'_k$, while $p(A'_{k-1}) + p(A'_k) = p(A_{k-1}) + p(A_k)$ and $p(B'_{k-1}) + p(B'_k) = p(B_{k-1}) + p(B_k)$. Assume that $\alpha' = \beta'$, and denote by R' the left side of the inequality (4.4) for these partitions $\alpha' = \beta'$. As indicated above, in this case $R' = 1$. Since $\alpha' = \beta'$, it follows that

$$p(A'_{k-1}) + p(A'_k) = p(B'_{k-1}) + p(B'_k),$$

$$p(A_{k-1}) + p(A_k) = p(B_{k-1}) + p(B_k).$$

So either $A_{k-1} \subset B_{k-1}$ and $B_k \subset A_k$ or $B_{k-1} \subset A_{k-1}$ and $A_k \subset B_k$.

Let, for certainty, $A_{k-1} \subset B_{k-1}$ and $B_k \subset A_k$. Consider the last addenda of the right parts R' and R of the inequality (4.4). Denote by m' and m the corresponding sums up to $k-2$:

$$m' = \sum_{i=1}^{k-2} \left(p(A'_i \cap B'_i) + \sum_{j=1}^{k-2} p(A'_i \cap B'_j) \left(\log \frac{p(A'_i \cap B'_j)}{p(A'_i)} + \log \frac{p(A'_i \cap B'_j)}{p(B'_j)} \right) \right),$$

$$m = \sum_{i=1}^{k-2} \left(p(A_i \cap B_i) + \sum_{j=1}^{k-2} p(A_i \cap B_j) \left(\log \frac{p(A_i \cap B_j)}{p(A_i)} + \log \frac{p(A_i \cap B_j)}{p(B_i)} \right) \right).$$

Thus, $R' = m' + r'$ and $R = m + r$, where

$$r' = p(A'_{k-1}) + p(A'_k) = p(B'_{k-1}) + p(B'_k),$$

$$r = p(A_{k-1}) + p(A_{k-1}) \log \frac{p(B_{k-1})}{p(A_{k-1})} + p(B_k) + p(B_k) \log \frac{p(A_k)}{p(B_k)}.$$

Since $A_{k-1} \subset B_{k-1}$ and $B_k \subset A_k$, it follows that $p(A_{k-1}) \leq p(B_{k-1})$ and $p(A_k) \geq p(B_k)$. Thus, both

$$\log \frac{p(B_{k-1})}{p(A_{k-1})} \geq 0 \quad \text{and} \quad \log \frac{p(A_k)}{p(B_k)} \geq 0.$$

Finally, according to the definition of partitions α' and β' , one obtains that $m' = m$ and $r' = r$. Thus, $R' \leq R$.

The use of the same reasons for the remaining cases with $A'_i = A_i$, $B'_i = B_i$ when $i = k-2, k-3, \dots, 1$ is similar. Thus, if $\alpha = \beta$, then the left side of the inequality (4.4) is $R = 1$, and if $\alpha \neq \beta$, then $R \geq 1$ and the required inequality is proven. ■

As indicated above, Lemma 4.3 justifies the implementation of the Ornstein distance as an admissible estimated distance, while the actual distance is specified by the Rokhlin metric. Note that in addition to the Rokhlin and Ornstein metrics, the Hamming distance between partitions α and β can be measured. Such a metric is defined as the size of the symmetric difference between the partitions as follows [20]:

$$d_{Ham}(\alpha, \beta) = |\alpha \Delta \beta| = |(\alpha \setminus \beta) \cup (\beta \setminus \alpha)|,$$

and for this distance it follows that for any pair (α, β) of partitions of the sample space X , the Hamming distance is not less than the Rokhlin distance d , that is, $d(\alpha, \beta) \leq d_{Ham}(\alpha, \beta)$. For additional information on the Rokhlin and Ornstein distances and their various applications, see, for example, [14].

Finally, let us consider the Rokhlin and the Ornstein distance dependence on the location probabilities that are defined on the sample space X . In particular, let us first clarify the dependence of the Rokhlin distance on the location probabilities. Since the Rokhlin distance between the partitions meets the metric requirements, we also refer to it as the Rokhlin metric.

Denote by $\chi = \{\alpha | \alpha \text{ is a partition of } X\}$ the set of all possible partitions of the set X . Note that the number of elements of the partitions set χ is rather large, and it is greater

than the number of elements of the power set $\mathcal{X} = 2^X$, which includes all possible subsets of the set X . In particular, the number $N = |\chi|$ of partitions of the set X with cardinality $n = |X|$ is given by the Bell number (see, e.g., [20])

$$\Phi(n) = \sum_{k=0}^n \Phi(n, k), \quad (4.5)$$

where $\Phi(n, k) = \Phi(n - 1, k - 1) + k \times \Phi(n - 1, k)$ are Sterling numbers of type 2, which determine a number of partitions of the set with cardinality n into k subsets. For the Sterling numbers $\Phi(n, k)$, it is assumed that $\Phi(n, k) = 0$ if $k > n$, $\Phi(n, 0) = 0$ for $n > 0$, and $\Phi(0, 0) = 1$. In the recursive form, the Bell number is defined as follows [20]:

$$\Phi(n) = \sum_{k=0}^{n-1} \binom{n-1}{k} \Phi(k), \quad \Phi(0) = 1. \quad (4.6)$$

Given the set χ of partitions of the sample space X , let us implement the refinement relation ' \prec ' next. In general, this relation defines a partial order over the set χ , so that the pair (χ, \prec) is a lattice with the 'unit' element determined by the trivial partition $\theta = \{X, \emptyset\}$. In addition, denote by $\gamma = \{\{x_1\}, \{x_2\}, \dots, \{x_n\}\}$, $x_i \in X$, $i = 1, 2, \dots, n$, the discrete partition of the sample space X . The next lemma presents some properties of the Rokhlin distance over the lattice (χ, \prec) .

Lemma 4.4 [21, 22]. *Let (χ, \prec) be a lattice of partitions of the sample space X and let $\theta \in \chi$ and $\gamma \in \chi$ be trivial and discrete partitions, correspondingly. Then it follows that:*

1. $d(\theta, \gamma) = \max_{(\alpha, \beta) \in \chi \times \chi} d(\alpha, \beta).$
2. Either $d(\theta, \alpha) = d(\theta, \beta)$ or $d(\alpha, \gamma) = d(\beta, \gamma)$ implies $d(\alpha, \beta) = 0$.
3. Distance d is non-decreasing with refining.

Proof.

1. Assume, on the contrary, that there exist two partitions $\alpha, \beta \in \chi$ such that $d(\alpha, \beta) > d(\theta, \gamma)$, that is,

$$H(\alpha|\beta) + H(\beta|\alpha) > H(\theta|\gamma) + H(\gamma|\theta).$$

Note that $H(\theta) = 0$, and since $\theta \prec \gamma$, from Lemma 4.1 it follows that $H(\theta|\gamma) = 0$. Moreover, for any partition $\xi \in \chi$ it is satisfied that $\xi \prec \gamma$; thus $H(\gamma|\xi) = H(\gamma) - H(\xi)$. This implies that $H(\gamma) = H(\gamma|\xi) + H(\xi)$ and, in particular, $H(\gamma|\theta) = H(\gamma)$. Then, the considered inequality can be written as follows:

$$H(\alpha|\beta) + H(\beta|\alpha) > H(\gamma) - H(\theta) = H(\gamma) = H(\gamma|\beta) + H(\beta).$$

Hence, one obtains $H(\alpha|\beta) - H(\gamma|\beta) > H(\beta) - H(\beta|\alpha)$. However, $H(\beta) \geq H(\beta|\alpha)$, and since $\beta \prec \gamma$, it follows that $H(\gamma|\beta) \geq H(\alpha|\beta)$. So

$$H(\alpha|\beta) - H(\gamma|\beta) \leq 0 \quad \text{and} \quad H(\beta) - H(\beta|\alpha) \geq 0,$$

in contradiction with the assumption.

2. Since the Rokhlin distance meets the metric requirements, it follows that $d(\alpha, \alpha) = d(\alpha, \alpha) = 0$. Let us demonstrate that the equivalence $d(\theta, \alpha) = d(\theta, \beta)$ implies $d(\alpha, \beta) = 0$. Without loss of generality, assume that $\alpha \prec \beta$. Then, by definition and according to Lemma 4.1, $d(\theta, \alpha) = H(\alpha|\theta) + H(\theta|\alpha) = H(\alpha)$ and $d(\theta, \beta) = H(\beta|\theta) + H(\theta|\beta) = H(\beta)$. So $d(\theta, \alpha) = d(\theta, \beta)$ implies $H(\alpha) = H(\beta)$. In addition, $d(\alpha, \beta) = H(\alpha|\beta) + H(\beta|\alpha)$, and since $\alpha \prec \beta$, one obtains $H(\alpha|\beta) = 0$ and $H(\beta|\alpha) = H(\beta) - H(\alpha) = 0$, since $H(\alpha) = H(\beta)$. Thus $d(\alpha, \beta) = 0$.

For the distances from the discrete partition γ , the reasoning is similar. In fact, $d(\alpha, \gamma) = H(\gamma) - H(\alpha)$ and $d(\beta, \gamma) = H(\gamma) - H(\beta)$, so if $d(\alpha, \gamma) = d(\beta, \gamma)$, then $d(\alpha, \beta) = 0$.

3. Let $\alpha, \beta, \xi \in \chi$ be three partitions and assume that $\alpha \prec \beta \prec \xi$. Since, according to Lemma 4.1, $H(\alpha) \leq H(\xi) \leq H(\beta)$, it follows that $d(\theta, \alpha) \leq d(\theta, \xi) \leq d(\theta, \beta)$. Thus, distance d is non-decreasing with refining. ■

Denote the Rokhlin distance for the uniform distribution of target location (i.e., an equiprobable sample space X) by $d_u : \chi \times \chi \rightarrow [0, \infty)$ and the Rokhlin distance for some other probability mass function p by $d_p : \chi \times \chi \rightarrow [0, \infty)$. In addition, denote by H_u the entropy (and conditional entropy) which is defined for the equiprobable sample space, and by H_p the entropy (and conditional entropy) which is defined for another probability distribution over the sample space. Then the following lemma holds.

Lemma 4.5 [21, 22]. *If every partition $\alpha \in (\chi, \prec)$, $\alpha \neq \theta$, satisfies $H_p(\alpha) \neq H_u(\alpha)$, then for every pair (α, β) , $\alpha, \beta \in \chi$, it follows that $d_p(\alpha, \beta) \leq d_u(\alpha, \beta)$.*

Proof. Recall that the ‘unit’ element of the lattice (χ, \prec) is a trivial partition $\theta = \{X, \emptyset\}$, and denote by $\gamma = \{\{x_1\}, \{x_2\}, \dots, \{x_n\}\}$, $x_i \in X$, $i = 1, 2, \dots, n$, the discrete partition of the sample space X . From Lemma 4.1 it follows that for every probability mass function p that $d_p(\theta, \gamma) = H_p(\theta|\gamma) + H_p(\gamma|\theta) = H_p(\gamma|\theta) = H_p(\gamma)$. Since partition γ is a discrete partition, the entropy $H_p(\gamma)$ is equal to the entropy $H_p(X)$ of the sample space. Thus, the Jensen inequality (see, e.g., [19]) implies that $H_p(\gamma) \leq H_u(\gamma)$. Lemma 4.4 states that distance d is non-decreasing with refining. Moreover, since it is given that the probability mass function the entropy is concave [19], the distances d_p and d_u are also concave.

According to the condition of the lemma, for every $\alpha \in (\chi, \prec)$ it follows that

$$d_p(\theta, \alpha) \leq d_u(\theta, \alpha).$$

In fact, if, on the contrary, $d_p(\theta, \alpha) > d_u(\theta, \alpha)$, then, since d_p and d_u are concave, and $d_p(\alpha, \beta) \leq d_p(\theta, \gamma)$ and $d_u(\alpha, \beta) \leq d_u(\theta, \gamma)$, there exists a partition $\xi \in (\chi, \prec)$ such that $d_p(\theta, \xi) = d_u(\theta, \xi)$, but this is impossible since $H_p(\alpha) \neq H_u(\alpha)$ for any α .

Let $\alpha, \beta \in (\chi, \prec)$, and, without loss of generality, assume that $\alpha \prec \beta$. From the metric properties of d it follows that both $d_p(\alpha, \alpha) = 0$ and $d_u(\alpha, \alpha) = 0$. Under the lemma’s requirement, it follows that $d_p(\theta, \beta) \leq d_u(\theta, \beta)$. Thus, the distance d_u increases faster than the distance d_p . Hence, $d_p(\theta, \beta) - d_p(\theta, \alpha) \leq d_u(\theta, \beta) - d_u(\theta, \alpha)$.

Finally, recalling that if $\alpha \prec \beta$ then $d(\alpha, \beta) = H(\beta) - H(\alpha)$, and that $d(\theta, \beta) = H(\beta)$, $d(\theta, \alpha) = H(\alpha)$, we obtain the required inequality. ■

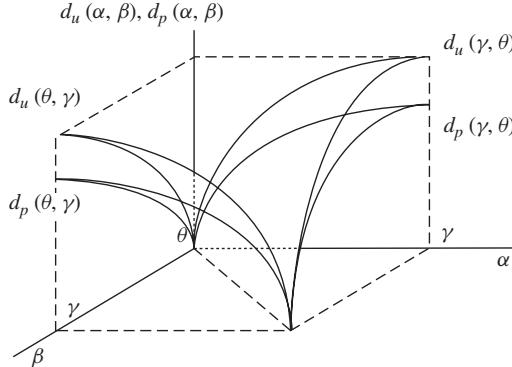


Figure 4.1 Dependence of the Rokhlin distances on location probabilities.

In other words, the lemma states that the Rokhlin distance depends on the probability distribution over a sample space, and for a uniform distribution the values of the Rokhlin distances are greater than those obtained by any other distribution. As seen below, such a property provides a basis for the probability updating scheme during the search, while maintaining the admissibility requirement of the distances [23].

The relation between the Rokhlin distances d_p and d_u is illustrated in Figure 4.1.

Note that Lemma 4.5 presents the relation between the probability distributions on the sample space X and the Rokhlin metric on the search space χ . This statement is also true for the steady state dynamics of the system, which includes both the target and the searcher.

The presented distances between partitions provide effective informational measures that can be applied for the search algorithms. Below, we implement informational distances in the algorithms of search for static and moving targets, which are based on the above A*-type algorithms.

4.2 Static target search: Informational LRTA* algorithm

In Section 2.3.2, we considered the methods of real-time search over graphs. In particular, the LRTA* algorithm (Algorithm 2.13), which was proposed by Korf [24–26], was outlined and its properties were presented. The LRTA* algorithm follows the line of A* algorithm (Algorithm 2.11) and, in particular, the RTA* algorithm (Algorithm 2.12), and implements online updating of the evaluated costs of the states.

In this section, we apply this algorithm for a search over the partitions space χ that includes the partitions of the sample space X , while the required evaluation functions are defined by using the Rokhlin distance above. Since this distance is defined using the entropies of partitions and specifies information which is required to remove the remaining uncertainty, while the possible information has already been obtained (see Lemma 4.2), the suggested algorithm is called the *Informational Learning Real-Time A** algorithm [21, 23, 27]. In a certain sense, this algorithm is related with the min–max Real-Time Search algorithm [28, 29] (Algorithm 3.1 in Section 3.4.3) that acts on non-deterministic domains.

4.2.1 Informational LRTA* algorithm and its properties

Let us start with the formulation of the ILRTA* algorithm. This algorithm is a special case of the general group-testing search procedure (Procedure 2.1) and is applied to search for a static target over a probabilistic space. The algorithm defines the method for selecting the next state similarly to the LRTA* algorithm, while the evaluated functions and the neighborhoods of the states are defined by the use of Rokhlin distances.

As above, let $X = \{x_1, x_2, \dots, x_n\}$ be a sample space with probability mass function $p : X \rightarrow [0, 1]$ that defines location probabilities $p(x_i)$, $i = 1, \dots, n$, $0 \leq p(x_i) \leq 1$, $\sum_{i=1}^n p(x_i) = 1$. Denote by χ the set of all possible partitions of X and assume that the Rokhlin metric (Equation 4.3) is defined over χ . The partitions set χ along with the Rokhlin metric is called the *partitions space*. Let us apply the LRTA* algorithm (Algorithm 2.13) to the partitions space χ .

Recall that the LRTA* algorithm acts on the finite graph $\mathcal{G} = (S, E)$ of states, where S is a set of states and $E \subset S \times S$ is a set of edges, and implements the *evaluation cost function* (Equation 2.50), which is defined as the sum

$$c_{RT}(s) = \bar{c}(a[s_{curr}, s]) + \tilde{c}(a^*[s, s_{fin}])$$

of the cost $\bar{c}(a[s_{curr}, s])$ of the path from the current searcher's state $s_{curr} \in S$ to the neighboring state $s \in N(s_{curr}) \subset S$ and the estimated cost $\tilde{c}(a^*[s, s_{fin}])$ of the optimal path from the state s to the final state $s_{fin} \in S$. In addition, recall that the actions of the LRTA* algorithm are based on the *admissibility assumption* (Equation 2.48) regarding all states $s \in S$

$$\tilde{c}(a^*[s, s_{fin}]) \leq \tilde{c}^*(a^*[s, s_{fin}]),$$

where $\tilde{c}^*(a^*[s, s_{fin}])$ stands for the actual cost of the optimal path from state s to the final state s_{fin} . Finally, recall that while considering the A* algorithm, we noted the *consistency assumption* (Equation 2.49), which for the real-time search has the form

$$\tilde{c}(a^*[s_{curr}, s_{fin}]) - \tilde{c}(a^*[s, s_{fin}]) \leq c_{RT}(a[s_{curr}, s]),$$

and represents a kind of triangle inequality regarding the estimated and evaluated costs of the states $s_{curr} \in S$ and $s \in N(s_{curr})$.

In order to apply the LRTA* algorithm over the partitions space χ , let us define the estimated and actual costs by using the Rokhlin metric $d : \chi \times \chi \rightarrow [0, \infty)$, which specifies the distances $d(\alpha, \beta)$ between the partitions $\alpha, \beta \in \chi$. We assume that for any pair (α, β) of partitions, the *actual cost* $\tilde{c}^*(a^*[\alpha, \beta])$ of the path from the partition α to the partition β is defined by the Rokhlin distance, that is,

$$d(\alpha, \beta) \equiv \tilde{c}^*(a^*[\alpha, \beta]).$$

Similar to the LRTA* algorithm, we say that $d(\alpha, \beta)$ specifies the *actual distance* between the partitions α and β .

Let $\gamma \in \chi$ be a final partition, which corresponds to the final state $s_{fin} \in S$ of the LRTA* algorithm where the target is located. The actual distance from each partition $\alpha \in \chi$ to the final partition γ that corresponds to the actual cost $\tilde{c}^*(a^*[s, s_{fin}])$ in the LRTA* algorithm is $d(\alpha, \gamma)$. Assume that over the partitions space χ there is another metric

$\tilde{d} : \chi \times \chi \rightarrow [0, \infty)$ defined such that, with respect to the Rokhlin metric d , it meets the *admissibility assumption*: that is, for any partition $\alpha \in \chi$ it follows that

$$\tilde{d}(\alpha, \gamma) \leq d(\alpha, \gamma). \quad (4.7)$$

In the light of Lemma 4.3, such a metric \tilde{d} can be considered as an Ornstein metric over χ . In general, we are not restricted in our use of the Rokhlin and Ornstein metrics and can apply any pair of metrics for which the admissibility (Equation 4.7) holds.

For any pair (α, β) of partitions $\alpha, \beta \in \chi$ the distance $\tilde{d}(\alpha, \beta)$, which is provided by the metric \tilde{d} , is associated with the *estimated cost* $\tilde{c}(a^*[\alpha, \beta])$ of the path from partition α to partition β , that is,

$$\tilde{d}(\alpha, \beta) \equiv \tilde{c}(a^*[\alpha, \beta]).$$

As above, we say that $\tilde{d}(\alpha, \beta)$ provides the *estimated distance* between partitions α and β . Note that without loss of generality we can assume that $\tilde{d}(\alpha, \gamma) = 0$ for every partition $\alpha \in \chi$, although such an assumption will prolong convergence of the algorithm.

In contrast to the LRTA* algorithm, where the neighbors of the state are defined as successors of the state in the graph on which the algorithm acts, the neighbors in the ILRTA* algorithm are defined by using the above distances. In particular, for any partition $\alpha \in \chi$, we say that the set $N(\alpha) \subset \chi$ is a *set of neighbors* of α if the set $N(\alpha)$ meets the following requirements:

- $\alpha \notin N(\alpha)$;
- for every partition $\beta \in N(\alpha)$ it follows that $\tilde{d}(\beta, \alpha) = d(\beta, \alpha)$.

The first requirement is used to avoid staying at the current partition. The second requirement specifies an ‘optimistic’ assumption which indicates that the searcher knows the exact distances at least to the neighboring partitions. This definition of neighborhood represents clear observation over the local area around the current state of the searcher, while observation of the remote states is uncertain. In the ILRTA*, if for some initial steps such assumptions are violated, then the searcher can assume that the distance to all partitions is less than the entropy of the sample space. Then, after few iterations of estimation updates, the above assumptions become valid.

Recall that without loss of generality we can assume that $\tilde{d}(\alpha, \gamma) = 0$. This distance estimation is considered ‘optimistic,’ since it represents an a priori assumption of the searcher that moving to the candidate partition will result in finding the target. Using such an estimation is conservative with respect to the search time; it prolongs convergence of the ILRTA* algorithm, since the distance estimations between any two partitions can be set to zero at the beginning of the search and are gradually refined during the search until they converge to their real distance values. The refinement of the distances is based on the fact that in the neighborhood of a partition, the distance estimates are accurate.

The neighborhood of states is a key concept in the stochastic local search algorithms and, thus, also in the ILRTA* algorithm: the distance estimation is always accurate between neighboring states. The neighbors in the suggested ILRTA* algorithm are defined by the proposed informational distances between the partitions. Practically, this means that when the searcher reaches a certain (current) partition, for example, $\alpha_1 = \{\{x_1, x_2\}, \{x_3, \dots, x_n\}\}$, there is a subset of neighborhood partitions that corresponds to a refinement of this partition, for example, $\alpha_2 = \{\{x_1\}, \{x_2\}, \{x_3, \dots, x_n\}\}$, with an accurate distance measure from the

current partition that can be reached by a certain group test. If the searcher elects to move to this partition (a vertex in the graph) a new subset of neighborhood partitions is available, for example, $\alpha_2 = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4, \dots, x_n\}\}$, and the process of learning continues until the target is found. Moreover, say that at a certain point in time the searcher is exposed to new (external) side information that changes the location probabilities (e.g., in the defective unit example the new information indicates that it is more reasonable to believe that the faulty unit is located toward the end of the batch); then this information is instantaneously absorbed in the new location probabilities (e.g., using a Bayesian update scheme) and transformed to new distance measures that are used by the searcher to select the next group test. Note that this notion of online local learning updates and convergence is very different from offline search procedures that assume that all the information has to be taken into account at the beginning of the search to define a priori all the search steps.

The consistency assumption in such a case obtains a simple form

$$\tilde{d}(\alpha, \gamma) - \tilde{d}(\beta, \gamma) \leq d(\alpha, \beta) + \tilde{d}(\beta, \gamma), \quad (4.8)$$

where $\beta \in N(\alpha)$, and holds true if $\tilde{d}(\alpha, \beta) \leq d(\alpha, \beta)$ for any $\alpha, \beta \in \chi$. If, for example, the distance estimation \tilde{d} is defined as an Ornstein metric, where the actual distance d is given by the Rokhlin metric, then the consistency holds.

The other possibility of defining the neighborhood $N(\alpha)$ is based on specification of the structure of the neighboring partitions constituted by $\tilde{d}(\beta, \alpha) = d(\beta, \alpha)$ for $\beta \in N(\alpha)$. The examples of such a definition will be considered below.

Now let us outline the ILRTA* algorithm, which is similar to the LRTA* algorithm (Algorithm 2.13).

Algorithm 4.1 (ILRTA* algorithm) [21, 23, 27]. Given the partitions space χ with admissible metrics \tilde{d} and d , initial partition $\theta \in \chi$, final partition $\gamma \in \chi$, and initial distance estimations $\tilde{d}_0(\alpha, \gamma)$, $\alpha \in \chi$, do:

1. Define $ResPath$ as a list of partitions.
2. Set $\alpha_{curr} = \theta$.
3. While $d(\alpha_{curr}, \gamma) \neq 0$ do:
 - 3.1. Obtain the neighborhood $N(\alpha_{curr})$ of the current partition α_{curr} : The neighborhood $N(\alpha_{curr})$ includes such partitions $\xi \in \chi$, except α_{curr} , that is, $\tilde{d}(\xi, \alpha) = d(\xi, \alpha)$.
 - 3.2. Set $\tilde{d}(\alpha_{curr}, \gamma) = \max\{\tilde{d}(\alpha_{curr}, \gamma), \min_{\alpha \in N(\alpha_{curr})}\{d(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \gamma)\}\}$.
 - 3.3. Set $\alpha_{next} = \operatorname{argmin}_{\alpha \in N(\alpha_{curr})}\{d(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \gamma)\}$; ties are broken randomly.
 - 3.4. $ResPath.include(\alpha_{curr})$.
 - 3.5. Set $\alpha_{curr} = \alpha_{next}$.
4. Return $ResPath$. ■

The ILRTA* algorithm (Algorithm 4.1) is a modification of the LRTA* algorithm (Algorithm 2.13), in which the estimated and actual costs are strictly defined by the estimated and actual informational distances, and the actions of the ILRTA* algorithm are similar to

those of the LRTA* algorithm. However, note that the ILRTA* algorithm does not require the states graph for obtaining the neighboring states and creates the neighborhood of each partition on the basis of estimated and real distances. Since the elements $\alpha \in \chi$ of the partitions space χ are partitions of the sample space X with defined location probabilities $p(x)$, $x \in X$, the ILRTA* algorithm, similar to the min–max RTA* algorithm [28, 29], can be implemented for search under uncertainty. If the location probabilities are equal, that is, the sample space X is equiprobable, then the ILRTA* algorithm can be reduced to the data-mining and classification algorithms recently presented by Jaroszewicz [17] and by Peltonen [18]. Below, we will consider the application of the ILRTA* algorithm for classification tasks for a sample space with arbitrary location probabilities.

Now let us show that for the ILRTA* algorithm (Algorithm 4.1) the basic properties of the LRTA* algorithm (Algorithm 2.13) hold. Below we provide independent proofs of these properties on the basis of the properties of distance measures that provides additional intuition to the application of the ILRTA* to classification problems [30, 31] and stresses the relation of the algorithm with the Markov decision process (MDP) models [32] and MTS problem [27]. Let us proceed with the main theorem of this section.

Theorem 4.1 [21]. (in parallel to Theorem 2.13). *If for every partition $\alpha \in \chi$ the admissibility assumption $\tilde{d}(\alpha, \gamma) \leq d(\alpha, \gamma)$ holds, then the trial of Algorithm 4.1 (ILRTA* algorithm) always terminates.*

Proof. The proof of the theorem is based on the next two claims.

Claim 1. *If final partition γ is regarded as a neighbor of the current partition α_{curr} , that is, $\gamma \in N(\alpha_{curr})$, then in the ILRTA* algorithm partition selection is $\alpha_{curr} \leftarrow \gamma$ and the estimation update is $\tilde{d}(\alpha_{curr}, \gamma) \leftarrow d(\alpha_{curr}, \gamma)$.*

Proof. Consider the triangle $(\alpha_{curr}, \alpha, \gamma)$ with the corresponding partitions α_{curr} , α , and γ in its vertices, and assume, on the contrary, that the algorithm chooses partition α . This implies that

$$d(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \gamma) \leq d(\alpha_{curr}, \gamma) + \tilde{d}(\gamma, \gamma) = d(\alpha_{curr}, \gamma).$$

Since in the triangle $\gamma \in N(\alpha)$ it follows that $\tilde{d}(\alpha, \gamma) = d(\alpha, \gamma)$, thus

$$d(\alpha_{curr}, \alpha) + d(\alpha, \gamma) \leq d(\alpha_{curr}, \gamma).$$

But, according to the triangle inequality,

$$d(\alpha_{curr}, \alpha) + d(\alpha, \gamma) \geq d(\alpha_{curr}, \gamma).$$

Hence, the only possible selection of α is $\alpha = \gamma$.

This claim guarantees that the algorithm selects the final partition while it is in the neighborhood of the current partition.

Claim 2. *Let the current partition be α_{curr} and the previous partition be α_{prev} . If there exists a partition $\alpha \in N(\alpha_{curr})$, $\alpha \neq \alpha_{prev}$, then at the next step, the ILRTA* algorithm will select α . In other words, the ILRTA* in the next step does not return to the previous partition.*

Proof. Consider the step when partition α_{curr} is chosen. For partition α_{curr} , according to the triangle inequality and the admissibility assumption, it follows that

$$\tilde{d}(\alpha_{prev}, \gamma) \leq d(\alpha_{prev}, \alpha_{curr}) + \tilde{d}(\alpha_{curr}, \gamma).$$

Thus, the estimation updating in the ILRTA* algorithm is as follows:

$$\tilde{d}(\alpha_{prev}, \gamma) \leftarrow d(\alpha_{prev}, \alpha_{curr}) + \tilde{d}(\alpha_{curr}, \gamma).$$

Let $\alpha \in N(\alpha_{curr})$, and assume, on the contrary, that the next partition chosen after α_{curr} is $\alpha_{prev} \in N(\alpha_{curr})$. Then, it follows that

$$2d(\alpha_{prev}, \alpha_{curr}) + \tilde{d}(\alpha_{curr}, \gamma) + \tilde{d}(\alpha_{prev}, \gamma) \leq d(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \gamma).$$

If $\alpha \in N(\alpha_{curr})$ and $\alpha_{prev} \in N(\alpha_{curr})$, then $d(\alpha_{prev}, \alpha_{curr}) = \tilde{d}(\alpha_{prev}, \alpha_{curr})$ and $d(\alpha_{curr}, \alpha) = \tilde{d}(\alpha_{curr}, \alpha)$. Hence, the last inequality leads to

$$\tilde{d}(\alpha_{prev}, \alpha_{curr}) + \tilde{d}(\alpha_{curr}, \gamma) + \tilde{d}(\alpha_{prev}, \alpha_{curr}) + \tilde{d}(\alpha_{prev}, \gamma) \leq \tilde{d}(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \gamma).$$

However, since in the previous step the partition α_{curr} was chosen, it follows that

$$\tilde{d}(\alpha_{prev}, \alpha_{curr}) + \tilde{d}(\alpha_{curr}, \gamma) \leq \tilde{d}(\alpha_{prev}, \alpha) + \tilde{d}(\alpha, \gamma).$$

Thus, the previous inequality requires $\tilde{d}(\alpha_{prev}, \alpha_{curr}) = 0$ and $\tilde{d}(\alpha_{prev}, \gamma) = 0$, which implies both $\alpha_{prev} = \alpha_{curr}$ and $\alpha_{prev} = \gamma$, which is impossible. Therefore, the chosen partition has to be α and the algorithm does not return to the previous partition α_{prev} .

Now let us prove the theorem. Let α be a partition such that $\gamma \in N(\alpha)$. If the current partition is $\alpha_{curr} = \alpha$, then according to Claim 1 the final partition γ is chosen and the algorithm terminates.

Let us consider the previous step and let $\alpha_{prev} \neq \alpha$ be a partition which was chosen at this step and $\gamma \notin N(\alpha_{prev})$. Then, $d(\alpha_{curr}, \alpha_{prev}) + \tilde{d}(\alpha_{prev}, \gamma) \leq \tilde{d}(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \gamma)$.

If $\alpha \in N(\alpha_{prev})$, then using the metric properties one obtains $d(\alpha_{curr}, \alpha_{prev}) + d(\alpha, \gamma) + \tilde{d}(\alpha_{prev}, \gamma) \geq \tilde{d}(\alpha_{curr}, \alpha_{prev}) + \tilde{d}(\alpha_{prev}, \gamma)$, and since $\tilde{d}(\alpha, \gamma) = d(\alpha, \gamma)$ when $\gamma \in N(\alpha)$, this leads to the equivalence $\alpha = \alpha_{prev}$.

Let, in contrast, $\alpha \notin N(\alpha_{prev})$. If $\alpha_{curr} \in N(\alpha_{prev})$ is a single partition in the neighborhood of α_{prev} , then the next partition after α_{prev} is α_{curr} . Then, according to Claim 2, the algorithm continues without returning to α_{prev} and selects the next partition from $N(\alpha_{curr})$. Assume that there exists a partition $\beta \in N(\alpha_{prev})$, $\beta \neq \alpha_{curr}$. Thus, according to Claim 2, the algorithm chooses partition β . If $\gamma \in N(\beta)$, then, according to Claim 1, the algorithm chooses the final partition γ and terminates. If $\gamma \notin N(\beta)$, then the same reasoning is applied to partition $\alpha_{curr} = \beta$.

By applying a similar consideration recursively to the steps before the current selection, and so on, up to the initial partition θ , one obtains the statement of the theorem. ■

The next lemma guarantees that the ILRTA* algorithm (Algorithm 4.1) preserves an admissibility property of the distances and distance estimation.

Lemma 4.6 [21]. (in parallel to Lemma 2.12). *Throughout the trial of Algorithm 4.1 (ILRTA* algorithm) and at its termination, for all partitions $\alpha \in \chi$ the admissibility $\tilde{d}(\alpha, \gamma) \leq d(\alpha, \gamma)$ is preserved.*

Proof. If $\tilde{d}(\alpha_{curr}, \gamma) \geq \min_{\alpha \in N(\alpha_{curr})} \{d(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \gamma)\}$, then the estimation updating does not change the estimated distance $\tilde{d}(\alpha_{curr}, \gamma)$ and the admissibility is preserved.

Let $\tilde{d}(\alpha_{curr}, \gamma) < \min_{\alpha \in N(\alpha_{curr})} \{d(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \gamma)\}$ and denote

$$\alpha_{\min} = \operatorname{argmin}_{\alpha \in N(\alpha_{curr})} \{d(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \gamma)\}.$$

We need to show that $d(\alpha_{curr}, \alpha_{\min}) + \tilde{d}(\alpha_{\min}, \gamma) \leq d(\alpha_{curr}, \gamma)$.

Assume, on the contrary, that $d(\alpha_{curr}, \alpha_{\min}) + \tilde{d}(\alpha_{\min}, \gamma) > d(\alpha_{curr}, \gamma)$. Then, using the triangle inequality we obtain

$$d(\alpha_{curr}, \alpha_{\min}) + \tilde{d}(\alpha_{\min}, \gamma) > d(\alpha_{curr}, \gamma) \leq d(\alpha_{curr}, \alpha_{\min}) + d(\alpha_{\min}, \gamma).$$

Since α_{\min} is a partition that supplies a minimal distance estimation, the segment $(\alpha_{curr}, \alpha_{\min})$ is in the path, which gives the exact distance value $d(\alpha_{curr}, \gamma)$. Therefore,

$$d(\alpha_{curr}, \gamma) = d(\alpha_{curr}, \alpha_{\min}) + d(\alpha_{\min}, \gamma) \quad \text{and} \quad \tilde{d}(\alpha_{\min}, \gamma) > d(\alpha_{\min}, \gamma),$$

which contradicts the admissibility $\tilde{d}(\alpha_{\min}, \gamma) \leq d(\alpha_{\min}, \gamma)$ of the non-updated distances. ■

Finally, let us show that the suggested ILRTA* algorithm (Algorithm 4.1) obtains an optimal solution in the sense of informational distances between partitions of the partitions space χ .

Theorem 4.2 [21]. (in parallel to Theorem 2.14). *Algorithm 4.1 (ILRTA* algorithm) converges with trials to the sequence of partitions such that the difference between the sum of the distances between its partitions and the actual distance $d(\theta, \gamma)$ reaches its minimum and $\tilde{d}(\alpha, \gamma) = d(\alpha, \gamma)$ for all partitions $\alpha \in \chi$.*

Proof. The statement of the theorem is an immediate consequence of Theorem 4.1 and Lemma 4.6. In fact, if for the partitions $\alpha \in \chi$, which are included in the sequence at the i th trial, it follows that $\tilde{d}_0^i(\alpha, \gamma) < \tilde{d}^i(\alpha, \gamma) \leq d(\alpha, \gamma)$, then for $(i+1)$ th trial these partitions have initial estimations $\tilde{d}_0^{i+1}(\alpha, \gamma) = \tilde{d}^{i+1}(\alpha, \gamma) \leq d(\alpha, \gamma)$. So, for each path chosen on the next trial j we have $\tilde{d}_0^{i+j}(\alpha, \gamma) \rightarrow d(\alpha, \gamma)$ while $j \rightarrow \infty$ for all partitions $\alpha \in \chi$ included in the path chosen in the $(i+j)$ th trial. Since the number of paths is finite and the distance estimations are strictly increasing with the trials, these estimations will converge to their upper bound, which is provided by real distances according to the admissibility requirement (Equation 4.7). Thus, after a finite number of trials, all distance estimations will be equal to the real distances, and the obtained path will follow such distances, as required. ■

Note that since the estimated and actual distances meet the assumptions regarding the corresponding costs as they are used in the LRTA* algorithm, the properties of the

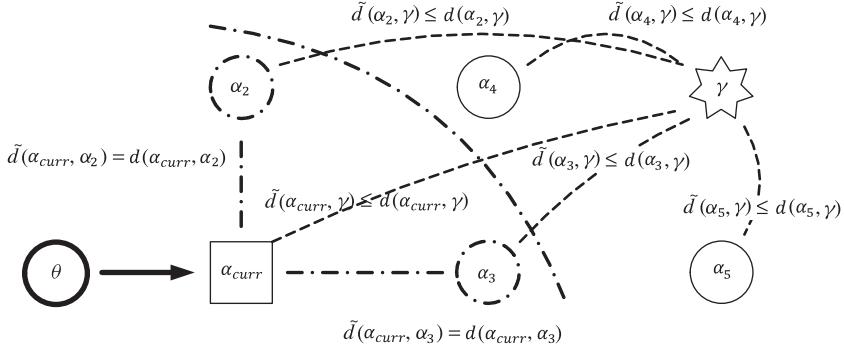


Figure 4.2 Actual and estimated distances and the neighborhood of the partition in the ILRTA* algorithm.

LRTA* algorithm can be implemented straightforwardly. Nevertheless, the determination of different probability measures on the sample space X allows an application of the suggested ILRTA* algorithm for various tasks that are different from the search problem.

The actions of the ILRTA* algorithm (Algorithm 4.1) are similar to those of the LRTA* algorithm (Algorithm 2.13) and will be illustrated by several examples below. The example of general relations between the estimated and actual distances and corresponding neighborhoods in the partitions space $\chi = \{\theta = \alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6 = \gamma\}$, which includes seven partitions, is shown in Figure 4.2.

In the figure, the algorithm starts with the initial partition $\theta = \alpha_0$ and in the current step the considered partition is $\alpha_{curr} = \alpha_1$. It is assumed that the neighborhood $N(\alpha_{curr})$ of the current partition α_{curr} includes two following partitions $N(\alpha_{curr}) = \{\alpha_2, \alpha_3\}$. Then for such partitions the estimated and actual distances are equal, that is, $d(\alpha_{curr}, \alpha_2) = d(\alpha_{curr}, \alpha_2)$ and $d(\alpha_{curr}, \alpha_3) = d(\alpha_{curr}, \alpha_3)$, while for the other partitions the admissibility assumption holds.

The ILRTA* algorithm (Algorithm 4.1) acts over the partitions space and has the same properties as the LRTA* algorithm. In particular, it is demonstrated that it converges with trials to the optimal solution. In addition, the suggested approach allows a direct comparison between the suggested ILRTA* algorithm and known information theory algorithms of search for a static target. Such a comparison provides the basis for a unified approach of search based on partitions of the sample space. In the next section, we address the actions of the algorithm under conditions that result in an optimal or near-optimal solution in one trial and compare the algorithm with the group-testing search algorithms which were considered in Section 2.2.4.

4.2.2 Group-testing search using the ILRTA* algorithm

Let us consider the relation between the ILRTA* algorithm (Algorithm 4.1) and the group-testing search algorithms with the Huffman–Zimmerman search (Algorithm 2.9) and with the GOTA (Algorithm 2.10). The first algorithm presents the optimal bottom-up search, while the second follows the top-down near-optimal scenario. In both cases, we consider the solution obtained by a single trial of the ILRTA* algorithm and show that

under suitable conditions it gives optimal and near-optimal solutions, as obtained by the Huffman–Zimmerman procedure and by the GOTA.

Recall (see Section 2.2.4) that, according to Theorem 2.7, which defines the construction of the Huffman search tree $\mathcal{T}[X]$, the tree is constructed by the *bottom-up* scenario from the leaves to the root level. Since it is assumed that the search tree $\mathcal{T}[X]$ is binary and the observations are certain, the tree $\mathcal{T}[X]$ defines a set partition system (see Section 2.1.1): that is, the subsets that correspond to the vertices of each level of the tree are disjoint, while their union results in the complete sample space X . Correspondence between the search tree $\mathcal{T}[X]$ and partitions is illustrated by Example 2.17 for the Huffman search tree and Examples 2.18 and 2.19 for the GOTA search tree.

We start by considering the relation between the ILRTA* algorithm and the Huffman–Zimmerman search. Let $X = \{x_1, x_2, \dots, x_n\}$ be a sample space with the probabilities $p_i = p(x_i) \in [0, 1]$, $i = 1, \dots, n$, $\sum_{i=1}^n p(x_i) = 1$, and let χ be a partitions set over X . Assume, in addition, that the Rokhlin distance (Equation 4.3) over χ and the refinement relation ‘ \prec ’ are defined. Given a Huffman binary search tree $\mathcal{T}[X]$ over the sample space X , let us enumerate its levels so that $j = 0$ for the root level and $j = l$ for the leaves level.

Let α be a partition that corresponds to some level of the Huffman binary search tree $\mathcal{T}[X]$, and denote by

$$A'_{\min} = \operatorname{argmin}_{A \in \alpha} \{p(A)\} \quad \text{and} \quad A''_{\min} = \operatorname{argmin}_{A \in (\alpha \setminus \{A'_{\min}\})} \{p(A)\}$$

the first and the second sets from the partition α , for which the probabilities $p(A) = \sum_{x \in A} p(x)$ are minimal. Let α_l be a partition corresponding to the leaves of the Huffman binary search tree $\mathcal{T}[X]$, where l stands for the level of the deepest branch in the tree $\mathcal{T}[X]$. Then, according to the Huffman–Zimmerman procedure (see Theorem 2.7), partition α_{l-j-1} , which corresponds to the $(l - j - 1)$ th level of the tree, $j = 0, 1, \dots, l - 1$, is as follows:

$$\alpha_{l-j-1} = \{A'_{\min} \cup A''_{\min}, A_k | A_k \in \alpha_{l-j}, A_k \neq A'_{\min}, A_k \neq A''_{\min}, k = 1, 2, \dots, |\alpha_{l-j}| - 1\}.$$

Accordingly, we say that partitions α and β are *Huffman neighbors*, if one is a refinement of the other and their size differs by 1; that is, partitions $\alpha = \{A | A \subset X\}$ and $\beta = \{B | B \subset X\}$ are Huffman neighbors if either

$$\alpha = \{B_i \cup B_j, B_1, B_2, \dots, B_{i-1}, B_{i+1}, \dots, B_{j-1}, B_{j+1}, \dots, B_{|\beta|} | B_i \in \beta, i = 1, 2, \dots, |\beta|\},$$

or

$$\beta = \{A_i \cup A_j, A_1, A_2, \dots, A_{i-1}, A_{i+1}, \dots, A_{j-1}, A_{j+1}, \dots, A_{|\alpha|} | A_i \in \alpha, i = 1, 2, \dots, |\alpha|\}.$$

The set of Huffman neighbors is denoted by $N_{\text{Huf}}(\cdot)$. Thus if $\alpha \in N_{\text{Huf}}(\beta)$ then $\alpha \prec \beta$ and $|\alpha| = |\beta| - 1$, and if $\beta \in N_{\text{Huf}}(\alpha)$ then $\beta \prec \alpha$ and $|\beta| = |\alpha| - 1$. Note that by specifying the size of Huffman members we implement the assumption that the partition is not a neighbor of itself, that is, $\alpha \notin N_{\text{Huf}}(\alpha)$. As indicated above regarding the estimated and actual costs of Huffman neighbors, we assume that if $\alpha \in N_{\text{Huf}}(\beta)$ then $\tilde{d}(\alpha, \beta) = d(\alpha, \beta)$. Later we will show that such an assumption is reasonable since it reflects a simple entropy

measure of A'_{\min} and A''_{\min} , which is the required knowledge for constructing the Huffman tree at each stage.

Let α_l be a partition corresponding to the leaves of the Huffman binary search tree $T[X]$, and recall that $H(\alpha|\beta)$ stands for the conditional entropy of partition α given partition β , as defined by Equation 4.2.

Lemma 4.7 [21]. *According to the Huffman–Zimmerman procedure (Theorem 2.7), given α_l and $\alpha_{curr} \prec \alpha_l$ the next partition α_{next} among all Huffman neighbors $\alpha \in N_{Huf}(\alpha_{curr})$ of the current partition α_{curr} is chosen as follows:*

$$\alpha_{next} \leftarrow \operatorname{argmax}_{\alpha \in N_{Huf}(\alpha_{curr})} \{H(\alpha_l|\alpha_{curr}) - H(\alpha_l|\alpha)\}.$$

Proof. Let $\alpha_{curr} = \{A_1, A_2, \dots, A_m\}$. Denote $p_i = p(A_i)$, $i = 1, 2, \dots, m$, and assume that $p_1 \leq p_2 \leq \dots \leq p_m$. It is clear that such ordering always exists and does not change the form of the Huffman tree.

Then, according to the Huffman–Zimmerman procedure, the next partition α_{next} is $\alpha_{next} = \{A' = A_1 \cup A_2, A_3, \dots, A_m\}$ with $p' = p_1 + p_2$ and we should prove that

$$\alpha_{next} = \{A_1 \cup A_2, A_3, \dots, A_m\} = \operatorname{argmax}_{\alpha \in N_{Huf}(\alpha_{curr})} \{H(\alpha_l|\alpha_{curr}) - H(\alpha_l|\alpha)\}.$$

Since $\alpha_{curr} \prec \alpha_l$ and $\alpha_{next} \prec \alpha_l$, according to Lemma 4.6, for the conditional entropies $H(\alpha_l|\alpha_{curr})$ and $H(\alpha_l|\alpha_{next})$ it follows that $H(\alpha_l|\alpha_{curr}) = H(\alpha_l) - H(\alpha_{curr})$ and $H(\alpha_l|\alpha_{next}) = H(\alpha_l) - H(\alpha_{next})$. Thus, we need to demonstrate that

$$H(p_1 + p_2, p_3, \dots, p_m) \geq H(p_1 + p_i, p_2, \dots, p_{i-1}, p_{i+1}, \dots, p_m). \quad (4.9)$$

If $m = 2$, then the inequality (4.9) is an obvious equality:

$$H(p_1 + p_2, 0) = H(p_1 + p_i, 0).$$

Let $m > 2$. Applying the full form of the entropy, we obtain the inequality (4.9) in the following form:

$$-(p_1 + p_2) \log(p_1 + p_2) - \sum_{j=3}^m p_j \log p_j \geq -(p_1 + p_i) \log(p_1 + p_i) - \sum_{j=2, j \neq i}^m p_j \log p_j,$$

and finally, with the change of sign,

$$(p_1 + p_2) \log(p_1 + p_2) + p_i \log p_i \leq (p_1 + p_i) \log(p_1 + p_i) + p_2 \log p_2.$$

Recall that $p_1 \leq p_2 \leq p_i$ and $p_1 + p_2 + p_i \leq 1$. Let us fix the probabilities p_1 and p_i , and consider the function

$$f(p_2) = (p_1 + p_2) \log(p_1 + p_2) + p_i \log p_i - (p_1 + p_i) \log(p_1 + p_i) - p_2 \log p_2.$$

We need to show that $f(p_2) \leq 0$ for $p_1 \leq p_2 \leq p_i$.

The first derivative df/dp_2 of the function f is

$$\frac{df}{dp_2} = \log(p_1 + p_2) - \log p_2 = \log \frac{p_1 + p_2}{p_2}$$

and the second derivative $d^2 f/dp_2^2$ of the function f is

$$\frac{d^2 f}{dp_2^2} = \frac{1}{\ln 2} \left(\frac{1}{p_1 + p_2} - \frac{1}{p_2} \right).$$

Since $p_1 \leq p_2 \leq p_i$, the first derivative df/dp_2 is non-negative and the second derivative $d^2 f/dp_2^2$ is negative. Hence, the function f increases with p_2 up to the value

$$f(p_i) = (p_1 + p_i) \log(p_1 + p_i) + p_i \log p_i - (p_1 + p_i) \log(p_1 + p_i) - p_i \log p_i = 0,$$

which is a maximum of the function f .

Thus, given current partition $\alpha_{curr} = \{A_1, A_2, \dots, A_m\}$ such that $p(A_1) \leq p(A_2) \leq \dots \leq p(A_m)$, the maximal value of the difference $H(\alpha_l|\alpha_{curr}) - H(\alpha_l|\alpha)$ is reached for the next partition $\alpha_{next} = \{A' = A_1 \cup A_2, A_3, \dots, A_m\}$. ■

This lemma justifies the following formalization of the Huffman–Zimmerman procedure (see Theorem 2.7) over the space χ of partitions of the sample space X . Let $\mathcal{T}[X]$ be a Huffman binary search tree and let partition α_0 correspond to the root level of the tree $\mathcal{T}[X]$ and partition α_l correspond to its leaves level. Then, the Huffman–Zimmerman procedure over the partitions space χ is outlined as follows.

Algorithm 4.2 (Huffman–Zimmerman search over partitions space) [21, 23]. Given the partitions space χ , root-level partition $\alpha_0 \in \chi$, and leaves-level partition $\alpha_l \in \chi$ do:

1. Set $\alpha_{curr} = \alpha_l$.
2. While $H(\alpha_{curr}|\alpha_0) \neq 0$ do:
 - 2.1. Obtain the Huffman neighborhood $N_{Huf}(\alpha_{curr})$ of the current partition α_{curr} .
 - 2.2. Set $\alpha_{next} = \operatorname{argmax}_{\alpha \in N_{Huf}(\alpha_{curr})} \{H(\alpha_l|\alpha_{curr}) - H(\alpha_l|\alpha)\}$; ties are broken randomly. Since the next partition is chosen from among the Huffman neighbors of the current partition, it is, according to Lemma 4.1, equivalent to the choice: $\alpha_{next} = \operatorname{argmax}_{\alpha \in N_{Huf}(\alpha_{curr})} \{H(\alpha)\}$.
 - 2.3. Set $\chi = \chi \setminus \{\alpha_{next}\}$.
 - 2.4. Set $\alpha_{curr} = \alpha_{next}$.
3. Return. ■

Let us demonstrate the relation between the ILRTA* algorithm (Algorithm 4.2) and the Huffman–Zimmerman procedure (Algorithm 4.2). Let $\alpha, \beta \in \chi$ be two partitions of the sample space X . As above, assume that the actual distance between the partitions α and β is defined by the Rokhlin metric $d(\alpha, \beta)$ specified by Equation 4.3, and that estimated distance $\tilde{d}(\alpha, \beta)$ is specified by some other metric such that the admissibility assumption (Equation 4.7) holds. Recall that θ stands for an initial partition and γ stands for a final partition.

Theorem 4.3 [21]. Assume that for the partitions α and β that are not Huffman neighbors the estimated distance is zero, that is, $\tilde{d}(\alpha, \beta) = 0$ if $\alpha \notin N_{\text{Huf}}(\beta)$ and $\beta \notin N_{\text{Huf}}(\alpha)$. Then, if $\gamma \prec \theta$, the ILRTA* algorithm (Algorithm 4.2) applied to the Huffman neighborhoods creates a search tree which is equivalent to the Huffman search tree created by the Huffman–Zimmerman procedure (Algorithm 4.2).

Proof. Consider the Huffman–Zimmerman procedure (Algorithm 4.2). The procedure starts from the partition $\theta = \alpha_l$ and, according to Lemma 4.7, the action choice is based on the difference between the conditional entropies

$$H(\theta|\alpha_{curr}) - H(\theta|\alpha), \quad \alpha \prec \alpha_{curr} \prec \theta,$$

among all partitions $\alpha \in \chi$. Hence, for the chosen partitions $\alpha_{l-i-1} \prec \alpha_l$, $i = 0, 1, \dots, l-1$, by the Huffman procedure it follows that $\theta = \alpha_l \succ \alpha_{l-1} \succ \dots \succ \alpha_1 \succ \alpha_0 = \gamma$.

According to the properties of the conditional entropy (see Lemma 4.6), which, in particular, specify that if $\beta \succ \alpha$ then $H(\beta|\alpha) = H(\beta) - H(\alpha)$, for every partition $\alpha \in \chi$ one obtains

$$H(\theta|\alpha_{curr}) - H(\theta|\alpha) = H(\theta) - H(\alpha_{curr}) - H(\theta) + H(\alpha) = H(\alpha) - H(\alpha_{curr}).$$

Thus, given the initial partition θ , the selection

$$\alpha_{next} \leftarrow \operatorname{argmax}_{\alpha \in N_{\text{Huf}}(\alpha_{curr})} \{H(\theta|\alpha_{curr}) - H(\theta|\alpha)\},$$

which is used in Line 2.2 of the Huffman–Zimmerman procedure (Algorithm 4.2), has the following form:

$$\alpha_{next} \leftarrow \operatorname{argmax}_{\alpha \in N_{\text{Huf}}(\alpha_{curr})} \{H(\alpha) - H(\alpha_{curr})\},$$

which is equivalent to the selection

$$\alpha_{next} \leftarrow \operatorname{argmin}_{\alpha \in N_{\text{Huf}}(\alpha_{curr})} \{H(\alpha_{curr}) - H(\alpha)\}.$$

Now consider the action choice in Line 3.3 of the ILRTA* algorithm (Algorithm 4.2)

$$\alpha_{next} \leftarrow \operatorname{argmin}_{\alpha \in N(\alpha_{curr})} \{d(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \gamma)\}.$$

Following the assumptions of the theorem and taking into account that the Huffman neighborhood is applied, for such action choice one obtains

$$\alpha_{next} \leftarrow \operatorname{argmin}_{\alpha \in N_{\text{Huf}}(\alpha_{curr})} \{H(\alpha|\alpha_{curr}) + H(\alpha_{curr}|\alpha)\},$$

with $H(\alpha|\alpha_{curr}) = 0$ and $H(\alpha_{curr}|\alpha) = H(\alpha_{curr}) - H(\alpha)$. Hence, the choice is

$$\alpha_{next} \leftarrow \operatorname{argmin}_{\alpha \in N_{\text{Huf}}(\alpha_{curr})} \{H(\alpha_{curr}) - H(\alpha)\},$$

which is equivalent to the above-obtained form of the next partition choice in the Huffman–Zimmerman procedure.

Now consider the estimation update in Line 3.2 of the ILRTA* algorithm (Algorithm 4.2). According to Claim 2 in the proof of Theorem 4.3, the goal of the estimation update is to avoid a return of the algorithm to the previous partition in the same trial. This is

equivalent to a deletion of this partition from the set of all available partitions – a rule that is executed by Line 2.3 in the Huffman–Zimmerman procedure (Algorithm 4.2).

Finally, we need to show that the termination condition $d(\alpha_{curr}, \gamma) = 0$, which is used in the ILRTA* algorithm (Algorithm 4.2), is equivalent to the termination condition $H(\alpha_{curr}|\alpha_0) = 0$ of the Huffman–Zimmerman procedure (Algorithm 4.2). For the distance between the partitions, which is implemented in the ILRTA* algorithm, it follows that

$$d(\alpha_0, \alpha_{curr}) = H(\alpha_0|\alpha_{curr}) + H(\alpha_{curr}|\alpha_0).$$

Since $\alpha_0 < \alpha_{curr}$, it follows that $H(\alpha_0|\alpha_{curr}) = 0$. Hence, $d(\alpha_0, \alpha_{curr}) = H(\alpha_{curr}|\alpha_0)$, which is the termination condition of the Huffman–Zimmerman procedure. ■

As a consequence of Theorem 4.3 we obtain the following. The Huffman–Zimmerman procedure is considered as optimal in the sense of minimal average number of search actions. Theorem 4.3 states that the ILRTA* algorithm, which implements the same conditions, creates the same search tree as the Huffman–Zimmerman procedure. Thus, under the conditions of this procedure the suggested ILRTA* algorithm is optimal in the same sense of minimal average number of search actions.

The relation between the ILRTA* algorithm and the Huffman–Zimmerman procedure is illustrated by the following example.

Example 4.1 [21]. Let $X = \{x_1, x_2, x_3\}$ be a sample space with the location probabilities $p(x_1) = 0.1$, $p(x_2) = 0.3$, and $p(x_3) = 0.6$. Partitions space χ of the sample space X consists of the following five partitions:

$$\begin{aligned}\alpha_0 &= \{\{x_1\}, \{x_2\}, \{x_3\}\}, \\ \alpha_1 &= \{\{x_1\}, \{x_2, x_3\}\}, \\ \alpha_2 &= \{\{x_2\}, \{x_1, x_3\}\}, \\ \alpha_3 &= \{\{x_3\}, \{x_1, x_2\}\}, \\ \alpha_4 &= \{\{x_1, x_2, x_3\}, \emptyset\}.\end{aligned}$$

Recall that in the Huffman–Zimmerman procedure the search tree is created from bottom, which corresponds to the leaves level, to top, which corresponds to the root level. Thus, for this procedure an initial partition θ and a final partition γ are as follows:

$$\theta = \alpha_0 = \{\{x_1\}, \{x_2\}, \{x_3\}\} \quad \text{and} \quad \gamma = \alpha_4 = \{\{x_1, x_2, x_3\}, \emptyset\}.$$

Given the above location probabilities, the Huffman search tree for the sample space $X = \{x_1, x_2, x_3\}$ and the partitions, which correspond to the levels of the tree, are shown in Figure 4.3a. The procedure starts with the partition $\theta = \alpha_0 = \{\{x_1\}, \{x_2\}, \{x_3\}\}$, after which it chooses partition $\alpha_3 = \{\{x_3\}, \{x_1, x_2\}\}$, and then, finally, it terminates with the final partition $\alpha_4 = \{\{x_1, x_2, x_3\}, \emptyset\}$.

Let us consider the actions of the ILRTA* algorithm (Algorithm 4.2) on the partitions space $\chi = \{\theta = \alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4 = \gamma\}$, which includes the same five partitions. The ILRTA* algorithm starts from the initial partition $\theta = \alpha_0 = \{\{x_1\}, \{x_2\}, \{x_3\}\}$. The Huffman neighborhood $N_{Huf}(\alpha_0)$ of this partition includes three partitions such that α_0 is their

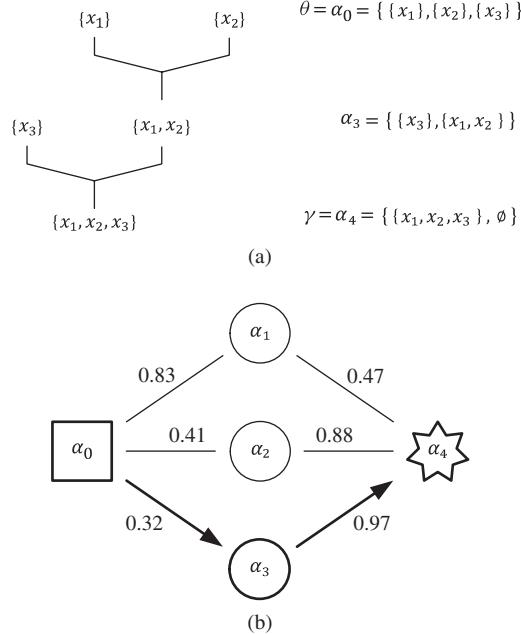


Figure 4.3 Correspondence between the Huffman–Zimmerman procedure and the ILRTA* algorithm. (a) Bottom-up Huffman search tree and corresponding partitions of the sample space. (b) The path created by the ILRTA* algorithm in the partitions space with Rokhlin distances between the partitions.

refinement and whose size is $|\alpha_0| - 1 = 3 - 1 = 2$. All partitions of the space χ except the initial α_0 and final α_4 partitions meet these requirements, so $N_{\text{Huf}}(\alpha_0) = \{\alpha_1, \alpha_2, \alpha_3\}$. The Rokhlin distances to these partitions that form the partition α_0 are as follows:

$$d(\alpha_0, \alpha_1) = 0.83, \quad d(\alpha_0, \alpha_2) = 0.41, \quad \text{and} \quad d(\alpha_0, \alpha_3) = 0.32.$$

Since minimal distance is obtained for the partition α_3 , the algorithm chooses this partition as the next consideration.

The Huffman neighborhood of the partition α_3 includes only the final partition, that is, $N_{\text{Huf}}(\alpha_3) = \{\alpha_4\}$, and the Rokhlin distance between partitions α_3 and α_4 is

$$d(\alpha_3, \alpha_4) = 0.97.$$

Since the partition α_4 is a single Huffman neighbor of α_3 , the algorithm chooses this partition and terminates. The path created by the ILRTA* algorithm in the partitions space χ and the Rokhlin distances between the partitions are shown in Figure 4.3b.

The length of the path $a[\alpha_0, \alpha_3] \circ a[\alpha_3, \alpha_4]$ from the initial partition α_0 to the final partition α_4 is $d(\alpha_0, \alpha_3) + d(\alpha_3, \alpha_4) = 0.32 + 0.97 = 1.29$, while the lengths of the other two paths $a[\alpha_0, \alpha_1] \circ a[\alpha_1, \alpha_4]$ and $a[\alpha_0, \alpha_2] \circ a[\alpha_2, \alpha_4]$ are $d(\alpha_0, \alpha_1) + d(\alpha_1, \alpha_4) = 0.83 + 0.47 = 1.30$ and $d(\alpha_0, \alpha_2) + d(\alpha_2, \alpha_4) = 0.41 + 0.88 = 1.29$. Hence, as expected, the length of the obtained path is minimal, while, as in the previous algorithms, the path with the minimal length is not unique. ■

As indicated above, the Huffman–Zimmerman procedure acts from the bottom leaves level of the tree and continues up to the root level. It was demonstrated that if the ILRTA* algorithm starts from the partition which corresponds to the leaves level, then it creates the search tree that is equal to the optimal Huffman search tree. Now let us consider the relation of the ILRTA* algorithm with the top-down GOTA (Algorithm 2.10), which was considered as an example of the group-testing search algorithms in Section 2.2.4.

As above, let us start with the presentation of the *top-down* GOTA (Algorithm 2.10) in terms of actions over the partitions space. Such a presentation is similar to Algorithm 2.10 but, in contrast, implements the neighborhoods so that the resulting algorithm obtains the same form as the bottom-up procedure considered above.

Recall that in the GOTA, in addition to the sample space X with the location probabilities, and its partitions space χ , there is a strictly positive cost function $c : \chi \times \chi \rightarrow (0, \infty)$ defined so that it provides the cost $c(\alpha, \beta) > 0$ of the movement from partition α to partition β , $\alpha, \beta \in \chi$, in the partitions space χ .

According to Algorithm 2.10, the GOTA starts from the initial partition $\theta = \{X, \emptyset\}$ and moves to the final partition $\gamma \succ \theta$. Then, if α_{curr} is the current partition, for the next chosen partition α_{next} it follows that

$$\theta \prec \dots \prec \alpha_{curr} \prec \alpha_{next} \prec \dots \prec \gamma \quad \text{and} \quad |\alpha_{next}| = |\alpha_{curr}| + 1.$$

Notice again that the direction of the partitions' choice by the *top-down* GOTA is opposite to that by the *bottom-up* Huffman–Zimmerman procedure, where $\theta \succ \dots \succ \alpha_{curr} \succ \alpha_{next} \succ \dots \succ \gamma$ and $|\alpha_{next}| = |\alpha_{curr}| - 1$.

In the GOTA, as follows from Line 6.1 of Algorithm 2.10 and the function `next_partition_GOTA()`, given current partition α_{curr} , the next partition α_{next} is selected as follows:

$$\alpha_{next} \leftarrow \operatorname{argmax}_{\alpha \in \chi, \alpha \succ \alpha_{curr}, |\alpha| = |\alpha_{curr}| + 1} \{H(\gamma | \alpha_{curr}) - H(\gamma | \alpha)\},$$

over partitions $\alpha \in \chi$ such that $\alpha \succ \alpha_{curr}$ and $|\alpha| = |\alpha_{curr}| + 1$, and the GOTA terminates when the selected partition satisfies the condition $\alpha_{curr} \succ \gamma$.

Assume that all possible tests that partition the sample space are available, and that each test has a binary outcome. Following the above domain of maximization, we say that partition β is a *GOTA-neighbor* of the partition α if $\beta \succ \alpha$ and $|\beta| = |\alpha| + 1$. In other words, if $\beta = \{B_1 B_2, \dots, B_m\}$ and it is a neighbor of α , then

$$\alpha = \{B_i \cup B_j, B_1, B_2, \dots, B_{i-1}, B_{i+1}, \dots, B_{j-1}, B_{j+1}, \dots, B_{|\beta|} | B_i \in \beta, i = 1, 2, \dots, |\beta|\}.$$

Such a neighborhood is denoted by $N_{\text{GOTA}}(\cdot)$. Using these terms, the GOTA (Algorithm 2.10) can be outlined as follows.

Algorithm 4.3 (GOTA over partitions space) [21, 23]. Given the partitions space χ , initial partition $\alpha_0 \in \chi$, final partition $\gamma \in \chi$, and a cost function $c : \chi \times \chi \rightarrow (0, \infty)$ do:

1. Set $\alpha_{curr} = \alpha_0$.
2. While $H(\gamma | \alpha_{curr}) \neq 0$ do:
 - 2.1. Obtain the GOTA neighborhood $N_{\text{GOTA}}(\alpha_{curr})$ of the current partition α_{curr} .

- 2.2. Set $\alpha_{next} = \operatorname{argmax}_{\alpha \in N_{GOTA}(\alpha_{curr})} \left\{ \frac{1}{c(\alpha_{curr}, \alpha)} (H(\gamma | \alpha_{curr}) - H(\gamma | \alpha)) \right\}$; ties are broken randomly.
- 2.3. Set $\chi = \chi \setminus \{\alpha_{next}\}$.
- 2.4. Set $\alpha_{curr} = \alpha_{next}$.
3. Return. ■

Usually (see Examples 2.18 and 2.19) in the GOTA, the cost function c is defined by using the probabilities of the considered partitions. Following the introduced terms, it is specified as $c(\alpha, \beta) = \sum_{A \in (\beta \setminus \alpha)} p(A)$. If, in contrast, for every pair of partitions (α, β) , $\alpha \neq \beta$, it is assumed that $c(\alpha, \beta) = 1$, then the selection by the GOTA is formally equivalent to the selection by the above Huffman–Zimmerman procedure (Algorithm 4.2), which results in the search tree as it is built by the ‘division by half’ Algorithm 2.8. Nevertheless, the Huffman–Zimmerman procedure and the GOTA use different neighborhoods and different termination conditions that represent different directions of search tree creation: bottom-up by the Huffman–Zimmerman procedure *versus* top-down by the GOTA.

Similar to the ‘division by half’ Algorithm 2.8 and in contrast to the Huffman–Zimmerman procedure, the search by the GOTA can be implemented online; however, the search tree which is created by the GOTA is near optimal in the sense of the minimum of the given cost function. Recall (see Section 2.2.4) that the total cost of creating the search tree $\mathcal{T}[X]$ by the GOTA is defined by the sum $C(\mathcal{T}[X]) = \sum_{j=0}^{D(\mathcal{T}[X])-1} c(\alpha_j, \alpha_{j+1})$, where $D(\mathcal{T}[X])$ is the depth of the tree $\mathcal{T}[X]$, and α_j is a partition that corresponds to the j th level of the tree. The minimum of the total cost is guaranteed by the next theorem, proven by Hartmann *et al.* [33].

Theorem 4.4 [33]. *If, according to Equation 2.44, for all partitions α , β , and ξ of X such that $\alpha \prec \xi \prec \beta$, regarding the cost function c , it follows that $c(\alpha, \beta) \leq c(\alpha, \xi) + c(\xi, \beta)$, then the GOTA creates the search tree $\mathcal{T}[X]$ with minimal total cost $C(\mathcal{T}[X])$.*

Note that the requirement $c(\alpha, \beta) \leq c(\alpha, \xi) + c(\xi, \beta)$ is a triangle inequality, as it is defined for the metric functions, while the relation $\alpha \prec \xi \prec \beta$ implies that over the partitions space χ the partial ordering is defined. Such an observation relates the GOTA with Lemma 4.4 and allows consideration of different probability mass functions over the sample space X and different cost functions, in particular, such that they implement certain entropy measures.

Now let us demonstrate the relation between the ILRTA* algorithm and the GOTA, while assuming that the GOTA’s requirements for the partitions sequences and for the cost function c are valid. Recall that $N_{GOTA}(\alpha) \subset \chi$ stands for a GOTA neighborhood of the partition $\alpha \in \chi$ in the partitions space.

Theorem 4.5 [21]. *If it follows that $\tilde{d}(\alpha, \beta) = 0$, when $\alpha \in N_{GOTA}(\beta)$ and $\beta \in N_{GOTA}(\alpha)$, and for the cost function c it follows that $c(\alpha, \beta) = -c(\beta, \alpha)$, $\alpha, \beta \in \chi$, then the ILRTA* algorithm (Algorithm 4.2) with the GOTA’s cost function is equivalent to the GOTA.*

Proof. First, let us demonstrate the equivalence of the action choice. Under the assumptions of the theorem, since $\alpha \succ \alpha_{curr}$ one obtains

$$d(\alpha_{curr}, \alpha) = H(\alpha_{curr} | \alpha) + H(\alpha | \alpha_{curr}) = H(\alpha) - H(\alpha_{curr}).$$

Thus, the choice of the next partition in Line 3.3 of the ILRTA* algorithm (Algorithm 4.2) is as follows:

$$\alpha_{next} \leftarrow \operatorname{argmin}_{\alpha \in N(\alpha_{curr})} \{H(\alpha) - H(\alpha_{curr})\}. \quad (4.10)$$

Similarly, under the assumptions of the theorem for the GOTA's utility it follows that

$$\begin{aligned} \frac{1}{c(\alpha_{curr}, \alpha)}(H(\gamma|\alpha_{curr}) - H(\gamma|\alpha)) &= \frac{1}{c(\alpha_{curr}, \alpha)}(H(\alpha) - H(\alpha_{curr})) \\ &= \frac{1}{c(\alpha, \alpha_{curr})}(H(\alpha_{curr}) - H(\alpha)), \end{aligned}$$

where the assumed property of the costs function is applied. Accordingly, the selection of the next partition by the GOTA (see Line 2.2 in Algorithm 4.3) is expressed as follows:

$$\alpha_{next} \leftarrow \operatorname{argmax}_{\alpha \in N_{GOTA}(\alpha_{curr})} \left\{ \frac{1}{c(\alpha, \alpha_{curr})}(H(\alpha_{curr}) - H(\alpha)) \right\},$$

which is equivalent to the selection (Equation 4.10) by ILRTA* algorithm over the GOTA neighborhood and implementation of the GOTA's cost function, that is,

$$\alpha_{next} \leftarrow \operatorname{argmax}_{\alpha \in N_{GOTA}(\alpha_{curr})} \left\{ \frac{1}{c(\alpha_{curr}, \alpha)}(H(\alpha) - H(\alpha_{curr})) \right\}.$$

Now let us consider the termination rule. For the ILRTA* algorithm, it follows that

$$d(\alpha_{curr}, \gamma) = H(\alpha_{curr}|\gamma) + H(\gamma|\alpha_{curr}).$$

If $\alpha_{curr} = \gamma$, then both $H(\alpha_{curr}|\gamma) = 0$ and $H(\gamma|\alpha_{curr}) = 0$, and the equivalence of the GOTA's termination rule to the termination rule of the ILRTA* algorithm is obtained. Let $\alpha_{curr} \neq \gamma$. According to Lemma 4.6, it follows that if $\alpha_{curr} \prec \gamma$ then $H(\gamma|\alpha_{curr}) > 0$, and if $\alpha_{curr} \succ \gamma$ then $H(\gamma|\alpha_{curr}) = 0$. Thus, the termination rules are equal.

The reasons for the estimation update are similar to those used in the proof of Theorem 4.3, which addresses the relation between the ILRTA* algorithm and the Huffman–Zimmerman procedure. ■

Note that in contrast to the Huffman–Zimmerman procedure (Algorithm 4.2), a direct application of the termination rule which is used in the ILRTA* algorithm (Algorithm 4.2) termination rule to the GOTA (Algorithm 4.3) is different from the termination rule which is implemented in the original procedure [33]. From the near-optimality of the GOTA (Algorithm 4.3) stated by Theorem 4.3 and the equivalence between the GOTA and ILRTA* algorithm acting in the GOTA conditions, it follows that the ILRTA* algorithm is near optimal in the same sense as the GOTA.

The actions of the ILRTA* algorithm with the GOTA neighborhood and cost function are illustrated by the following example.

Example 4.2 [21]. Let $X = \{x_1, x_2, x_3, x_4\}$ be a sample space with location probabilities $p(x_1) = 0.1$, $p(x_2) = 0.2$, $p(x_3) = 0.3$, and $p(x_4) = 0.4$. According to Equations 4.5

and 4.6, the partitions space χ of the sample space X consists of 15 partitions, which are as follows:

$$\begin{aligned}\alpha_0 &= \{\{x_1, x_2, x_3, x_4\}, \emptyset\}, \\ \alpha_1 &= \{\{x_1\}, \{x_2, x_3, x_4\}\}, \quad \alpha_2 = \{\{x_2\}, \{x_1, x_3, x_4\}\} \\ \alpha_3 &= \{\{x_3\}, \{x_1, x_2, x_4\}\}, \quad \alpha_4 = \{\{x_4\}, \{x_1, x_2, x_3\}\}, \\ \alpha_5 &= \{\{x_1, x_2\}, \{x_3, x_4\}\}, \quad \alpha_6 = \{\{x_1, x_3\}, \{x_2, x_4\}\}, \quad \alpha_7 = \{\{x_1, x_4\}, \{x_2, x_3\}\}, \\ \alpha_8 &= \{\{x_1\}, \{x_2, x_3\}, \{x_4\}\}, \quad \alpha_9 = \{\{x_1\}, \{x_2\}, \{x_3, x_4\}\}, \quad \alpha_{10} = \{\{x_1\}, \{x_3\}, \{x_2, x_4\}\}, \\ \alpha_{11} &= \{\{x_2\}, \{x_1, x_3\}, \{x_4\}\}, \quad \alpha_{12} = \{\{x_2\}, \{x_1, x_4\}, \{x_3\}\}, \quad \alpha_{13} = \{\{x_3\}, \{x_1, x_2\}, \{x_4\}\}, \\ \alpha_{14} &= \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}\}.\end{aligned}$$

As in Example 2.19, assume that the cost function c is defined by using probabilities of the elements of the partitions as $c(\alpha, \beta) = \sum_{A \in (\beta \setminus \alpha)} p(A)$, while $\alpha \prec \beta$. Then the Rokhlin distances between the GOTA neighbors $\beta \in N_{\text{GOTA}}(\alpha)$ in the partitions space χ weighted by the costs $c(\alpha, \beta)$ are as shown in the tables in Figure 4.4a.

According to the values of the weighted distances that are presented in the figure, the minimal distance between the initial partition α_0 and its GOTA neighbors, $N_{\text{GOTA}}(\alpha_0)$, is reached for the partition α_1 and its value is $d(\alpha_0, \alpha_1)/c(\alpha_0, \alpha_1) = 0.47$. Thus the algorithm chooses the partition α_1 as its next partition. The minimum of the weighted distances over the GOTA neighbors, $N_{\text{GOTA}}(\alpha_1)$ of the partition α_1 , is reached for the partition α_9 and the value of the weighted distance is $d(\alpha_1, \alpha_9)/c(\alpha_1, \alpha_9) = 0.79$. Then the algorithm chooses this partition for the next consideration. The GOTA neighborhood $N_{\text{GOTA}}(\alpha_9)$ of the partition α_9 includes a single partition α_{14} that is the final partition. Thus the algorithm chooses partition α_{14} and terminates. The weighted distance between the partition α_9 and final partition α_{14} is $d(\alpha_9, \alpha_{14})/c(\alpha_9, \alpha_{14}) = 0.99$.

The obtained top-down search tree with corresponding partitions is shown in Figure 4.4b, and the path in the partitions space, which follows the selected partitions, is shown in Figure 4.4c.

In Figure 4.4c, the arrows denote possible alternatives, and the chosen path is denoted by the bold arrows. Note that the chosen partitions and the path shown in the figure are obtained by using weighted distances. ■

So far we have considered representations of both the Huffman procedure and the GOTA by implementing the suggested ILRTA* algorithm to search for a single static target by a *single searcher*. The next section demonstrates that the ILRTA* algorithm allows a simple generalization to search by *multiple searchers*.

4.2.3 Search by the ILRTA* algorithm with multiple searchers

The basis for the implementation of the ILRTA* algorithm (Algorithm 4.2) for search by several searchers is provided by considering the parallelized Huffman–Zimmerman search, which was conducted by Abrahams [34]. In such a parallelized procedure, the search is conducted by a number $m \geq 1$ of searchers, each of which follows the Huffman–Zimmerman search procedure. During the search, each searcher acts on a subset of sample space X , and the searchers do not communicate until one of them finds the target. Below, we present a

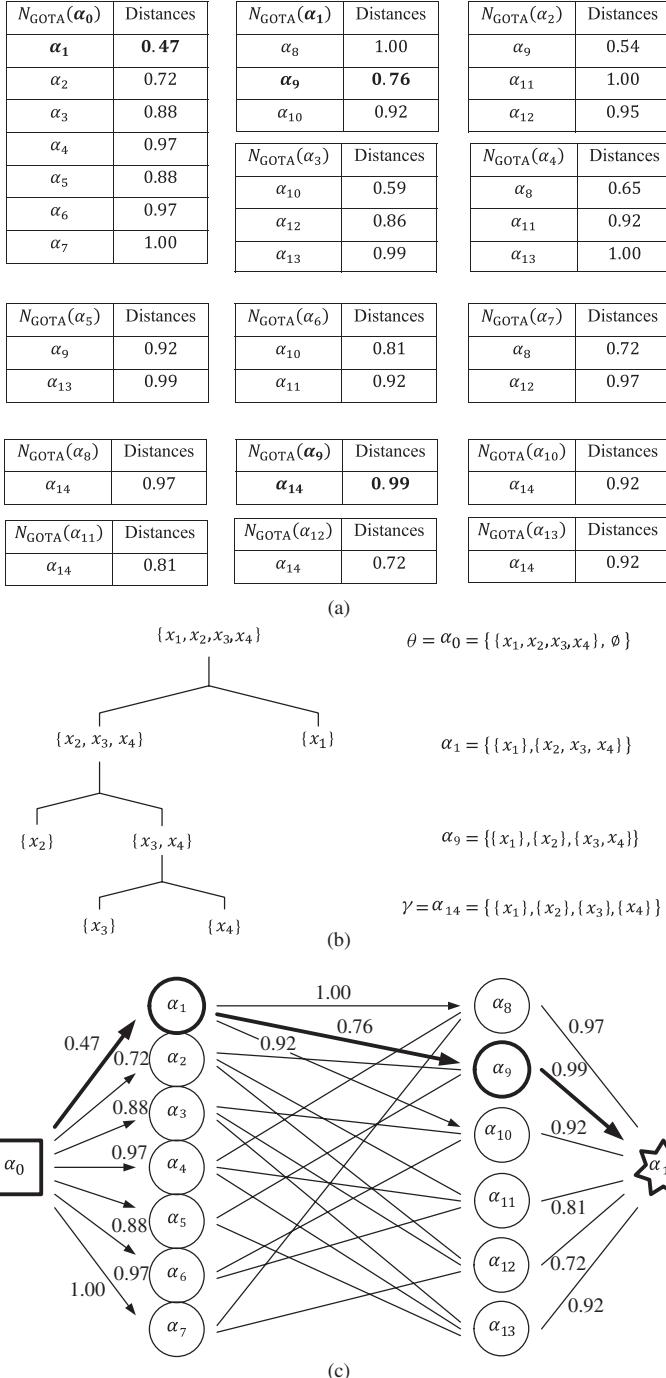


Figure 4.4 Correspondence between the GOTA and the ILRTA* algorithm. (a) GOTA neighborhoods and weighted distances between the partitions of the sample space. (b) Bottom-up GOTA search tree and corresponding partitions of the sample space. (c) The path created by the ILRTA* algorithm acting in the GOTA conditions.

generalization of the suggested ILRTA* algorithm (Algorithm 4.2) to a search by a number $m \geq 1$ of searchers. Note that, as in the Huffman–Zimmerman search, in the Abrahams parallelized search it is assumed that the searcher has to check a single point $x \in X$ of the sample space X to obtain a definite result of whether the target is at this point x , as long as there are no detection errors.

First, let us formulate the results obtained by Abrahams [34] in terms of partitions. Let, as above, $X = \{x_1, x_2, \dots, x_n\}$ be a sample space with location probabilities $p(x_i)$, $i = 1, \dots, n$, $0 \leq p(x_i) \leq 1$, $\sum_{i=1}^n p(x_i) = 1$, and assume that there are $m \geq 1$ searchers looking for a target located at some point $x^* \in X$. Let $\alpha = \{A_1, A_2, \dots, A_m\}$ be a partition of the sample space X such that the size $|\alpha| = m$ of the partition α is equal to the number of searchers. The number of points in the sets $A_k \in \alpha$, $k = 1, \dots, m$, is denoted by n_k , that is, $n_k = |A_k|$. Consequently, we denote by $p_k(x_i)$ the location probabilities of the points $x_i \in A_k$, $k = 1, \dots, m$, $i = 1, \dots, n_k$.

Assume that each searcher follows the Huffman–Zimmerman search procedure (Algorithm 2.9), and denote by $\mathcal{T}[A_k]$, $k = 1, \dots, m$, a Huffman search tree that is built for the search over the set A_k . Following the notation used in Section 2.2.4 (see the inequality (2.39) and Theorem 2.7, in particular), denote by $L(A_k)$ the number of steps required to find a target in the area $A_k \in \alpha$ or to indicate that a target is not located in A_k , $k = 1, \dots, m$; that is, in terms of group-testing search (see Section 2.2), obtain an observation result $z = \mathbf{1}(A_k, x^*) \in \{0, 1\}$. The number of steps $L(A_k)$ is specified as follows [34]:

$$L(A_k) = \frac{1}{p(A_k)} \sum_{i=1}^{n_k} l_k(x_i) p_k(x_i),$$

where $l_k(x_i)$ stands for the number of edges between a root and the leaf x_i , $i = 1, \dots, n_k$, in the search tree $\mathcal{T}[A_k]$, and, as above, $p(A_k) = \sum_{i=1}^{n_k} p_k(x_i)$ is a sum probability that the target is located in one of the points of the set A_k . Then, an average number of search steps over the forest of m trees $\mathcal{T}[A_k]$, $k = 1, \dots, m$, is given by the following value [34]:

$$\begin{aligned} L(\alpha) &= \sum_{k=1}^m p(A_k) L(A_k) \\ &= \sum_{k=1}^m \sum_{i=1}^{n_k} l_k(x_i) p_k(x_i). \end{aligned} \tag{4.11}$$

As in Section 2.2.4 (see Equation 2.40), let

$$H(X) = - \sum_{i=1}^n p_i \log p_i - \sum_{k=1}^m \sum_{i=1}^{n_k} p_k(x_i) \log p_k(x_i)$$

be the entropy of the sample space X , where the equivalence holds true since α is a partition of X . In addition, let

$$H(A_k) = - \sum_{i=1}^{n_k} \frac{p_i}{p(A_k)} \log \frac{p_i}{p(A_k)}$$

be the entropy of the set $A_k \in \alpha$, $k = 1, \dots, m$, which is calculated using the normalized probabilities $p_i/p(A_k)$, $i = 1, \dots, n_k$, and let

$$H(\alpha) = - \sum_{k=1}^m p(A_k) \log p(A_k)$$

be the entropy of partition α , as defined in Section 4.1 (Equation 4.1). Then, regarding the average number of search steps $L(\alpha)$, the following theorem holds.

Theorem 4.6 [34]. (in parallel to Lemmas 2.4 and 2.5; see the inequality (2.39)). *The average number of search steps $L(\alpha)$, which are required for the search according to the forest of m Huffman trees $\mathcal{T}[A_k]$, $k = 1, \dots, m$, is bounded as follows:*

$$H(X) - H(\alpha) \leq L(\alpha) < H(X) - H(\alpha) + 1.$$

Proof. According to Equation 2.39, for each Huffman tree $\mathcal{T}[A_k]$, $k = 1, \dots, m$, it follows that

$$H(A_k) \leq L(A_k) < H(A_k) + 1.$$

Then, for the average number of search steps $L(\alpha) = \sum_{k=1}^m p(A_k)L(A_k)$ we obtain

$$\sum_{k=1}^m p(A_k)H(A_k) \leq L(\alpha) < \sum_{k=1}^m p(A_k)(H(A_k) + 1).$$

Let us consider the lower bound. According to formulas above, we have

$$\begin{aligned} \sum_{k=1}^m p(A_k)H(A_k) &= -\sum_{k=1}^m p(A_k)\sum_{i=1}^{n_k} \frac{p_i}{p(A_k)} \log \frac{p_i}{p(A_k)} \\ &= -\sum_{k=1}^m \sum_{i=1}^{n_k} p_i \log \frac{p_i}{p(A_k)} \\ &= -\sum_{k=1}^m \sum_{i=1}^{n_k} p_i \log p_i + \sum_{k=1}^m \sum_{i=1}^{n_k} p_i \log p(A_k) \\ &= -\sum_{i=1}^n p_i \log p_i + \sum_{k=1}^m p(A_k) \log p(A_k) \\ &= H(X) - H(\alpha). \end{aligned}$$

By using the obtained equality, regarding the upper bound, it follows that

$$\sum_{k=1}^m p(A_k)(H(A_k) + 1) = \sum_{k=1}^m p(A_k)H(A_k) + \sum_{k=1}^m p(A_k) = H(X) - H(\alpha) + 1,$$

as required. ■

From Theorem 4.6, it follows that if α is a discrete partition, that is, $|\alpha| = |X|$, then $0 \leq L(\alpha) < 1$. In such a case, the number of searchers $m = |\alpha|$ is equal to the size of the sample space n ; the searchers are spread over all points of the sample space and no search has to be initiated. On the other hand, if $\alpha = \{X, \emptyset\}$ is a trivial partition, $|\alpha| = m = 1$, then Theorem 4.6 results in the bounds $H(X) \leq L(\alpha) < H(X) + 1$, as required by the Huffman–Zimmerman search by a single searcher.

In the Abrahams parallelized search the searchers act in parallel and do not communicate during their actions. Below, in Section 5.1.1, we will use this approach for the development of the recursive ILRTA* algorithm. In addition, note that, in a similar manner as in Theorem 4.6, the bound for the parallelized Hu–Tucker search can be obtained [34]. However, since the Hu–Tucker search procedure [35] is less effective than the Huffman–Zimmerman search, we do not consider such bounds here.

Let us consider a generalization of the Abrahams search and assume that the searchers are allowed to communicate during the search. We assume that the searchers are *Bayesian rational* [36]: that is, given suitable information, each searcher chooses such a strategy that maximizes the searcher's reward. Such an assumption implies that for an unknown

finite number of steps and for the ILRTA* algorithm's informational distances considered as reward functions, the searchers are indeed Bayesian rational. In fact, the informational distances defined for the partitions do not strictly increase with the search steps. Thus, being in the current state, the searcher cannot assume that movement to the next state will increase the available information about the target. This means that for an unknown finite number of steps the searcher has to choose such a state that supports the maximum informational reward at the current state.

For an informational search by a single searcher, the Bayesian rationality corresponds to the search using maximum information strategies, while the search by a number of searchers implies a search using those strategies that support the maximum relative information given the choices of the other searchers. Let us consider the choices of the partitions by a number $m \geq 1$ of searchers.

Let X be a sample space given the location probabilities, and let χ be the partitions space on which the Rokhlin metric d given by Equation 4.3 is defined. Denote by $\chi(k) \subset \chi$ a partitions space such that the size of every partition $\xi \in \chi(k)$ is less than or equivalent to k , that is, $|\xi| \leq k$.

Let $m \geq 1$ be the number of searchers conducting the search for a single static target over the sample space X of size $n = |X|$; we assume that $m \leq n$. Denote by α_j^t a partition of the sample space X that is chosen by the j th searcher at time t . Recall (see Section 4.1) that for two partitions α and β , multiplication $\alpha \vee \beta$ results in the partition that consists of all possible intersections $A \cap B$ of the sets $A \in \alpha$ and $B \in \beta$. We denote by

$$\alpha_{-i}^t = \vee_{j=1, j \neq i}^m \alpha_j^t = \alpha_1^t \vee \alpha_2^t \vee \cdots \vee \alpha_{i-1}^t \vee \alpha_{i+1}^t \vee \cdots \vee \alpha_m^t$$

the result of multiplication of m partitions excluding partition α_i^t and by

$$\alpha^t = \vee_{j=1}^{j=1} \alpha_j^t = \alpha_1^t \vee \alpha_2^t \vee \cdots \vee \alpha_m^t$$

the result of multiplication of m partitions. As above, following Equation 4.3, for two partitions α and β we define the Rokhlin distance by $d(\alpha, \beta) = H(\alpha|\beta) + H(\beta|\alpha)$, and for the initial partition α_0^t we specify

$$d(\alpha_{-0}^t, \alpha_0^t) = H(\alpha_0^t).$$

According to the assumption that the searchers are allowed to communicate during the search, there are two possible methods for selecting the partitions:

- **Non-coalitional choice:** The searchers choose their partitions sequentially one after another while the choice of the next searcher does not influence the choice of the previous searchers.
- **Coalitional choice:** The choices of next searchers influence the choices of the previous searchers.

Let us consider these possibilities in particular. In the following consideration we essentially use the properties of the entropy of partitions given by Lemma 4.6.

Theorem 4.7 (non-coalitional choice) [21, 37]. *Assume that there is a search procedure with $m \geq 1$ searchers, and the choice is sequential according to the searcher's index i . Then, if the i th searcher chooses a partition α_i^t such that the distance $d(\alpha_{-i}^t, \alpha_i^t)$ is a maximum*

over all possible partitions from $\chi(2)$, then the entropy $H(\alpha^t)$ of the partition $\alpha^t = \vee_{j=1}^m \alpha_j^t$, increases with i in $\chi(i+2)$.

Proof. If $m = 1$, then the search is conducted by a single searcher and the statement of the theorem is equivalent to the definition of the strategies for Bayesian rational searchers as implemented in the ILRTA* algorithm (Algorithm 4.4).

Assume that $m > 1$. Given $\alpha = \vee_{j=1}^{i-1} \xi_j$, let ξ_i be a partition which provides a maximum of the distance $d(\alpha, \xi_i) = \max_{\zeta \in \chi(2)} \{d(\alpha, \zeta)\}$ in the partitions space $\chi(i+2)$. Then, we need to demonstrate that

$$H(\alpha \vee \xi_i) = \max_{\zeta \in \chi(2)} \{H(\alpha \vee \zeta)\}.$$

Let, in contrast, $\beta \in \chi(2)$ be a partition such that $d(\alpha, \beta) < d(\alpha, \xi_i)$ and $H(\alpha \vee \beta) > H(\alpha \vee \xi_i)$. Then, according to the definition of the Rokhlin distance d and the properties of the entropy of partitions specified by Lemma 4.6, it follows that

$$d(\alpha, \xi_i) = 2H(\alpha \vee \xi_i) - H(\alpha) - H(\xi_i) \quad \text{and} \quad d(\alpha, \gamma) = 2H(\alpha \vee \beta) - H(\alpha) - H(\beta),$$

where $\gamma = \{\{x_1\}, \{x_2\}, \dots, \{x_n\}\}$, as above, stands for the discrete partition of the sample space. Thus, for the partition β the following conditions are satisfied:

$$2H(\alpha \vee \beta) - H(\beta) < 2H(\alpha \vee \xi_i) - H(\xi_i) \quad \text{and} \quad H(\alpha \vee \beta) > H(\alpha \vee \xi_i),$$

from which it follows that $H(\beta) > H(\xi_i)$.

On the other hand, from the properties of the entropy, it follows that

$$d(\alpha, \xi_i) \leq H(\alpha) + H(\xi_i) \quad \text{and} \quad d(\alpha, \beta) \leq H(\alpha) + H(\beta).$$

Thus, $d(\alpha, \beta) < d(\alpha, \xi_i)$ results in $d(\alpha, \beta) < H(\alpha) + H(\xi_i)$ and gives $H(\beta) < H(\xi_i)$, which contradicts the previously obtained relation between $H(\beta)$ and $H(\xi_i)$. ■

The theorem gives a sufficient (but not necessary) condition for the *non-coalitional choice* of partitions by m Bayesian rational searchers such that the obtained expected number of steps is minimal over all such search procedures.

Now let us consider the *coalitional choice* of partitions. The proof of the sufficient condition, which minimizes the expected number of steps, is based on the following lemma.

Lemma 4.8 [21, 37]. *If the distances $d(\alpha_i^t, \alpha_j^t)$ between all pairs of partitions $\alpha_i^t, \alpha_j^t \in \chi(2)$ chosen by different searchers $i \neq j$ are equivalent and maximal over partitions space $\chi(2)$, then the entropy $H(\alpha^t)$ reaches its maximum over partitions space $\chi(m+1)$.*

Proof. If $1 \leq m \leq 2$ then the proposition of the lemma follows directly from Theorem 4.4 corresponding to the non-coalitional choice.

Let $m \geq 3$ and let, as above, $\alpha_0 = \{X, \emptyset\}$. It is clear that $H(\alpha_0) = 0$, and from the properties of the entropy of partitions (see Lemma 4.6) it follows that $d(\alpha_0, \xi) = H(\xi)$ for every $\xi \in \chi(m+2)$, $m = 1, 2, \dots$. In addition, it follows that $\max_{\xi \in \chi(2)} d(\alpha_0, \xi) = 1$ over all possible probability mass functions on X . Then, the partitions space $\chi(2)$ with the

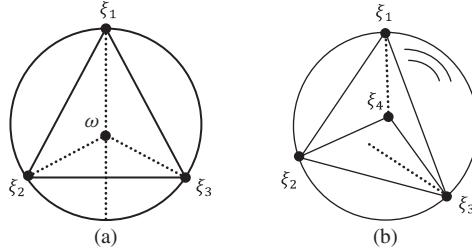


Figure 4.5 Partitions space for the search by three and four searchers. (a) Space $\chi(2)$ for $m = 3$ searchers. (b) Space $\chi(2)$ for $m = 4$ searchers.

Rokhlin distance d is a bounded ball with a center $\omega \in \chi(2)$ and a boundary $\partial = \{\xi | \xi \in \chi(2), d(\omega, \xi) = 1/2\}$.

From the equality of the maximal distances between the considered partitions it results that these partitions lie on the sphere ∂ and are the vertexes of the perfect polyhedron inscribed in the sphere ∂ . For example, if $m = 3$ then ∂ is a circumference and the polyhedron is a triangle; if $m = 4$ then ∂ is a two-dimensional sphere in three-dimensional space and the polyhedron is a tetrahedron; and so on. The cases of $m = 3$ and $m = 4$ are illustrated in Figure 4.5.

For the Rokhlin metric d , the volumes of the indicated polyhedrons are proportional to the entropy $H(\xi)$. Thus, to obtain the proposition of the lemma it remains to note that the volume of the polyhedron inscribed in the sphere is maximal if and only if its edges have equal lengths. ■

The choice of the partitions by a coalition of m searchers is governed by the following theorem.

Theorem 4.8 (coalitional choice) [21, 37]. *Let $\omega \in \chi(2)$ be the center of the partitions space $\chi(2)$. If the searchers choose partitions $\alpha_i^t \in \chi(2)$ such that $d(\omega, \alpha_i^t) = 1/2$ and $d(\alpha_{-i}^t, \alpha_i^t) = \sqrt{(m+1)/2m}$, then the entropy $H(\alpha^t)$ reaches its maximum over partitions space χ .*

Proof. Consider a D -dimensional perfect tetrahedron inscribed in a $(D-1)$ -dimensional sphere with radius R_D . Denote by a the length of the edge of the tetrahedron. As shown by Buchholz [38], the radius R_D and the length a follow the equation

$$R_D = \frac{a}{\sqrt{2}} \sqrt{D/(D+1)}. \quad (4.12)$$

If $D = 2$ and $D = 3$ this gives the elementary formulas $R_2 = a/\sqrt{3}$ for the triangle and $R_3 = (a/4)\sqrt{6}$ for the tetrahedron, respectively.

Let $\partial = \{\xi | \xi \in \chi(2), d(\omega, \xi) = 1/2\}$, $\omega \in \chi(2)$, be the circumsphere for the $(m+1)$ -dimensional tetrahedron with the vertices ξ_i . The volume of the tetrahedron is proportional to the entropy $H(\vee_{i=1}^{m+1} \xi_i)$ and reaches its maximum if the tetrahedron is perfect, that is, if its vertices ξ_i lie on the sphere ∂ . Thus, $d(\omega, \xi_i) = 1/2$. Using Equation 4.12 with $D = m+1$ and $R_{m+1} = 1/2$ we obtain $d(\xi_i, \xi_j) = \sqrt{(m+1)/2m}$, $i \neq j$, as required. ■

On the basis of these statements, the procedure for the search for a single static target by a number $m \geq 1$ of searchers is formulated as follows.

Algorithm 4.4 (search by m searchers) [21, 37]. Given sample space X of size $n = |X|$ with location probabilities $p^0(x)$, $x \in X$, partitions space χ , and a number of searchers m , $1 \leq m \leq n$, do:

1. Set $t = 1$.
2. Do
 - 2.1. For each searcher $i = 1, \dots, m$ do:
 - 2.1.1 Choose partition $\alpha_i^t \in \chi(2)$ such that
 - a. Non-coalitional search: $d(\alpha_{-i}^t, \alpha_i^t)$ is maximal over $\chi(i+2)$;
 - b. Coalitional search: $d(\omega, \alpha_i^t) = \frac{1}{2}$, $d(\alpha_{-i}^t, \alpha_i^t) = \sqrt{\frac{m+1}{2m}}$.
 - 2.2. Set $\alpha^t = \vee_{i=1}^m \alpha_i^t$.
 - 2.3. For each subset $A \in \alpha^t$ do:

Update location probabilities $p^t(x)$ for $x \in A$ according to the Bayes rule.
 - 2.4. For each point $x \in X$ do: Set $p^{t+1}(x) = p^t(x)$.
 - 2.5. Set $t = t + 1$.
 3. While $p^t(x) \neq p^{t-1}(x)$ for each $x \in X$.
 3. Return point x such that $p^t(x) = 1$. ■

The actions of Algorithm 4.4 are illustrated by the trials presented in the following example.

Example 4.3 [21]. Let $X = \{x_1, x_2, \dots, x_{10}\}$ be a sample space, and assume that there are $m = 2$ searchers looking for a target. Note that for the search by $m = 2$ searchers the non-coalitional and coalitional choices of the partitions give the same results. As above, denote by $\theta = \alpha_0 = \{X, \emptyset\}$ a trivial partition of X .

Trial 1. Assume that the target is located at point $x_8 \in X$ of the sample space X .

Step $t = 0$. Let initial location probabilities $p_i^0 = p^0(x_i)$, $i = 1, 2, \dots, 10$, be defined as follows:

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
p_i^0	0.045	0.098	0.043	0.004	0.259	0.091	0.085	0.077	0.225	0.076

Then, the first searcher chooses the partition

$$\alpha_1^0 = \{\{x_1, x_3, x_5, x_8, x_{10}\}, \{x_2, x_4, x_6, x_7, x_9\}\}$$

that gives $d(\theta, \alpha_1^0) = H(\alpha_1^0) = 1$, and the second searcher chooses the partition

$$\alpha_2^0 = \{\{x_1, x_3, x_4, x_8, x_9, x_{10}\}, \{x_2, x_5, x_6, x_7\}\}$$

such that $d(\alpha_1^0, \alpha_2^0) = 1.995$ is maximal.

The resulting partition α^0 for updating location probabilities and inspection at the current step $t = 0$ is as follows:

$$\alpha^0 = \alpha_1^0 \vee \alpha_2^0 = \{\{x_1, x_3, x_8, x_{10}\}, \{x_5\}, \{x_4, x_9\}, \{x_2, x_6, x_7\}\}.$$

Step t = 1. After inspection and probability updating by a Bayesian procedure, the following location probabilities $p_i^1 = p^1(x_i)$, $i = 1, 2, \dots, 10$, are obtained:

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
p_i^1	0.185	0	0.180	0	0	0	0	0.320	0	0.315

Given these location probabilities, the choice of the first searcher is a partition

$$\alpha_1^1 = \{\{x_1, x_2, x_4, x_5, x_6, x_7, x_9, x_{10}\}, \{x_3, x_8\}\}$$

with $d(\theta, \alpha_1^1) = H(\alpha_1^1) = 0.999$, and the choice of the second searcher is a partition

$$\alpha_2^1 = \{\{x_2, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}, \{x_1, x_3\}\}$$

that results in the maximal distance $d(\alpha_1^1, \alpha_2^1) = 1.947$.

The resulting partition α^1 for updating location probabilities and inspection at the current step $t = 1$ is as follows:

$$\alpha^1 = \alpha_1^1 \vee \alpha_2^1 = \{\{x_2, x_4, x_5, x_6, x_7, x_9, x_{10}\}, \{x_1\}, \{x_3\}, \{x_8\}\}.$$

Step t = 2. After inspection and probability updating, the following location probabilities $p_i^2 = p^2(x_i)$, $i = 1, 2, \dots, 10$, are obtained:

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
p_i^2	0	0	0	0	0	0	0	1	0	0

Result of Trial 1. The target is found at point x_8 after two steps.

The next trial gives an example of a one-step search procedure.

Trial 2. Assume that the target is located at the point x_4 and that the search runs as follows.

Step t = 0. Initial probabilities $p_i^0 = p^0(x_i)$, $i = 1, 2, \dots, 10$, are defined as follows:

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
p_i^0	0.186	0.002	0.061	0.252	0.255	0.129	0.022	0.008	0.064	0.021

After the searchers' selection of the partitions α_1^0 and α_2^0 , the resulting partition for updating probabilities and inspection is as follows:

$$\alpha^0 = \alpha_1^0 \vee \alpha_2^0 = \{\{x_1, x_2, x_3\}, \{x_4\}, \{x_5\}, \{x_6, x_7, x_8, x_9, x_{10}\}\}.$$

Step t = 1. Inspection and probability updating on the previous step $t = 0$ results in the following location probabilities $p_i^1 = p^1(x_i)$, $i = 1, 2, \dots, 10$, for consideration at the current step $t = 1$:

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
p_i^1	0	0	0	1	0	0	0	0	0	0

Result of Trial 2. The target is found at the point x_4 in one step.

The final trial illustrates the search that finishes in three steps.

Trial 3. Let the target be located at the point x_9 . Following location probabilities defined at the initial step $t = 0$, the search runs as follows.

Step t = 0. Assume that the initial probabilities $p_i^0 = p^0(x_i)$, $i = 1, 2, \dots, 10$, are as follows:

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
p_i^0	0.115	0.180	0.136	0.136	0.068	0.195	0.017	0.126	0.015	0.012

After the searchers' choices, the partition for inspection and probability updating is as follows:

$$\alpha^0 = \alpha_1^0 \vee \alpha_2^0 = \{\{x_6, x_7, x_9\}, \{x_3, x_4\}, \{x_1, x_8\}, \{x_2, x_5, x_{10}\}\}.$$

Step t = 1. Following the inspection and the update at step $t = 0$, the location probabilities $p_i^1 = p^1(x_i)$, $i = 1, 2, \dots, 10$, are as follows:

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
p_i^1	0	0	0	0	0	0.856	0.076	0	0.068	0

Then, the partition for inspection and probability updating is

$$\alpha^1 = \alpha_1^1 \vee \alpha_2^1 = \{\{x_1, x_2, x_3, x_4, x_5, x_8, x_{10}\}, \{x_6\}, \{x_7, x_9\}\}.$$

Step t = 2. The location probabilities $p_i^2 = p^2(x_i)$, $i = 1, 2, \dots, 10$, after inspection and update are as follows:

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
p_i^2	0	0	0	0	0	0	0.527	0	0.473	0

The partition for inspection and probability updating is

$$\alpha^2 = \alpha_1^2 \vee \alpha_2^2 = \{\{x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_{10}\}, \{x_4\}, \{x_9\}\}.$$

Step t = 3. The location probabilities $p_i^3 = p^3(x_i)$, $i = 1, 2, \dots, 10$, after inspection and update are as follows:

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
p_i^3	0	0	0	0	0	0	0	0	1	0

Result of Trial 3. The target is found at the point x_3 after three steps.

These trials demonstrate the search by multiple searchers that is similar to the search acting on the partitions space such that each next partition is a refinement of the current partition. ■

The ILRTA* algorithm implements a real-time search over the partitions space and presents a general framework for group-testing search for a static target. In the next section, similar methods are applied for the algorithm of search for moving targets.

4.3 Moving target search: Informational moving target search algorithm

Let us continue the implementation of the algorithms of search over graphs to the search over partitions space. In Section 2.3.3, we considered the methods of search for a moving target and presented the MTS algorithm (Algorithm 2.14), developed by Ishida and Korf [39, 40]. The MTS algorithm follows the line of the LRTA* algorithm (Algorithm 2.13) and implements the updating of the evaluated costs that are determined by movements of the target. In this section, as for the ILRTA* algorithm (Algorithm 4.4) above, we apply the MTS algorithm for a search over the partitions space χ with the defined Rokhlin distance. The suggested algorithm is called the *Informational Moving Target Search* algorithm [21, 27, 41]. In the same manner, the informational versions of the Fringe-Retrieving A*) (FRA*) algorithm (Algorithm 2.15) and the similar A^* -type algorithms can be developed.

4.3.1 The informational MTS algorithm and its properties

We start with the formulation of the IMTS algorithm. This algorithm is based on the same principles as the ILRTA* algorithm (Algorithm 4.4) above and implements a general group-testing search procedure (Procedure 2.1) in search for a moving target. The algorithm acts over the space of partitions of the sample space and uses the informational actual and estimated distances.

Let $X = \{x_1, x_2, \dots, x_n\}$ be a sample space with corresponding location probabilities $p(x_i)$, $i = 1, \dots, n$, $0 \leq p(x_i) \leq 1$, $\sum_{i=1}^n p(x_i) = 1$, and, as above, let χ be the set of all possible partitions of X . Similar to the previous consideration of the ILRTA* algorithm

(see Section 4.2.1), let d be the Rokhlin distance as defined by Equation 4.3 and let \tilde{d} be another metric defined on χ such that for every $\alpha, \beta \in \chi$ it follows that

$$\tilde{d}(\alpha, \beta) \leq d(\alpha, \beta), \quad (4.13)$$

which provides both the admissibility assumption (Equation 4.7) and the consistency assumption (Equation 4.8). As indicated above, if d is the Rokhlin metric (Equation 4.3) and \tilde{d} is the Ornstein metric d_{Orn} , then, from Lemma 4.3, the requirement (Equation 4.13) holds. Again, without loss of generality one can assume that $\tilde{d}(\alpha, \beta) = 0$ and the algorithm can still be applied.

For the purposes of the IMTS algorithm, the neighborhoods of the partitions in the partitions space χ are defined as follows. Let r be a positive number, $0 < r \leq H(X)$, where $H(X)$ is the entropy of sample space X ; the number r represents an *observation radius* and specifies a certain knowledge of the distances between the partitions. For any partition $\alpha \in \chi$, we say that the set $N(\alpha, r) \subset \chi$ is a *set of neighbors* of α if $N(\alpha, r)$ meets the following requirements:

- $\alpha \notin N(\alpha, r)$;
- for every partition $\beta \in N(\alpha, r)$ it follows that $\tilde{d}(\beta, \alpha) \leq r \leq d(\beta, \alpha)$.

The second requirement generalizes the neighborhood definition used in the ILRTA* algorithm (see Section 4.2.1). In addition, note that in contrast to the neighborhood definition that appears in the MTS algorithm (Algorithm 2.14), the presented definition essentially uses distances and distance estimations between the partitions. A similar definition was implemented in the ILRTA* algorithm and it is often justified in practical applications.

Denote by $\tau \in \chi$ (with suitable indices) the partitions chosen from the partitions space χ by the target; often the target's partition τ is defined by a single-point subset and its complementary subset of X . Accordingly, partitions $\alpha \in \chi$ (with suitable indices) will be considered as the partitions chosen from the partitions space χ by the searcher.

Assume that the target starts from its initial partition ϑ and moves over the partitions space χ or, similar to the MTS algorithm (Algorithm 2.14), over a certain subset $\chi_{tar} \subset \chi$, and let it choose at each step a partition τ_{next} such that the distance from its current partition τ_{curr} is less than r , that is, $d(\tau_{curr}, \tau_{next}) < r$. Such an assumption includes the possibility that the target remains with its current partition, and, in contrast to the MTS algorithm, allows variable lengths of steps.

Let θ and ϑ be the initial searcher's and target's partitions, respectively, and let $\tilde{d}_0(\alpha, \tau)$, $\alpha, \tau \in \chi$, be the initial distance estimations. In the worst case, from the searcher's viewpoint it may be assumed that $\tilde{d}_0(\alpha, \tau) = 0$ for all $\alpha, \tau \in \chi$. Then, the IMTS algorithm is formulated similarly to the MTS algorithm (Algorithm 2.14) and is outlined as follows.

Algorithm 4.5 (IMTS algorithm) [21, 27, 41]. Given the partitions space χ with admissible metrics \tilde{d} and d , initial searcher's partition $\theta \in \chi$ and initial target's partition $\vartheta \in \chi$, observation radius r , and initial distance estimations $\tilde{d}_0(\alpha, \tau)$, $\alpha, \tau \in \chi$, do:

1. Set $\alpha_{curr} = \theta$.
2. Set $\tau_{curr} = \vartheta$.

3. While $d(\alpha_{curr}, \tau_{curr}) \neq 0$ do:

Searcher's turn:

3.1. Obtain the neighborhood $N(\alpha_{curr}, r)$ of the current partition α_{curr} : The neighborhood $N(\alpha_{curr}, r)$ includes such partitions $\xi \in \chi$, except α_{curr} , that is, $\tilde{d}(\xi, \alpha) \leq r \leq d(\xi, \alpha)$.

3.2. Set $\alpha_{next} = \operatorname{argmin}_{\alpha \in N(\alpha_{curr}, r)} \{\tilde{d}(\alpha, \tau_{curr})\}$; ties are broken randomly.

3.3. Set $\tilde{d}(\alpha_{curr}, \tau_{curr}) = \max\{\tilde{d}(\alpha_{curr}, \tau_{curr}), \min_{\alpha \in N(\alpha_{curr})} \{\tilde{d}(\alpha, \tau_{curr}) + r\}\}$.

3.4. Set $\alpha_{curr} = \alpha_{next}$.

Target's turn:

3.5. Target moves from τ_{curr} to τ_{next} , $d(\tau_{curr}, \tau_{next}) < r$, or stays in τ_{curr} .

3.6. Set $\tilde{d}(\alpha_{curr}, \tau_{curr}) = \max\{\tilde{d}(\alpha_{curr}, \tau_{curr}), \tilde{d}(\alpha_{curr}, \tau_{next}) - r\}$.

3.7. Set $\tau_{curr} = \tau_{next}$.

4. Return α_{curr} . ■

The actions of the IMTS algorithm (Algorithm 4.5) are similar to those of the basic MTS algorithm (Algorithm 2.14), while the estimated and actual costs are defined by the estimated and actual informational distances, as implemented in the ILRTA* algorithm (Algorithm 4.1). In addition, similar to the ILRTA* algorithm, the neighborhoods in the IMTS algorithm are specified according to the actual and estimated distances rather than by the adjacent vertices of the graph, as defined in the MTS algorithm. The correctness of the defined actual and estimated distances and the neighborhoods is guaranteed by the next lemma.

Lemma 4.9 [21]. *The requirements for the neighborhood and the target's movement are correct in terms of distances and distance estimations.*

Proof. Let the searcher's partition be α_{curr} and the target's partition be τ_{curr} , and assume that the searcher moves to partition $\alpha_{next} \in N(\alpha_{curr}, r)$ and the target moves to partition τ_{next} such that $d(\tau_{curr}, \tau_{next}) < r$.

Since $\alpha_{next} \in N(\alpha_{curr}, r)$, it follows that $\tilde{d}(\alpha_{curr}, \alpha_{next}) \leq r \leq d(\alpha_{curr}, \alpha_{next})$, and according to the assumption regarding the target's movement it follows that $d(\tau_{curr}, \tau_{next}) \leq d(\tau_{curr}, \tau_{next}) < r$. Then, we need to show that the inequality

$$d(\alpha_{next}, \tau_{next}) - \tilde{d}(\alpha_{next}, \tau_{next}) \leq d(\alpha_{curr}, \tau_{next}) - \tilde{d}(\alpha_{curr}, \tau_{next})$$

holds.

If $\tau_{next} \in N(\alpha_{curr}, r)$, then $\tilde{d}(\alpha_{next}, \tau_{next}) = 0$, and since distance estimation \tilde{d} is a metric, it implies that $\alpha_{next} = \tau_{next}$. Thus, from the metric properties of actual distance d it follows that $d(\alpha_{next}, \tau_{next}) = 0$, as well. Hence, the required inequality becomes

$$0 \leq d(\alpha_{curr}, \tau_{next}) - \tilde{d}(\alpha_{curr}, \tau_{next}),$$

which is the admissibility assumption (Equation 4.13).

Let $\tau_{next} \notin N(\alpha_{curr}, r)$. Then $\tilde{d}(\alpha_{curr}, \tau_{next}) > r$, for which it follows that $d(\alpha_{curr}, \tau_{next}) > r$. Therefore, one obtains

$$d(\alpha_{next}, \tau_{next}) \leq d(\alpha_{curr}, \alpha_{next}) + d(\alpha_{curr}, \tau_{next}),$$

while $r \leq d(\alpha_{curr}, \alpha_{next})$ and $r < d(\alpha_{curr}, \tau_{next})$; and

$$\tilde{d}(\alpha_{next}, \tau_{next}) \leq \tilde{d}(\alpha_{curr}, \alpha_{next}) + \tilde{d}(\alpha_{curr}, \tau_{next}),$$

while $r \geq \tilde{d}(\alpha_{curr}, \alpha_{next})$ and $r < \tilde{d}(\alpha_{curr}, \tau_{next})$.

Subtracting the second inequality from the first, one obtains

$$\begin{aligned} d(\alpha_{next}, \tau_{next}) - \tilde{d}(\alpha_{next}, \tau_{next}) &\leq d(\alpha_{curr}, \alpha_{next}) + d(\alpha_{curr}, \tau_{next}) \\ &\quad - \tilde{d}(\alpha_{curr}, \alpha_{next}) - \tilde{d}(\alpha_{curr}, \tau_{next}), \end{aligned}$$

which holds since, as indicated, $d(\alpha_{curr}, \alpha_{next}) \geq r$, $d(\alpha_{curr}, \tau_{next}) > r$, $\tilde{d}(\alpha_{curr}, \alpha_{next}) \leq r$, and $\tilde{d}(\alpha_{curr}, \tau_{next}) > r$. ■

The relations between the estimated and actual distances and corresponding neighborhoods are illustrated in Figure 4.6.

In the figure, the algorithm started with the initial partition $\theta = \xi_0$ and at the current step the considered partition is $\alpha_{curr} = \xi_1$. The target's partition at that step is $\tau_{curr} = \xi_6$. The neighborhood $N(\alpha_{curr}, r)$ of the current partition α_{curr} includes two following partitions $N(\alpha_{curr}, r) = \{\xi_2, \xi_3\}$, for which estimated and actual distances meet the inequalities $\tilde{d}(\alpha_{curr}, \xi_2) \leq r \leq d(\alpha_{curr}, \xi_2)$ and $\tilde{d}(\alpha_{curr}, \xi_3) \leq r \leq d(\alpha_{curr}, \xi_3)$, while for the other partitions the admissibility assumption (Equation 4.13) holds. The neighbors of the target's current partition τ_{curr} are ξ_4 , ξ_5 , and ξ_7 , such that it follows that $d(\tau_{curr}, \xi_4) < r$, $d(\tau_{curr}, \xi_5) < r$, and $d(\tau_{curr}, \xi_7) < r$.

Now let us consider the properties of the suggested IMTS algorithm (Algorithm 4.5) in parallel to the properties of the MTS algorithm (Algorithm 2.14).

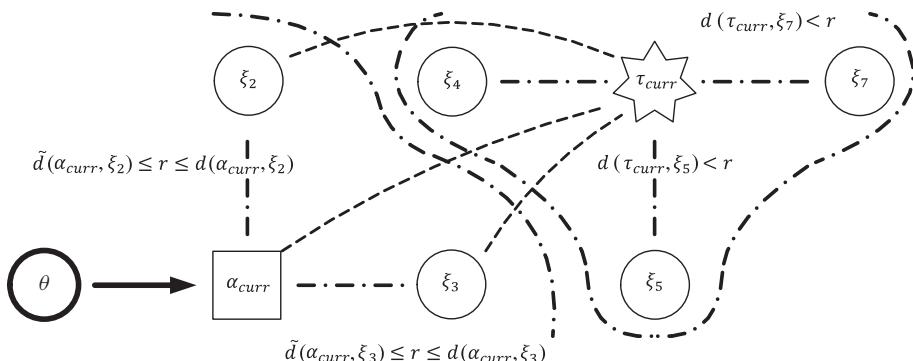


Figure 4.6 Actual and estimated distances and the neighborhoods of the partitions in the IMTS algorithm.

Lemma 4.10 [21]. (in parallel to Lemma 2.13). *Throughout the trial of Algorithm 4.5 (IMTS algorithm) and at its termination, for all searcher's partitions α and target's partitions τ , $\alpha, \tau \in \chi$, the admissibility $\tilde{d}(\alpha, \tau) \leq d(\alpha, \tau)$ is preserved.*

Proof. As required by the algorithm, assume that $\tilde{d}_0(\alpha, \tau) \leq d(\alpha, \tau)$ for every available pair $(\alpha, \tau) \in \chi \times \chi$. We need to show that, following the steps in the IMTS algorithm, it is guaranteed that $\tilde{d}(\alpha_{curr}, \tau_{next}) \leq d(\alpha_{curr}, \tau_{next})$. Let us consider the turns of the search and the target defined by the IMTS algorithm (Algorithm 4.5).

Searcher's turn. If $\tilde{d}(\alpha_{curr}, \tau_{curr}) \geq \min_{\alpha \in N(\alpha_{curr})} \{\tilde{d}(\alpha, \tau_{curr}) + r\}$, then, according to Line 3.3 of the algorithm, the value of the estimated distance $\tilde{d}(\alpha_{curr}, \tau_{curr})$ is not changed by the updating and the admissibility is preserved.

Assume that $\tilde{d}(\alpha_{curr}, \tau_{curr}) < \min_{\alpha \in N(\alpha_{curr})} \{\tilde{d}(\alpha, \tau_{curr}) + r\}$ and define

$$\alpha_{\min} = \operatorname{argmin}_{\alpha \in N(\alpha_{curr}, r)} \{\tilde{d}(\alpha, \tau_{curr})\}.$$

Then, according to the updating defined by Line 3.3, it follows that $\tilde{d}(\alpha_{curr}, \tau_{curr}) < \tilde{d}(\alpha_{\min}, \tau_{curr}) + r$ and it is required to demonstrate that

$$\tilde{d}(\alpha_{\min}, \tau_{curr}) + r \leq d(\alpha_{curr}, \tau_{curr}).$$

If $\tau_{curr} \in N(\alpha_{curr}, r)$ then $\tilde{d}(\alpha_{\min}, \tau_{curr}) = 0$ and $d(\alpha_{curr}, \tau_{curr}) = d(\alpha_{curr}, \alpha_{\min})$. Thus, the required inequality becomes

$$r \leq d(\alpha_{curr}, \alpha_{\min}), \alpha_{\min} \in N(\alpha_{curr}, r),$$

which holds according to the definition of neighborhood.

Let $\tau_{curr} \notin N(\alpha_{curr}, r)$. Thus $r < \tilde{d}(\alpha_{curr}, \tau_{curr})$, and from the triangle inequality it follows that

$$r + \tilde{d}(\alpha_{\min}, \tau_{curr}) \leq d(\alpha_{curr}, \alpha_{\min}) + d(\alpha_{\min}, \tau_{curr}).$$

Now, recalling that $r \leq d(\alpha_{curr}, \alpha_{\min})$ and $\tilde{d}(\alpha_{\min}, \tau_{curr}) \leq d(\alpha_{\min}, \tau_{curr})$ we obtain the required inequality.

Target's turn. If $\tilde{d}(\alpha_{curr}, \tau_{curr}) \geq \tilde{d}(\alpha_{curr}, \tau_{next}) - r$, then, according to Line 3.6 of the algorithm, the value of the estimated distance $\tilde{d}(\alpha_{curr}, \tau_{curr})$ is not changed by the updating and the admissibility is preserved.

Let $\tilde{d}(\alpha_{curr}, \tau_{curr}) < \tilde{d}(\alpha_{curr}, \tau_{next}) - r$. Then it is required to demonstrate that

$$\tilde{d}(\alpha_{curr}, \tau_{next}) - r \leq d(\alpha_{curr}, \tau_{curr}).$$

As above, the triangle inequality implies that $d(\alpha_{curr}, \tau_{curr}) + d(\tau_{curr}, \tau_{next}) \geq d(\alpha_{curr}, \tau_{next})$. Since $d(\tau_{curr}, \tau_{next}) < r$ and $d(\alpha_{curr}, \tau_{next}) \geq \tilde{d}(\alpha_{curr}, \tau_{next})$, it follows that

$$d(\alpha_{curr}, \tau_{curr}) + r \geq d(\alpha_{curr}, \tau_{next}) \geq \tilde{d}(\alpha_{curr}, \tau_{next}),$$

which is the required inequality. ■

Now we can prove that, similar to the MTS algorithm (Algorithm 2.14), the IMTS algorithm (Algorithm 4.5) always terminates.

Theorem 4.9 [21]. (in parallel to Theorem 2.14). If for every available pair $(\alpha, \tau) \in \chi \times \chi$ of the searcher's partition α and target's partition τ it follows that $\tilde{d}_0(\alpha, \tau) \leq d(\alpha, \tau)$, and for all searcher's partitions $\alpha, \alpha' \in \chi$ and target's partitions $\tau, \tau' \in \chi$ it holds true that $d(\tau, \tau') - \tilde{d}(\tau, \tau') < d(\alpha, \alpha') - \tilde{d}(\alpha, \alpha')$ for the searcher's and target's partitions, then Algorithm 4.5 (IMTS algorithm) always terminates.

Proof. Let the searcher's partition be α_{curr} and the target's partition be τ_{curr} , and assume that the searcher moves to partition $\alpha_{next} \in N(\alpha_{curr}, r)$ and the target moves to partition τ_{next} . We need to show that after updating it is guaranteed that

$$d(\alpha_{next}, \tau_{next}) - \tilde{d}(\alpha_{next}, \tau_{next}) \leq d(\alpha_{curr}, \tau_{next}) - \tilde{d}(\alpha_{curr}, \tau_{next}).$$

Since the target was not found in the previous step, it follows that $\tau_{curr} \notin N(\alpha_{curr}, r)$, $\tilde{d}(\alpha_{curr}, \tau_{curr}) > r$, and $d(\alpha_{curr}, \tau_{curr}) > r$.

If $\tau_{next} \in N(\alpha_{curr}, r)$, then $d(\alpha_{next}, \tau_{next}) = 0$. Hence, from the metric properties of \tilde{d} it follows that $\alpha_{next} = \tau_{next}$ and the algorithm terminates.

Let $\tau_{next} \notin N(\alpha_{curr}, r)$. Then $\tilde{d}(\alpha_{curr}, \tau_{next}) > r$ and $d(\alpha_{curr}, \tau_{next}) > r$. For any pair $(\xi, \zeta) \in \chi \times \chi$ of partitions, denote by $\tilde{d}'(\xi, \zeta)$ the values of estimated distances that are obtained after updating. Let us consider the actions of the algorithm.

Assume that in Line 3.2 the searcher chooses partition α_{next} as the next partition and the value of the estimated distance at the searcher's turn is updated to the value (see Line 3.3)

$$\tilde{d}'(\alpha_{curr}, \tau_{curr}) \leftarrow \max\{\tilde{d}(\alpha_{curr}, \tau_{curr}), \min_{\alpha \in N(\alpha_{curr})}\{\tilde{d}(\alpha, \tau_{curr}) + r\}\}.$$

After the searcher's movement from the current partition α_{curr} to the next partition α_{next} , this value is preserved as $\tilde{d}'(\alpha_{next}, \tau_{curr})$. Then, the estimation updating at the target's movement is as follows (see Line 3.6):

$$\tilde{d}(\alpha_{curr}, \tau_{next}) \leftarrow \max\{\tilde{d}'(\alpha_{next}, \tau_{curr}), \tilde{d}(\alpha_{curr}, \tau_{next}) - r\}.$$

There are four possible cases of updating:

1. $\tilde{d}'(\alpha_{next}, \tau_{curr}) = \tilde{d}(\alpha_{curr}, \tau_{curr})$ and $\tilde{d}'(\alpha_{next}, \tau_{next}) = \tilde{d}'(\alpha_{next}, \tau_{curr})$.

Following these equalities and applying the triangle inequality for actual and estimated distances, one obtains

$$\begin{aligned} \tilde{d}'(\alpha_{next}, \tau_{curr}) &\leq \tilde{d}(\alpha_{curr}, \alpha_{next}) + \tilde{d}(\alpha_{next}, \tau_{curr}), \\ \tilde{d}'(\alpha_{next}, \tau_{next}) &\leq \tilde{d}(\tau_{curr}, \tau_{next}) + \tilde{d}(\alpha_{next}, \tau_{curr}), \\ d(\alpha_{curr}, \tau_{curr}) &\leq d(\alpha_{curr}, \alpha_{next}) + d(\alpha_{next}, \tau_{curr}), \\ d(\alpha_{next}, \tau_{next}) &\leq d(\tau_{curr}, \tau_{next}) + d(\alpha_{next}, \tau_{curr}). \end{aligned}$$

Subtraction of the first inequalities from the second ones results in the following:

$$\begin{aligned} d(\alpha_{curr}, \tau_{curr}) - \tilde{d}'(\alpha_{next}, \tau_{curr}) &\leq d(\alpha_{curr}, \alpha_{next}) - \tilde{d}(\alpha_{curr}, \alpha_{next}) \\ &\quad + d(\alpha_{next}, \tau_{curr}) - \tilde{d}(\alpha_{next}, \tau_{curr}), \\ d(\alpha_{next}, \tau_{next}) - \tilde{d}'(\alpha_{next}, \tau_{next}) &\leq d(\tau_{curr}, \tau_{next}) - \tilde{d}(\tau_{curr}, \tau_{next}) \\ &\quad + d(\alpha_{next}, \tau_{curr}) - \tilde{d}(\alpha_{next}, \tau_{curr}). \end{aligned}$$

Thus, an inequality $d(\alpha_{next}, \tau_{next}) - \tilde{d}(\alpha_{next}, \tau_{next}) < d(\alpha_{curr}, \tau_{curr}) - \tilde{d}(\alpha_{curr}, \tau_{curr})$ is guaranteed, if it follows that

$$d(\tau_{curr}, \tau_{next}) - \tilde{d}(\tau_{curr}, \tau_{next}) < d(\alpha_{curr}, \alpha_{next}) - \tilde{d}(\alpha_{curr}, \alpha_{next}),$$

which is exactly the assumption of the theorem.

2. $\tilde{d}'(\alpha_{next}, \tau_{curr}) = \tilde{d}(\alpha_{curr}, \tau_{curr})$ and $\tilde{d}'(\alpha_{next}, \tau_{next}) = \tilde{d}(\alpha_{next}, \tau_{next}) - r$.

Since $d(\alpha_{next}, \tau_{next}) \leq d(\tau_{curr}, \tau_{next}) + \tilde{d}(\alpha_{next}, \tau_{curr})$, for the estimated distance $\tilde{d}'(\alpha_{next}, \tau_{next}) = \tilde{d}(\alpha_{next}, \tau_{next}) - r$ it also holds true that

$$\tilde{d}'(\alpha_{next}, \tau_{next}) \leq \tilde{d}(\tau_{curr}, \tau_{next}) + \tilde{d}(\alpha_{next}, \tau_{curr}).$$

Thus, this case is the same as the previous one.

3. $\tilde{d}'(\alpha_{next}, \tau_{curr}) = \tilde{d}(\alpha_{next}, \tau_{curr}) + r$ and $\tilde{d}'(\alpha_{next}, \tau_{next}) = \tilde{d}'(\alpha_{next}, \tau_{curr})$.

According to Lemma 4.10, the IMTS algorithm preserves the admissibility of estimations. Hence $\tilde{d}'(\alpha_{next}, \tau_{curr}) = \tilde{d}(\alpha_{next}, \tau_{curr}) + r \leq d(\alpha_{next}, \tau_{curr})$.

Since $\tilde{d}(\alpha_{curr}, \alpha_{next}) \leq r$ and

$$\tilde{d}'(\alpha_{next}, \tau_{curr}) = \tilde{d}(\alpha_{next}, \tau_{curr}) + r \leq \tilde{d}(\alpha_{curr}, \alpha_{next}) + \tilde{d}(\alpha_{next}, \tau_{curr}),$$

one obtains

$$\tilde{d}(\alpha_{curr}, \alpha_{next}) = r,$$

which is the case when α_{next} lies on the boundary of the neighborhood $N(\alpha_{curr}, r)$, for which the reasons of case 1 are still true.

4. $\tilde{d}'(\alpha_{next}, \tau_{curr}) = \tilde{d}(\alpha_{next}, \tau_{curr}) + r$ and $\tilde{d}'(\alpha_{next}, \tau_{next}) = \tilde{d}(\alpha_{next}, \tau_{next}) - r$.

This case is a combination of the situations considered in cases 2 and 3. ■

The properties formulated above show that the IMTS algorithm (Algorithm 4.5) has similar properties to the MTS algorithm (Algorithm 2.14). However, note that these properties are obtained here by using weaker requirements than those used in the MTS algorithm. Furthermore, as in the ILRTA* algorithm (Algorithm 4.2) of search for a static target, the IMTS algorithm allows the determination of different probability measures on the sample space X .

The implementation of the IMTS algorithm in the form of group-testing search procedures is similar to the above implementation of the ILRTA* algorithm in the Huffman–Zimmerman procedure (Algorithm 4.2) and the GOTA (Algorithm 4.3). As an additional illustration of the algorithm's properties, in the next section we consider the IMTS algorithm under the assumptions and requirements of the known Pollock model of search [42] (see Section 3.2.2). Such considerations allow us to present a complete running example of the algorithm's actions.

4.3.2 Simple search using the IMTS algorithm

Let us implement the IMTS algorithm (Algorithm 4.5) for the simple search that is defined by the Pollock model of search for a moving target [42], which was considered in

Section 3.2.2. Recall that the Pollock model of search considers a search for a target that moves between two boxes according to a given Markov process, and obtains results with regard to both perfect and imperfect detection. In the Pollock model, the sample space includes two points $X = \{x_1, x_2\}$, for which location probabilities $p_1^t = p^t(x_1)$ and $p_2^t = p^t(x_2)$ at times $t = 0, 1, 2, \dots$ are defined. Similar to the general definition of the search problem (see Section 2.1.1), it is assumed that the target moves according to a definite Markov process governed by the transition probability matrix $\rho = \|\rho_{ij}\|_{2 \times 2}$, $i, j = 1, 2$; so, given initial location probabilities $\vec{p}^0 = (p_1^0, p_2^0)$, location probabilities at the next moments are defined by the usual multiplication $\vec{p}^{t+1} = \vec{p}^t \times \rho$. Regarding the detection probabilities,

$$\psi_i^t = \psi(x_i, t) = \Pr\{\text{target is detected at moment } t | \text{target is located at } x_i \text{ at moment } t\},$$

while the searcher observes the corresponding point x_i , $i = 1, 2$, it is assumed that they do not depend on t and the corresponding index is omitted. In the case of *perfect detection*, it is specified that $\psi_1 = \psi_2 = 1$, and in the case of *imperfect detection* the probabilities obtain arbitrary values in the range $0 < \psi_1, \psi_2 \leq 1$.

Now let us implement the IMTS algorithm (Algorithm 4.5) in the Pollock model of search (see Section 3.2.2). In the IMTS algorithm, metrics \tilde{d} and d do not distinguish the partitions which correspond to the target's location at the point x_1 or x_2 . In other words, for the sample space $X = \{x_1, x_2\}$ and perfect detection with detection probabilities $\psi_1 = \psi_2 = 1$, if the searcher observes point x_1 and does not detect the target, then the searcher knows that the target is at the point x_2 . Hence, direct implementation of the IMTS algorithm results in its termination after the first step. In the Pollock model, in contrast, if the searcher observes the box x_1 and does not detect the target then the target moves and the search continues up to the moment when the searcher detects the target.

To introduce the Pollock search model into the framework of the IMTS algorithm, let us consider a *fictive point* x_3 , which is included in the sample space X . Note that the idea of a fictive point in the context of the Pollock model was suggested by Singh and Krishnamurthy [43]; however, the meaning of fictive point below is rather different.

Given the two-point Pollock model of search with location probabilities

$$p_1^{t=0} + p_2^{t=0} = 1$$

at time $t = 0$ and transition probabilities

$$\rho_{11} + \rho_{12} = \rho_{21} + \rho_{22} = 1, \quad \rho_{12} > 0, \quad \rho_{21} > 0,$$

let us consider the sample space $X = \{x_1, x_2, x_3\}$ with a fictive point x_3 .

The location probabilities $p_i^t = p^t(x_i)$, $i = 1, 2, 3$, $t = 0, 1, 2, \dots$, for the sample space $X = \{x_1, x_2, x_3\}$ with the fictive point x_3 are defined as follows:

$$\begin{aligned} p^t(x_1) &= p^t(x_1) - \varepsilon_1, & p^t(x_2) &= p^t(x_2) - \varepsilon_2, & p^t(x_3) &= \varepsilon_1 + \varepsilon_2, \\ 0 < \varepsilon_1, \varepsilon_2 &< p^t(x_1), & 0 < \varepsilon_1, \varepsilon_2 &< p^t(x_2), \end{aligned}$$

and the transition probabilities for the fictive box x_3 are

$$\rho_{13} = \rho_{23} = \rho_{31} = \rho_{32} = 0 \quad \text{and} \quad \rho_{33} = 1.$$

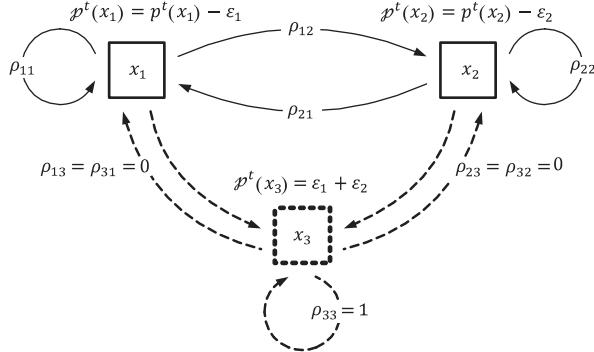


Figure 4.7 Markov process for the Pollock model of search with a fictive point.

The Markov process, which governs the target's movement over the sample space $X = \{x_1, x_2, x_3\}$ with the fictive point x_3 and location probabilities $p^t(x_i)$, $i = 1, 2, 3$, is illustrated in Figure 4.7.

Note that in terms of the IMTS algorithm, the fictive point x_3 is required to distinguish the choices of points x_1 and x_2 , while in the Pollock model of search the fictive point models an imperfect detection case. The next lemma presents the relation between the fictive point and the detection uncertainty.

Lemma 4.11 [21]. *A search by the IMTS algorithm (Algorithm 4.5) over the sample space $X = \{x_1, x_2, x_3\}$ with a fictive point x_3 is equivalent to the Pollock search procedure with imperfect detection and the following detection probabilities $\psi_1 = 1 - \epsilon_1/p_1^0$ and $\psi_2 = 1 - \epsilon_2/p_2^0$, where $0 < \epsilon_1, \epsilon_2 < p_1^0$ and $0 < \epsilon_1, \epsilon_2 < p_2^0$.*

Proof. In the proof we apply Equation 3.20 of probability updating in the Pollock model of search (see Section 3.2.2).

Denote $\psi_1 p_1^0 = p_1^0 - \epsilon_1$, and consider the expected probability

$$\tilde{p}_1^1(z(\{x_1\}, x^0) = 0) = \frac{p_1^0(1 - \psi_1)\rho_{11} + p_2^0\rho_{21}}{1 - \psi_1 p_1^0},$$

where $z(\{x_1\}, x^0)$ stands for the result of observation of the point x_1 while the target is located at the point $x^0 \in \{x_1, x_2\}$. Substitution of $p_1^0 = p_1^0 - \epsilon_1$ instead of $\psi_1 p_1^0$ gives

$$\tilde{p}_1^1(z(\{x_1\}, x^0) = 0) = \frac{p_1^0\rho_{11} - (p_1^0 - \epsilon_1)\rho_{11} + p_2^0\rho_{21}}{1 - (p_1^0 - \epsilon_1)} = \frac{p_2^0\rho_{21} + \epsilon_1\rho_{11}}{1 - p_1^0 + \epsilon_1}.$$

Rewrite $\psi_1 = 1 - \delta_1$. Then, from the first equality, one obtains

$$\tilde{p}_1^1(z(\{x_1\}, x^0) = 0) = \frac{p_1^0\rho_{11} - \psi_1 p_1^0\rho_{11} + p_2^0\rho_{21}}{1 - \psi_1 p_1^0} = \frac{p_2^0\rho_{21} + \delta_1 p_1^0\rho_{11}}{1 - p_1^0 + \delta_1 p_1^0}.$$

Comparison of this equality to the previous one gives $\epsilon_1 = \delta_1 p_1^0 = (1 - \psi_1)p_1^0$, and finally $\psi_1 = 1 - (\epsilon_1/p_1^0)$.

The equality $\psi_1 = 1 - (\varepsilon_1/p_2^0)$ follows from the same reasons applied to the expected probability $\tilde{p}_2^1(z(\{x_2\}, x^0) = 0)$. ■

Let us consider the Rokhlin estimated distances \tilde{d} between the partitions applied to the Pollock model of search. These distances will be applied in the example below.

In the Pollock model of search the target chooses one of the points x_1 or x_2 . Thus, for the sample space $X = \{x_1, x_2, x_3\}$ with the fictive point x_3 , the partitions available to the target are

$$\tau_1 = \{\{x_1\}, \{x_2, x_3\}\} \quad \text{and} \quad \tau_2 = \{\{x_2\}, \{x_1, x_3\}\}.$$

The partitions that are available to the searcher include the same partitions

$$\alpha_1 = \{\{x_1\}, \{x_2, x_3\}\} \quad \text{and} \quad \alpha_2 = \{\{x_2\}, \{x_1, x_3\}\},$$

and additional an initial partition θ and a partition α_3 that corresponds to the fictive point

$$\theta = \{\{x_1, x_2, x_3\}, \emptyset\} \quad \text{and} \quad \alpha_3 = \{\{x_3\}, \{x_1, x_2\}\}.$$

Let us calculate the initial distance estimations between the available partitions. Let, as above, $p_i^0 = p^0(x_i)$, $i = 1, 2, 3$, be location probabilities over $X = \{x_1, x_2, x_3\}$ at time $t = 0$, such that $p_1^0 = p_1^0 - \varepsilon_1$, $p_2^0 = p_2^0 - \varepsilon_2$, and $p_3^0 = \varepsilon_1 + \varepsilon_2$. For convenience, denote $\xi_1 = \alpha_1 = \tau_1$, $\xi_2 = \alpha_2 = \tau_2$ and $\xi_3 = \alpha_3$. Then, the Rokhlin estimated distances \tilde{d}_0 between the partitions at time $t = 0$ are defined as follows:

$$\begin{aligned} \tilde{d}_0(\theta, \xi_1) &= -p_1^0 \log p_1^0 - (p_2^0 + p_3^0) \log(p_2^0 + p_3^0), \\ \tilde{d}_0(\theta, \xi_2) &= -p_2^0 \log p_2^0 - (p_1^0 + p_3^0) \log(p_1^0 + p_3^0), \\ \tilde{d}_0(\theta, \xi_3) &= -p_3^0 \log p_3^0 - (p_1^0 + p_2^0) \log(p_1^0 + p_2^0), \\ \tilde{d}_0(\xi_1, \xi_2) &= -p_1^0 \log \frac{p_1^0}{p_1^0 + p_3^0} - p_3^0 \log \frac{p_3^0}{p_1^0 + p_3^0} \\ &\quad - p_2^0 \log \frac{p_2^0}{p_2^0 + p_3^0} - p_3^0 \log \frac{p_3^0}{p_2^0 + p_3^0}, \\ \tilde{d}_0(\xi_1, \xi_3) &= -p_1^0 \log \frac{p_1^0}{p_1^0 + p_2^0} - p_2^0 \log \frac{p_2^0}{p_1^0 + p_2^0} \\ &\quad - p_3^0 \log \frac{p_3^0}{p_2^0 + p_3^0} - p_2^0 \log \frac{p_2^0}{p_2^0 + p_3^0}, \\ \tilde{d}_0(\xi_2, \xi_3) &= -p_2^0 \log \frac{p_2^0}{p_1^0 + p_2^0} - p_1^0 \log \frac{p_1^0}{p_1^0 + p_2^0} \\ &\quad - p_3^0 \log \frac{p_3^0}{p_1^0 + p_3^0} - p_1^0 \log \frac{p_1^0}{p_1^0 + p_3^0}. \end{aligned}$$

The properties of the distance estimations follow directly from the given equalities. Below, we formulate them in the form of the next lemma.

Lemma 4.12 [21]. If for ε_1 and ε_2 it follows that $0 < \varepsilon_1, \varepsilon_2 < p_1^0$ and $0 < \varepsilon_1, \varepsilon_2 < p_2^0$, then the following inequalities hold:

1. $\tilde{d}_0(\theta, \xi_1) < \tilde{d}_0(\xi_1, \xi_3)$.
2. $\tilde{d}_0(\theta, \xi_2) < \tilde{d}_0(\xi_2, \xi_3)$.
3. If $\varepsilon_1 + \varepsilon_2 < p_1^0$, then $\tilde{d}_0(\theta, \xi_3) < \tilde{d}_0(\theta, \xi_1)$.
4. If $\varepsilon_1 + \varepsilon_2 < p_2^0$, then $\tilde{d}_0(\theta, \xi_3) < \tilde{d}_0(\theta, \xi_2)$.

Proof. By substituting the formulas for the distances $\tilde{d}_0(\theta, \xi_3)$, $\tilde{d}_0(\xi_1, \xi_3)$, and $\tilde{d}_0(\xi_2, \xi_3)$ into inequalities 1 and 2, one obtains

1. $2\mathcal{P}_2^0 \log \mathcal{P}_2^0 + \mathcal{P}_3^0 \log \mathcal{P}_3^0 < 2(\mathcal{P}_2^0 + \mathcal{P}_3^0) \log(\mathcal{P}_2^0 + \mathcal{P}_3^0) + (\mathcal{P}_1^0 + \mathcal{P}_2^0) \log(\mathcal{P}_1^0 + \mathcal{P}_2^0)$
2. $2\mathcal{P}_1^0 \log \mathcal{P}_1^0 + \mathcal{P}_3^0 \log \mathcal{P}_3^0 < 2(\mathcal{P}_1^0 + \mathcal{P}_3^0) \log(\mathcal{P}_1^0 + \mathcal{P}_3^0) + (\mathcal{P}_1^0 + \mathcal{P}_2^0) \log(\mathcal{P}_1^0 + \mathcal{P}_2^0)$

Comparison of the addendums on the left and the right sides of the inequalities gives the required statement.

To prove inequalities 3 and 4, recall that $\mathcal{P}_1^0 = \mathcal{P}_1^0 - \varepsilon_1$, $\mathcal{P}_2^0 = \mathcal{P}_2^0 - \varepsilon_2$, and $\mathcal{P}_3^0 = \varepsilon_1 + \varepsilon_2$. Then the required inequalities obtain the following form:

3. $-(\varepsilon_1 + \varepsilon_2) \log(\varepsilon_1 + \varepsilon_2) - (1 - \varepsilon_1 + \varepsilon_2) \log(1 - \varepsilon_1 + \varepsilon_2) < -(\mathcal{P}_1^0 - \varepsilon_1) \log(\mathcal{P}_1^0 - \varepsilon_1) - (1 - \mathcal{P}_1^0 + \varepsilon_1) \log(1 - \mathcal{P}_1^0 + \varepsilon_1)$,
4. $-(\varepsilon_1 + \varepsilon_2) \log(\varepsilon_1 + \varepsilon_2) - (1 - \varepsilon_1 + \varepsilon_2) \log(1 - \varepsilon_1 + \varepsilon_2) < -(\mathcal{P}_2^0 - \varepsilon_2) \log(\mathcal{P}_2^0 - \varepsilon_2) - (1 - \mathcal{P}_2^0 + \varepsilon_2) \log(1 - \mathcal{P}_2^0 + \varepsilon_2)$.

By using the requirements that $\varepsilon_1 + \varepsilon_2 < p_1^0$ and $\varepsilon_1 + \varepsilon_2 < p_2^0$ for inequalities 3 and 4, respectively, one obtains that in inequality 3 it follows that $\varepsilon_1 + \varepsilon_2 < \mathcal{P}_1^0 - \varepsilon_1$, and in inequality 4 it follows that $\varepsilon_1 + \varepsilon_2 < \mathcal{P}_2^0 - \varepsilon_2$. Thus, inequalities 3 and 4 hold. ■

From Lemma 4.12 it follows that the next statement holds.

Corollary 4.1 [21]. If $\varepsilon_1 + \varepsilon_2 < p_1^0$, then $\tilde{d}_0(\theta, \xi_3) < \tilde{d}_0(\xi_1, \xi_3)$, and if $\varepsilon_1 + \varepsilon_2 < p_2^0$, then $\tilde{d}_0(\theta, \xi_3) < \tilde{d}_0(\xi_2, \xi_3)$.

Remark 4.1 [21]. Requirements 3 and 4 of Lemma 4.12 that correspondingly result in the inequalities $\tilde{d}_0(\theta, \xi_3) < \tilde{d}_0(\xi_1, \xi_3)$ and $\tilde{d}_0(\theta, \xi_3) < \tilde{d}_0(\xi_2, \xi_3)$ given by Corollary 4.1 are not unique. In particular, it is true that:

3. If $p_1^0 - \varepsilon_1 < p_2^0 + \varepsilon_1$, then $\tilde{d}_0(\theta, \xi_3) < \tilde{d}_0(\xi_1, \xi_3)$.
4. If $p_2^0 - \varepsilon_2 < p_1^0 + \varepsilon_2$, then $\tilde{d}_0(\theta, \xi_3) < \tilde{d}_0(\xi_2, \xi_3)$.

Proof. Consider inequality 3. After the transformations, one obtains

$$2\mathcal{P}_2^0 \log \mathcal{P}_2^0 + \mathcal{P}_1^0 \log \mathcal{P}_1^0 < 2(\mathcal{P}_1^0 + \mathcal{P}_2^0) \log(\mathcal{P}_1^0 + \mathcal{P}_2^0) + (\mathcal{P}_2^0 + \mathcal{P}_3^0) \log(\mathcal{P}_2^0 + \mathcal{P}_3^0).$$

It is seen that for the first addendums on the left and right sides of the inequality it follows that $\mathcal{P}_2^0 \log \mathcal{P}_2^0 < (\mathcal{P}_1^0 + \mathcal{P}_2^0) \log(\mathcal{P}_1^0 + \mathcal{P}_2^0)$ for every positive ε_1 and ε_2 . Thus, we need to consider the second addendums.

Recalling that $\mathcal{P}_1^0 = p_1^0 - \varepsilon_1$, $\mathcal{P}_2^0 = p_2^0 - \varepsilon_2$, and $\mathcal{P}_3^0 = \varepsilon_1 + \varepsilon_2$, one obtains

$$\mathcal{P}_2^0 + \mathcal{P}_3^0 = p_2^0 - \varepsilon_2 + \varepsilon_2 = p_2^0 + \varepsilon_1.$$

Hence, under the requirements of inequality 3 it follows that $\mathcal{P}_1^0 < \mathcal{P}_2^0 + \mathcal{P}_3^0$, and for the second addendums on the left and right sides of the required inequality 3 it also holds that $\mathcal{P}_1^0 \log \mathcal{P}_1^0 < 2(\mathcal{P}_2^0 + \mathcal{P}_3^0) \log(\mathcal{P}_2^0 + \mathcal{P}_3^0)$.

For inequality 4 the reasons are similar. ■

Lemma 4.13 [21]. *If $\varepsilon_1 + \varepsilon_2 < p_1^0$ and $\varepsilon_1 + \varepsilon_2 < p_2^0$, then $\tilde{d}_0(\xi_1, \xi_2) < \tilde{d}_0(\xi_1, \xi_3)$ and $\tilde{d}_0(\xi_1, \xi_2) < \tilde{d}_0(\xi_2, \xi_3)$.*

Proof. Consider the inequality $\tilde{d}_0(\xi_1, \xi_2) < \tilde{d}_0(\xi_1, \xi_3)$. After the transformations, one obtains

$$\begin{aligned} -\mathcal{P}_3^0 \log \mathcal{P}_3^0 - (1 - \mathcal{P}_3^0) \log(1 - \mathcal{P}_3^0) \\ < -(\mathcal{P}_1^0 + \mathcal{P}_3^0) \log(\mathcal{P}_1^0 + \mathcal{P}_3^0) - \mathcal{P}_2^0 \log \mathcal{P}_2^0. \end{aligned}$$

Recall that $\mathcal{P}_1^0 = p_1^0 - \varepsilon_1$, $\mathcal{P}_2^0 = p_2^0 - \varepsilon_2$, and $\mathcal{P}_3^0 = \varepsilon_1 + \varepsilon_2$. Then

$$\begin{aligned} -(\varepsilon_1 + \varepsilon_2) \log(\varepsilon_1 + \varepsilon_2) - (1 - \varepsilon_1 + \varepsilon_2) \log(1 - \varepsilon_1 + \varepsilon_2) \\ < -(\mathcal{P}_1^0 + \varepsilon_1) \log(\mathcal{P}_1^0 + \varepsilon_1) - (\mathcal{P}_2^0 - \varepsilon_2) \log(\mathcal{P}_2^0 - \varepsilon_2). \end{aligned}$$

For the second inequality $\tilde{d}_0(\xi_1, \xi_2) < \tilde{d}_0(\xi_2, \xi_3)$, in a similar manner, one obtains

$$\begin{aligned} -\mathcal{P}_3^0 \log \mathcal{P}_3^0 - (1 - \mathcal{P}_3^0) \log(1 - \mathcal{P}_3^0) \\ < -(\mathcal{P}_2^0 + \mathcal{P}_3^0) \log(\mathcal{P}_2^0 + \mathcal{P}_3^0) - \mathcal{P}_1^0 \log \mathcal{P}_1^0 \end{aligned}$$

and

$$\begin{aligned} -(\varepsilon_1 + \varepsilon_2) \log(\varepsilon_1 + \varepsilon_2) - (1 - \varepsilon_1 + \varepsilon_2) \log(1 - \varepsilon_1 + \varepsilon_2), \\ < -(\mathcal{P}_2^0 + \varepsilon_2) \log(\mathcal{P}_2^0 + \varepsilon_2) - (\mathcal{P}_1^0 - \varepsilon_1) \log(\mathcal{P}_1^0 - \varepsilon_1). \end{aligned}$$

Now the inequalities follow directly from the conditions of the lemma. ■

This lemma completes our consideration of applying the IMTS algorithm (Algorithm 4.5) to the Pollock model of search. From these considerations and the optimality of Pollock's procedure, we immediately obtain the following result.

Theorem 4.10 [21]. *If $\varepsilon_1 \leq p_1^0(1 - \psi_1)$ and $\varepsilon_2 \leq p_2^0(1 - \psi_2)$, then the expected number of checks required by the IMTS algorithm (Algorithm 4.5) is bounded as in the Pollock model.*

Proof. The proof of the theorem follows from the definition of the detection probabilities $\psi_1 = 1 - (\varepsilon_1/p_1^0)$ and $\psi_2 = 1 - (\varepsilon_2/p_2^0)$ and their relation with the Rokhlin distances as proven in the lemmas above. ■

From the theorem, it follows that, as indicated above, for the perfect detection when $\psi_1 = \psi_2 = 1$ the expected number of checks required by the IMTS algorithm reaches its equivalent unit lower bound as expected in the Pollock model (see Section 3.2.2).

The presented lemmas and the actions of the IMTS algorithm are illustrated by the following example.

Example 4.4 [21]. Let, as above, $X = \{x_1, x_2, x_3\}$ be the sample space with the fictive point x_3 , so that the location probabilities are defined as $p^t(x_1) = p^t(x_1) - \varepsilon_1$, $p^t(x_2) = p^t(x_2) - \varepsilon_2$, $p^t(x_3) = \varepsilon_1 + \varepsilon_2$, while $0 < \varepsilon_1, \varepsilon_2 < p^t(x_1)$, $0 < \varepsilon_1, \varepsilon_2 < p^t(x_2)$, and for transition probabilities it follows that $\rho_{13} = \rho_{23} = \rho_{31} = \rho_{32} = 0$ and $\rho_{33} = 1$. Following the above given notation, we implement the following partitions of the sample space X :

$$\theta = \{\{x_1, x_2, x_3\}, \emptyset\}, \quad \xi_1 = \{\{x_1\}, \{x_2, x_3\}\}, \quad \xi_2 = \{\{x_2\}, \{x_1, x_3\}\}, \quad \xi_3 = \{\{x_3\}, \{x_1, x_2\}\}.$$

Assume that the requirements of Lemma 4.12 or Remark 4.1 and of Lemma 4.13 are satisfied. Thus, for the estimated distances the following inequalities hold:

$$\begin{aligned} \tilde{d}_0(\theta, \xi_1) &< \tilde{d}_0(\xi_1, \xi_3), & \tilde{d}_0(\theta, \xi_2) &< \tilde{d}_0(\xi_2, \xi_3), \\ \tilde{d}_0(\theta, \xi_3) &< \tilde{d}_0(\theta, \xi_1), & \tilde{d}_0(\theta, \xi_3) &< \tilde{d}_0(\theta, \xi_2). \end{aligned}$$

Then, it follows that

$$\tilde{d}_0(\theta, \xi_3) < \tilde{d}_0(\theta, \xi_1) < \tilde{d}_0(\xi_1, \xi_3) \quad \text{and} \quad \tilde{d}_0(\theta, \xi_3) < \tilde{d}_0(\theta, \xi_2) < \tilde{d}_0(\xi_2, \xi_3).$$

Partitions of the sample space and estimated distances are shown in Figure 4.8a.

According to the presented inequalities for the estimated distances, we assume that the radius r , which defines the neighborhood, meets the following requirement:

$$r > \tilde{d}_0(\xi_1, \xi_2).$$

Note that the estimated distance $\tilde{d}_0(\xi_1, \xi_2)$ itself does not appear in the inequalities above.

Let us consider the beginning of the search process by the IMTS algorithm (Algorithm 4.5) over the defined partitions and estimated distances.

Step 1. The searcher starts with partition θ (Line 1):

$$\alpha_{curr} \leftarrow \theta = \{\{x_1, x_2, x_3\}, \emptyset\}.$$

This step implies that the searcher checks all three points. If there is no target in one of the points, then the search is meaningless and algorithm terminates.

Step 2. Since the searcher is not being informed about the possible target's locations, the searcher follows the Line 3.1 and the Line 3.2:

$$\alpha_{next} \leftarrow \xi_3 = \operatorname{argmin}\{\tilde{d}_0(\theta, \xi_3) + r, \tilde{d}_0(\theta, \xi_1) + r, \tilde{d}_0(\xi_1, \xi_3) + r\}.$$

and chooses fictive partition ξ_3 for formal observation of the fictive point x_3 .

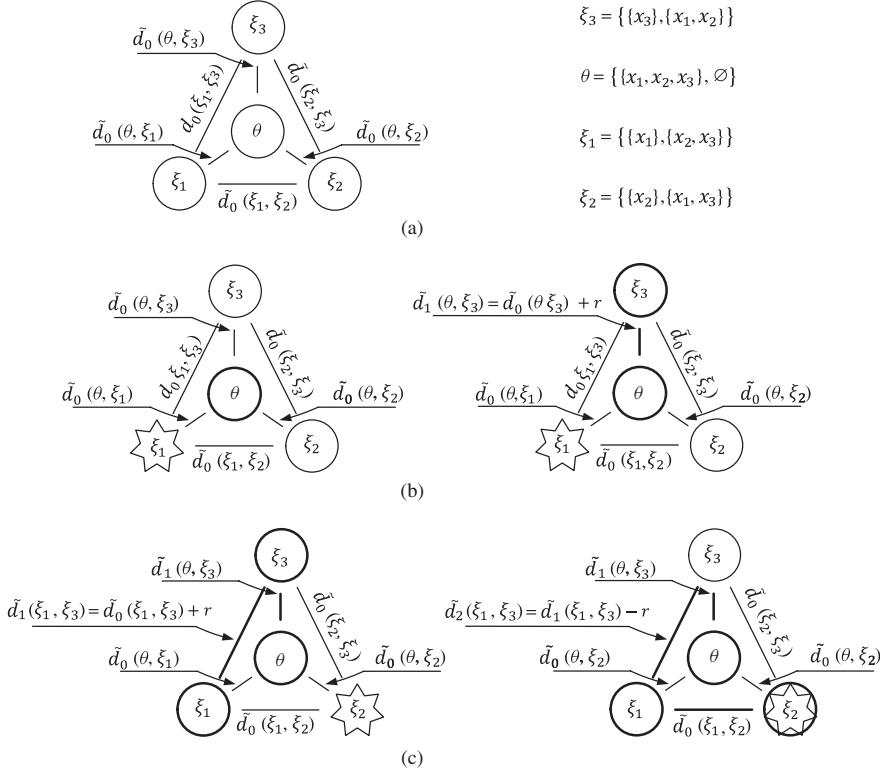


Figure 4.8 Example of the IMTS algorithm's actions for the Pollock model of search with a fictive point. (a) Partitions of the sample space and estimated distances. (b) Start with partition θ : Step 1. (c) Choice of fictive partition ξ_3 : Steps 2–4. (d) Choice of fictive partition ξ_1 : Steps 5–7. (e) Choice of fictive partition ξ_2 and stop: Steps 8–9.

Step 3. The estimated distance $\tilde{d}_0(\theta, \xi_3)$ is updated as follows (Line 3.3):

$$\tilde{d}_1(\theta, \xi_3) \leftarrow \tilde{d}_0(\theta, \xi_3) + r = \max\{\tilde{d}_0(\theta, \xi_3), \tilde{d}_0(\theta, \xi_3) + r\}.$$

and the searcher moves to the partition ξ_3 . Since partition ξ_3 is not available to the target, the searcher does not detect the target in x_3 , and the search continues with the partition $\alpha_{curr} \leftarrow \alpha_{next} = \xi_3$.

Note that the above-presented three steps are formal. However, by the consideration of these steps allows a strict definition of the radius r . After the Step 3, the searcher is with the partition ξ_3 and has to move either back to the initial partition θ or to one of the partitions ξ_1 and ξ_2 . It is clear that the movement to the initial partition θ is meaningless. Thus, radius r has to guarantee that after the updating of the estimated distance in Line 3.3, partitions ξ_1 and ξ_2 will be in the neighborhood $N(\xi_3, r)$ of the partition ξ_3 , while partition θ will be out of $N(\xi_3, r)$. If radius r is defined as follows:

$$r = \max\{\tilde{d}_0(\xi_1, \xi_3), \tilde{d}_0(\xi_2, \xi_3)\},$$

then this requirement for the neighborhood holds.

The target moves between the partitions ξ_1 and ξ_2 . Thus, requirement $r > \tilde{d}_0(\xi_1, \xi_2)$ implies that $\tilde{d}_0(\xi_1, \xi_2) < \max\{\tilde{d}_0(\xi_1, \xi_3), \tilde{d}_0(\xi_2, \xi_3)\}$, what is guaranteed by the Lemma 4.13.

Assume that $\tilde{d}_0(\xi_1, \xi_3) < d_0(\xi_2, \xi_3)$ and so $r = \tilde{d}_0(\xi_2, \xi_3)$. Following this assumption, the IMTS algorithm continues the search as follows.

Step 4. At this step, there is the target's turn to move. The searcher is not informed about the target's movement, thus, the possible target's partitions are ξ_1 and ξ_2 . Given the radius $r = \tilde{d}_0(\xi_2, \xi_3)$, the neighbors of the partition ξ_1 are θ and ξ_2 , and the neighbors of the partition ξ_2 are θ and ξ_3 . Since the searcher at the previous step updated the estimated distance between the partitions θ and ξ_3 , while the estimated distances between the other partitions are remained, the updating at the Line 3.6 does not change the values of the estimated distances.

Step 5. Recall that we assumed that $\tilde{d}_0(\xi_1, \xi_3) < \tilde{d}_0(\xi_2, \xi_3)$. Hence, on the current turn the searcher chooses the partition ξ_1 :

$$\alpha_{next} \leftarrow \xi_1 = \operatorname{argmin}\{\tilde{d}_1(\theta, \xi_3) + r, \tilde{d}_0(\xi_1, \xi_3) + r, \tilde{d}_0(\xi_2, \xi_3) + r\},$$

what implies an observation of the point x_1 .

Step 6. Assume that the target was not detected in x_1 implying that the target is at point x_2 and its partition is $\tau_{curr} = \xi_2$. Then the searcher updates the estimated distance $\tilde{d}_0(\xi_1, \xi_3)$ as follows:

$$\tilde{d}_1(\xi_1, \xi_3) \leftarrow \tilde{d}_0(\xi_1, \xi_3) + r = \max\{\tilde{d}_0(\xi_1, \xi_3), \min\{\tilde{d}_0(\xi_1, \xi_3), \tilde{d}_0(\xi_2, \xi_3) + r\}\},$$

and continues with the partition $\alpha_{curr} \leftarrow \alpha_{next} = \xi_1$.

Step 7. The neighbors of possible target's partitions are the same as in the Step 4 above. However, in this step the updated estimated distance $\tilde{d}_1(\xi_1, \xi_3) > \tilde{d}_0(\theta, \xi_1) + r$. Hence, the searcher updates it as follows:

$$\tilde{d}_2(\xi_1, \xi_3) \leftarrow \tilde{d}_1(\xi_1, \xi_3) - r = \tilde{d}_0(\xi_1, \xi_3).$$

Step 8. The next step the searcher selects partition ξ_2 :

$$\alpha_{next} \leftarrow \xi_2 = \operatorname{argmin}\{\tilde{d}_0(\xi_1, \xi_2) + r, \tilde{d}_2(\xi_1, \xi_3) + r, \tilde{d}_0(\theta, \xi_1) + r\},$$

what implies an observation of the point x_2 .

Step 9. Since, according to the Step 6, the target's current partition is $\tau_{curr} = \xi_2$, this is exactly the point in which the target is located. Hence, if the target in its turn chooses to stay in this point, the searcher will detect it. Assume that this is the situation; so the search terminates and the target is found in the point x_2 .

The choices of the partitions and the estimated distances updating are shown in Figure 4.8.

Following the presented actions, Step 1 and Steps 2–4 (see Figure 4.8b,c) correspond to the starting point of the algorithm, while it chooses the initial partition and observes the complete sample space $X = \{x_1, x_2, x_3\}$, and to the choice of the partition ξ_3 , which corresponds to the fictive point x_3 . These steps are formal and are required for updating the estimated distances; in practical search they are omitted. Steps 5–7 and Steps 8–9 (see Figure 4.8d–f) correspond to the choice of the partitions ξ_1 and ξ_2 , which represent possible

target locations at the points x_1 and x_2 , respectively. In the considered case, the target stays at the point x_2 and it is found when the searcher chooses partition ξ_2 . ■

This example completes our consideration of the general properties of the IMTS algorithm. Below, we consider some properties of this algorithm using numerical simulations.

4.3.3 Dependence of the IMTS algorithm's actions on the target's movement

To finalize our consideration of the IMTS algorithm (Algorithm 4.5) let us note that the expected number of required search steps significantly depends on the type of the target's movement.

In the numerical trials, the IMTS algorithm was applied in a search for static, Markovian, and Brownian targets with different values of the radius r , which determines the searcher's expectations regarding the 'length' of the target's steps. Possible movements of the Markovian target are: 'move north,' 'move south,' 'move east,' 'move west,' and 'stay at the current point', while the Brownian target was prohibited from staying at its current position.

First, we run the IMTS algorithm according to the Pollock model of search considered above. The results of the simulations are as follows:

Parameters $r, \varepsilon_1, \varepsilon_2$		Number of steps			
		Min	Max	Mean	Standard deviation
$r = 0.1$	$\varepsilon_1 = 0.1, \varepsilon_2 = 0.1$	1	5	1.503	0.855
	$\varepsilon_1 = 0.1, \varepsilon_2 = 0.2$	1	5	1.589	0.850
	$\varepsilon_1 = 0.1, \varepsilon_2 = 0.3$	1	5	1.615	0.823
	$\varepsilon_1 = 0.2, \varepsilon_2 = 0.2$	1	5	1.535	0.773
	$\varepsilon_1 = 0.2, \varepsilon_2 = 0.3$	1	5	1.563	0.766
$r = 1.0$	$\varepsilon_1 = 0.1, \varepsilon_2 = 0.1$	1	5	1.599	0.905
	$\varepsilon_1 = 0.1, \varepsilon_2 = 0.2$	1	5	1.597	0.852
	$\varepsilon_1 = 0.1, \varepsilon_2 = 0.3$	1	6	1.616	0.837
	$\varepsilon_1 = 0.2, \varepsilon_2 = 0.2$	1	7	1.574	0.783
	$\varepsilon_1 = 0.2, \varepsilon_2 = 0.3$	1	4	1.494	0.673

The results of the simulations show that the average number of search steps for the Pollock model of search is close to the entropy $\log_2 3 = 1.585$ of an equiprobable sample space. It slightly depends on the probabilities of detection errors $\varepsilon_1, \varepsilon_2$, and on the radius r that represents the searcher's expectation about the target's next step.

The next simulations addressed a general search by the IMTS algorithm for a target governed by different movement rules. The simulations were conducted in the same manner as those of the search procedures with informational decision rules (see Section 3.1.2). As above, the sample space X includes nine points, that is, $|X| = 9$, while the searcher was

allowed to use the partitions which correspond to the single-point observed areas. The results of the simulations are as follows:

Type of target's movement	Number of steps			
	Min	Max	Mean	Standard deviation
$r = 0.1$	Static target	1	8	4.486
	Markovian target	1	26	4.875
	Brownian target	1	33	4.987
$r = 1.0$	Static target	1	8	4.521
	Markovian target	1	26	4.627
	Brownian target	1	33	5.005

Note again that at the beginning of the search an offline search procedure is used. Accordingly, a fast zoom-in process can be obtained at the beginning of the group testing by an offline procedure, and then, in the vicinity of the target, an online procedure can be used to account for new side information.

As expected, the search for a target with a greater expected length of steps requires a greater number of search steps. In addition, the results demonstrate that the expected number of search steps, which is required for the search for a Markovian target, is less than the expected number of search steps for a Brownian target. As will be demonstrated in Section 5.2.3, the last property remains for the algorithm of search and screening, which is based on estimations of the target's further states.

The informational algorithms of search for static and moving targets, namely, ILRTA* (Algorithm 4.1) and IMTS (Algorithm 4.5), act on the partitions of the sample space. Since the implementation of such algorithms requires certain programming techniques, in the next section we consider some useful examples of the code.

4.4 Remarks on programming of the ILRTA* and IMTS algorithms

In Section 4.1 it was indicated that the size of the partitions space χ is given by the Bell number, which is defined by Equations 4.5 and 4.6, and this size exponentially increases with the size of the sample space X so that for small sample spaces it reaches large values. In the next section, we present certain examples in C++ that allow implementation of the ILRTA* and IMTS algorithms as presented above for reasonable sample spaces. The implementation of the required data structures is based on the combinatorial algorithms developed by Kreher and Stinson [20]. In the next section, we will describe some additional techniques and examples.

4.4.1 Data structures

As in the definitions of the ILRTA* and IMTS algorithms (Algorithms 4.1 and 4.5), the following data is required: sample space $X = \{x_1, x_2, \dots, x_n\}$ with location probabilities $p(x_i)$, $i = 1, \dots, n$; partitions space χ , which includes partitions of the sample space X ;

and appropriate functions which specify metrics of the partitions space χ . Below, we present a brief implementation of the data. Note that all tests for data correctness and exceptions, as well as technical definitions, are removed from the code.

The elementary type, which represents points $x_i \in X$, is defined by the class `Point`, which includes two fields, an integer `index` of the point and real `probability`, and appropriate functions which operate with the fields and the type itself. The following is a fragment of the class:

```
class Point
{
private:
    unsigned int m_index;
    float m_probability;
public:
    //...
    unsigned int index() const {
        return m_index;
    }

    float probability() const {
        return m_probability;
    }
    //...
}
```

The sample space $X = \{x_1, x_2, \dots, x_n\}$ is represented by the class `SampleSpace`, which aggregates the array of points along with additional fields and the functions which manipulate the data. The following is a fragment of the class:

```
class SampleSpace
{
private:
    Point** m_ppPoints;           //array of points
    unsigned int m_nSize;         //size of sample space
    //...
    float entropy() {            //calculates Shannon entropy of the
        unsigned int i;           //sample space; see Equation 2.40
        float h, p;

        p = 0; h = 0;
        for(i = 0; i < m_nSize; i++) {
            p = m_ppPoints[i]->probability();

            if(p != 0) h += p*log2(p);
        }
        return (-h);
    }

    Point* point(const unsigned int i) const {
        return m_ppPoints[i];
    }
    //...
}
```

For convenience, in our simulations the class `SampleSpace` was defined as a singleton; however, in the description here we omit such a definition.

In addition to these types, we implemented class `Locations` that specifies the points occupied by the target. Such a class is similar to the class `SampleSpace`, but it allows correct access protection and separation of the data that is available to the searcher and to the target. The class `Locations` included, in particular, the following fields and functions:

```
class Locations
{
    friend class Target;
private:
    class Location
    {
        public:
            Point* m_pPoint;           //point
            bool m_bOccupied;         //occupied or not
    };

    Location* m_pLocations;          //array of locations
    unsigned int m_nSize;
//...
    bool is_occupied(const Point* pPoint) const {
        unsigned int i;

        i = pPoint->index();

        return m_pLocations[i].m_bOccupied;
    }

    bool set_location(const Point* pPoint) {
        unsigned int i;

        i = pPoint->index();           //obtain index of the point

        if(!is_occupied(pPoint)) {
            m_pLocations[i].m_bOccupied = true;
            return true;
        }
        else
            return false;
    }
    void clean_location(const Point* pPoint) {
        unsigned int i;

        i = pPoint->index();           //obtain index of the point

        m_pLocations[i].m_bOccupied = false;
    }
//...
public:
//...
//checks whether the set of points includes the target
    bool observe(const Set* pSet) const {
        unsigned int i;
```

```

        for(i = 0; i < m_nSize; i++) {
            if(m_pLocations[i].m_bOccupied
                && pSet->includes(m_pLocations[i].m_pPoint))
                return true;
        }
        return false;
    }
//...
}

```

Similar to the previous class, class Locations was defined as a singleton. The type Set of the argument of the function observe() is considered below.

The presented classes define the basic data types so that, after their implementation, all required elementary objects and their attributes are created. The next classes define the data structures which are used in the algorithms.

The main structure used in the algorithms is the class Set, which defines the set $A \subset X = \{x_1, x_2, \dots, x_n\}$ of points. Since the points x_i , $i = 1, \dots, n$, with corresponding probabilities $p(x_i)$ have already been created and are included in the object defined by the class SampleSpace, the class Set has to include a Boolean identifier of whether a certain point is included in the set or not. The idea of such an implementation is as follows [20].

Assume that it is required to implement a subset A of the set X of size n , where A can be equal to X , and assume that the set X is already implemented and for each point $x \in X$ its index i is known. Then to define the subset A it is sufficient to define an array Arr of Boolean variables as follows. Each index i corresponds to a certain element $Arr(j)$ in the array Arr , where it is not necessary that j be equal to i , while the value 0/1 of the element $Arr(j)$ determines whether $x_i \in A$ or not. Since in this implementation only bit values and bitwise operations are required, each subset can be represented by a short array of integers so that each bit in the array corresponds to the point $x_i \in A$. On the basis of this idea, the key functions of the class Set are implemented as follows:

```

class Set
{
private:
    unsigned int m_nSize;           //size of the set

    unsigned int m_nIndex;          //index of the set

    unsigned int *m_pnArr;          //array of integers, whose
        //bits indicate whether the point is in the set or not

    unsigned int m_nArrSize;         //size of array of integers

    const Points* m_pPoints;        //pointer to the array
        //of points that is the same as in class SampleSpace
//...

//create an array of integers of the appropriate length
void create(unsigned int n = POINTS_NUMBER)
{
    unsigned int i;

```

```

    //define the size of the required array of integers
    //according to the size of the set
    m_nArrSize = ceil((double)n/(double)UINT_SIZE);

    //create the array of integers
    m_pnArr = new unsigned int [m_nArrSize+1];

    for(i = 0; i <= m_nArrSize; i++)          {
        m_pnArr[i] = 0;
    }
}

public:
//...
//check whether the set includes a point or not
bool includes(const Point* pPoint) const
{
    unsigned int u, i, j;

    u = pPoint->index();                  //obtain index of the point

    //define a number j of corresponding bit
    j =      UINT_SIZE - 1 - (u%UINT_SIZE);

    //define an integer in the array, which includes
    //the required j-th bit
    i = floor((double)u/(double)UINT_SIZE);

    if(m_pnArr[i] & (1 << j))           //check the value of the
        return true;                      //bit
    else
        return false;
}

//insert a point into the set
void insert(const Point* pPoint)
{
    unsigned int u, i, j;

    u = pPoint->index();                  //obtain index of the point
    //define a number j of corresponding bit
    j =      UINT_SIZE - 1 - (u%UINT_SIZE);

    //define an integer in the array, which includes
    //the required j-th bit
    i = floor((double)u/(double)UINT_SIZE);

    //set the j-th bit to "1" in the i-th integer
    m_pnArr[i] = m_pnArr[i] | (1 << j);

    m_nSize = update_size();
}

//exclude a point from the set
void exclude(const Point* pPoint)
{
    unsigned int u, i, j;

    u = pPoint->index();                  //obtain index of the point

```

```

//define a number j of corresponding bit
j =      UINT_SIZE - 1 - (u%UINT_SIZE);

//define an integer in the array, which includes
//the required j-th bit
i = floor((double)u/(double)UINT_SIZE);

//set the j-th bit to "0" in the i-th integer
m_pnArr[i] = m_pnArr[i] & ~(1 << j);

m_nSize = update_size();
}

//find intersection of this set and a set *pSet and
//write result into the set *pResSet
Set* intersection(const Set* pSet, Set* pResSet) const
{
    unsigned int i, j;

    if(pSet->m_nSize == 0) return pResSet;

    //apply Boolean bitwise AND operator over the array
    //of integers to obtain the intersection
    for(i = 0; i <= pResSet->m_nArrSize; i++) {
        pResSet->m_pnArr[i] =
            m_pnArr[i] & pSet->m_pnArr[i];

        for(j = 0; j < UINT_SIZE; j++)
            pResSet->m_nSize +=
                (pResSet->m_pnArr[i] & (1 << j))?1:0;
    }
    return pResSet;
}

//find union of this set and a set *pSet and
//write result into the set *pResSet
Set* unionation(const Set* pSet, Set* pResSet) const
{
    unsigned int i, j;

    if(pSet->m_nSize == 0) return pResSet;
    //apply Boolean bitwise OR operator over the array
    //of integers to obtain the union
    for(i = 0; i <= pResSet->m_nArrSize; i++) {
        pResSet->m_pnArr[i] =
            m_pnArr[i] | pSet->m_pnArr[i];

        for(j = 0; j < UINT_SIZE; j++)
            pResSet->m_nSize +=
                (pResSet->m_pnArr[i] & (1 << j))?1:0;
    }
    return pResSet;
}
//...
}

```

As indicated above, the class and functions include only the key lines required for the definition of the data structure, and do not include tests of data correctness and the lines which define probabilities, distances, and other information.

Note that the presented class `Set` includes the field `m_nIndex`, which has a similar meaning to the field `m_index` in the class `Point` and represents the index of the set A in the search space $\mathcal{X} = 2^X$. In other words, for $x_i \in X$ and $A_k \in \mathcal{X}$, the field `Point::m_index` corresponds to i and the field `Point::m_index` corresponds to k . Let us briefly outline the class `SearchSpace`, which defines the search space \mathcal{X} :

```
class SearchSpace
{
private:
    Set** m_ppSets;           //array of sets
    unsigned int m_nSize;      //size of search space

    Points m_Points;          //array of points defined by
                               //the use of additional type
    unsigned int m_nPointsSize; //size of the array of points

    unsigned int** m_nppArrT;   //array of integers, which
                               //defines the relation of the point to a certain set

    //two functions that define the size of the search space
    //factorial
    unsigned long int fact(unsigned int n) {...}
    //binomial coefficient
    unsigned long int binom(unsigned int n, unsigned int k)
    {
        return (fact(n)/(fact(k)*fact(n-k)));
    }

    //create array of sets
    void create(const unsigned int size)
    {
        unsigned int i;

        m_nSize = size;
        m_ppSets = new Set* [size];

        for(i = 0; i < m_nSize; i++) {
            m_ppSets[i] = new Set(m_nPointsSize);
            m_ppSets[i]->set_points(&m_Points);
            m_ppSets[i]->set_index(i);
        }

        init_space();
        //...
    }
}
//insert the sets in the sample space
void init_space()
{
    unsigned int i, j, k, m, n;
```

```

i = 1;                                //set m_ppSets[0] is empty
for(k = 1; k <= m_nPointsSize; k++) {
    n = binom(m_nPointsSize, k);

    for(j = 1; j <= n; j++) {
        for(m = 1; m <= k; m++) {
            m_ppSets[i]->insert(
                m_Points.point(m_nppArrT[i][m]-1));
        }
        i++;
        if(i == m_nSize - 1) break;
    }
}

i = m_nSize - 1;
m_ppSets[i]->erase();
for(m = 0; m < m_nPointsSize; m++) {
    m_ppSets[i]->insert(
        m_Points.point(m_nppArrT[i][m]-1));
}

//...
}

//specify lexicographical successor of the set
void set_lex_successor(unsigned int current,
                       unsigned int k)
{
    unsigned int i, j, next = current + 1;

    //...

    for(i = 0; i < m_nPointsSize; i++) {
        m_nppArrT[next][i] = m_nppArrT[current][i];
    }

    i = k;
    while((i >= 1) &&
          (m_nppArrT[current][i] == m_nPointsSize - k + i)) {
        i--;
    }

    for(j = i; j <= k; j++) {
        m_nppArrT[next][j] =
            m_nppArrT[current][i] + 1 + j - i;
    }
}

//create array of integers, which defines
//the correspondence of the points to the sets
void create_arr_t()
{
    unsigned int i;

    m_nppArrT = new unsigned int* [m_nSize];

    for(i = 0; i < m_nSize; i++) {

```

```

        m_nppArrT[i] =
            new unsigned int [m_nPointsSize];
    }

    init_arr_t();      //define the integer array; see below
}

//define the array of integers
//it is assumed that the sets
//will be ordered lexicographically
void init_arr_t()
{
    unsigned int i, j, k, m, n;

    for(i = 0; i < m_nSize; i++) {
        for(j = 0; j < m_nPointsSize; j++)
            m_nppArrT[i][j] = 0;
    }

    i = 1;
    for(k = 1; k <= m_nPointsSize; k++) {
        for(m = 1; m <= k; m++) {
            m_nppArrT[i][m] = m;
        }
    }

    n = binom(m_nPointsSize, k);

    for(j = 1; j <= n; j++) {
        set_lex_successor(i, k);

        i++;
        if(i == m_nSize - 1) break;
    }
}

i = m_nSize - 1;
for(m = 0; m < m_nPointsSize; m++) {
    m_nppArrT[i][m] = m+1;
}
}

public:
    //...
}

```

Similar to the class `SampleSpace` and class `Locations` presented above, the class `SearchSpace` was defined as a singleton.

The last data type, which is required by the ILRTA* and IMTS algorithms, is the partitions space χ . This space includes all possible partitions $\alpha = \{A | A \subset X\}$ of the sample space X . Since the sets $A \subset X$ which are included in the partitions $\alpha \in \chi$ are the elements $A \in \mathcal{X}$ of the search space \mathcal{X} , the class `PartitionsSpace` is based on the class `SearchSpace`. First, let us present the class `Partition`, which defines the elements of the partitions space and aggregates the sets that are defined by the class `Set` and included in the search space defined by the class `SearchSpace`. Since each partition is a set of elements which are already included in the search space, the class `Partition`

is defined similarly to the class `Set` and based on the array of integers, whose bits define whether the set is included in the partition or not. To distinguish the elements of the partitions from the elements of the sets, the sets $A \in \alpha$ are called *atoms*. The functions which are required for data type definition are similar to the functions that arrear in the class `Set`. As an example of operations with partitions, we present below just two functions: the first checks the refinement relation and the second calculates the Hamming distance between the partitions (see Section 4.1).

```

class Partition
{
private:
    unsigned int m_nSize;           //size of the partition

    unsigned int *m_pnArr;          //array of integers, whose
                                   //bits indicate whether the set is in the partition

    unsigned int m_nArrSize;        //size of array of integers
    //...
public:
    //...
    bool is_refinement_of(const Partition* pPartition) const
    {
        unsigned int i, j;
        Set* pSetThis, *pSet;
        bool res;

        if(m_nSize < pPartition->m_nSize) return false;

        for(i = 0; i < m_nSize; i++) {
            res = false;

            pSetThis = atom(i);      //returns i-th element
                                   //of partition

            for(j = 0; j < pPartition->m_nSize; j++) {
                pSet = pPartition->atom(j);

                if(pSetThis->is_subset_of(pSet))
                    res = true;
            }

            if(res == false) return false;
        }
        return true;
    }

    bool includes(const Set* pSet) const { ... }
    void insert(const Set* pSet) { ... }
    void exclude(const Set* pSet) { ... }

    unsigned int hamming_distance(
                                const Partition* pPartition) const
    {
        unsigned int i, j, d, a, b;

```

```

        if(pPartition->m_nSize == 0) return m_nSize;

        if(m_nSize == 0) return pPartition->m_nSize;

        d = 0;
        for(i = 0; i <= m_nArrSize; i++) {
            for(j = 0; j < UINT_SIZE; j++)
            {
                a = (this->m_pnArr[i] & (1 << j))?1:0;
                b = (pPartition->m_pnArr[i] & (1 << j))?1:0;

                if(a != b)
                    d++;
            }
        }
        return d;
    }
//...
}

```

Finally, let us outline the class `PartitionsSpace` that defines a container of partitions, which are defined by the class `Partition`. The implementation of this class is similar to the class `SearchSpace`.

```

class PartitionsSpace
{
private:
    Partition** m_ppPartitions;           //array of partitions
    unsigned long int m_nSize;             //size of partitions space

    const Points* m_pPoints;              //array of points defined by
                                           //the use of additional type
    unsigned int m_nPointsSize;           //size of the array of points

    const SearchSpace* m_pSearchSpace;     //pointer to the
                                           //search space
    unsigned int m_nSearchSpaceSize;       //size of search space
    unsigned int** m_nppArrF;              //array of integers, which
                                           //defines the relation of the set with a certain set

    unsigned int* m_npArrFmax;            //additional array,
                                           //which is used for temporary information storage
//...

//two functions that define the size of the search space
//Stirling number of the second kind (see Equation 4.5)
unsigned long int stirling2(unsigned int n,
                           unsigned int k) { ... }

//Bell number (see Equation 4.6)
unsigned long int bell(unsigned int n) { ... }

//create array of partitions (similar to SearchSpace)
void create(const unsigned long int size) { ... }

//insert partitions into the partitions space (similar to
//SearchSpace, the array m_nppArrF is specified)

```

```

void init_space(unsigned int m) { ... }

//create array of integers, which defines
//the correspondence of the sets to the partitions,
//and calls the next function init_arr_f()
void create_arr_f()
{
    //... (similar to the SearchSpace)
    init_arr_f(m_nPointsSize);
}

//define the array of integers
void init_arr_f(unsigned int m)
{
    unsigned long int index;
    unsigned int i, j;
    bool done;

    index = 0;
    done = false;

    while(!done) {
        index++;

        for(i = 0; i < m; i++) {
            m_nppArrF[index][i] =
                m_nppArrF[index-1][i];
        }

        for(i = m - 1; i > 0; i--) {
            if(i == 0) break;

            if(m_nppArrF[index][i] != m_npArrFmax[i])
                break;
        }

        if(i > 0) {
            m_nppArrF[index][i]++;
            for(j = i+1; j < m; j++) {
                m_nppArrF[index][j] = 1;

                if(m_nppArrF[index][i] ==
                   m_npArrFmax[i])
                    m_npArrFmax[j] =
                        m_npArrFmax[i] + 1;
                else
                    m_npArrFmax[j] =
                        m_npArrFmax[i];
            }
        }
        else {
            done = true;
        }
    }
}

//...
public:
//...
}

```

This outline finalizes the sketch of data structure definitions. As indicated above, the implementations of the data structures are based on the combinatorial algorithms presented by Kreher and Stinson [20]. Note again that the presented code is not complete and includes only key functions and fields. The complete C++ code can be found on the supporting CD.

Now let us present the key structures and functions of the code that implements the ILRTA* and IMTS algorithms.

4.4.2 Operations and algorithms

The ILRTA* and IMTS algorithms define the actions of the target and the searcher: in the ILRTA* algorithm (Algorithm 4.1) the target is static, while in the IMTS algorithm (Algorithm 4.5) the target moves according to the predefined probabilistic rule. The searcher, in its turn, acts according to the lines of the algorithms conducting the appropriate decision making. Thus, in order to specify correct usage of the actions and data access, the target and searcher can be defined by the separate classes that define the actions over the data structures presented above.

Both the target and the searcher act over the same partitions space χ , though the available partitions can be different. In the code below, we present the key operations over arbitrary partitions $\alpha = \{A | A \subset X\} \in \chi$ and binary partitions $\alpha = \{A_1, A_2\} \in \chi(2)$, including binary partitions such that either $|A_1| = 1$ or $|A_2| = 1$.

As demonstrated above, the ILRTA* and IMTS algorithms provide a general framework for different search algorithms that apply different decision-making criteria. Following this direction, the code implements these algorithms in generic form so that their type and decision-making criteria, as well as different estimated and actual distances, can be used. The implementation is similar for the classes that define both the target and the searcher, and will be clear from the outlines.

The class Target is defined as follows:

```
class Target
{
private:
    //...
    Locations* m_pLocations;           //the locations set
    unsigned int m_nSize;              //size of the locations set

    const BinaryPartitions* m_pBinaryPartitions;      //space
    //of binary partitions, which form a domain for the
    //target's movement

    Point* m_pLocation;                //current point at which the
    //the target is located

    Partition* m_pPartition;           //current partition, which
    //corresponds to the target's location

    TargetType m_enTargetType;         //type of the target:
    //Static target
    //Markovian target (can stay in its current state)
    //Brownian target (always changes its state)
```

```

DistanceType m_enDistanceType; //type of distance:
    //Ornstein, Rokhlin, or Hamming distance
float m_fRadius;           //neighborhood radius
//...
//set location in the sets of locations and partitions
void set_location(unsigned int i)
{
    Point* pPoint;

    if(m_pLocation)
        m_pLocations->clean_location(m_pLocation);
    //...
    m_pLocations->set_location(pPoint);
    m_pLocation = pPoint;
    m_pPartition = part_location();
}

//location in the partitions space
Partition* part_location() const {...}

//calculate appropriate distance
float distance(const Partition* pPartition_1,
               const Partition* pPartition_2) const
{
    switch(m_enDistanceType) {
        //...
        //case D_ORNSTEIN:
        //case D_ROKHILIN:
        //case D_HAMMING:
        //...
    }
    //...
}

public:
    //...
    //target's step
    Partition* step()
    {
        unsigned int i, j, nCur, n;
        float d;
        Partition* pCurP;

        if(m_enTargetType == T_STATIC) return m_pPartition;

        do {
            //...
            //obtain new target location according to
            //the movement rule
            //current location is nCur
            //index of the new location is n
        }while(n != nCur && m_pLocations->is_occupied(n));

        pCurP = m_pPartition;
        set_location(n);           //specifies m_pPartition for n

        d = distance(pCurP, m_pPartition);
        if(d >= m_fRadius) {

```

```

        set_location(nCur);
    }
    return m_pPartition;
}
//...
}
```

The searcher uses different algorithms and, according to the algorithm, can act either over the search space \mathcal{X} (see class `SearchSpace`) or over the partitions space χ (see class `PartitionsSpace`). In addition, the decision making was implemented on the basis of certain maximization criteria. The class `Searcher` is defined as follows:

```

class Searcher
{
private:
    const Locations* m_pLocations;           //the locations set
                                                //searcher has access to the function observe() only

    const SearchSpace* m_pSearchSpace; //search space

    const PartitionsSpace* m_pPartitionsSpace; //partitions
                                                //space

    Partition* m_pCurrentPartition;           //current
                                                //searcher's parti-
tion
    const Partition* m_pFinalPartition;       //final partition
                                                //speci-
fied by the target
    Array* m_pArrayEstimations;             //array of
                                                //estimated distances

    SearchAlgorithm* m_pSearchAlgorithm;      //search algorithm
                                                //ILRTA*, IMTS, Huffman, GOTA, MaxProbability,
                                                //MaxEntropy, MaxEntropyTree
    AlgorithmType m_enAlgorithmType;         //algorithm type
                                                //according to the algorithms above

    PartitionType m_enPartitionType;          //Binary partitions,
                                                //including partitions with single-point sets,
                                                //and arbitrary partitions
    DistanceType m_enDistanceType;           //actual distance type:
                                                //Ornstein, Rokhlin, or Hamming metrics
    EstimationType m_enEstimationType;        //estimated
                                                //distance type: zero, Ornstein, Rokhlin, or Hamming

    float m_fRadius;                         //neighborhood radius

    //update estimation (see Algorithms 4.1 and 4.5)
    void update_estimation(float fEstimation)
    {
        unsigned long int i;

        if(m_enPartitionType == P_BINARY ||
           m_enPartitionType == P_BINARY_ONE) {
            i = m_pCurrentPartition->bi_index();
        }
    }
}
```

```

    }
else {
    i = m_pCurrentPartition->index();
}
m_pArrayEstimations->set_element(i, fEstimation);
}

public:
//...
//define estimated distances according to distance type
void init_estimations(const Partition* pFinalPartition)
{
    unsigned long int i, size;
    float e;
    Partition* pIP = NULL;

    size = m_pArrayEstimations->size();
    m_pFinalPartition = pFinalPartition;

    //calculate estimations
    for(i = 0; i < size; i++) {
        //...
        //estimation is stored in the e variable
        switch(m_enEstimationType) {
            //...
            //case E_ZERO:
            //case E_ORNSTEIN:
            //case E_ROKHLIN:
            //case E_REAL:
                //estimated distance is equal to
                //actual distance
                switch(m_enDistanceType) {
                    //case D_ORNSTEIN:
                    //case D_ROKHLIN:
                    //case D_HAMMING:
                        ;
                        break;
                }
                m_pArrayEstimations->set_element(i, e);
        }
        return;
    }

    //update estimation for IMTS algorithm (Algorithm 4.5)
    void update_estimation(const Partition* pFinalPartition)
    {
        float fNewEstimation;

        fNewEstimation =
            m_pSearchAlgorithm->new_estimation_t(
                m_pCurrentPartition, m_pFinalPartition,
                pFinalPartition, m_fRadius);

        update_estimation(fNewEstimation);
        m_pFinalPartition = pFinalPartition;
    }

    //searcher's step
}

```

```

Partition* step()
{
    float fNewEstimation;
    Partition* pNextPartition;

    pNextPartition = m_pSearchAlgorithm->next_partition(
        m_pCurrentPartition,
        m_pFinalPartition, m_fRadius);

    fNewEstimation = m_pSearchAlgorithm->new_estimation(
        m_pCurrentPartition,
        m_pFinalPartition, m_fRadius);

    update_estimation(fNewEstimation);
    m_pCurrentPartition = pNextPartition;

    return m_pCurrentPartition;
}

//termination condition
bool stop()
{
    bool res;

    if(m_pFinalPartition == NULL)
        res = false;
    else
        res = m_pSearchAlgorithm->stop(pCurrentPartition,
                                         m_pFinalPartition);

    return res;
}
//...
}

```

As in the definition of the class `Searcher`, it uses several algorithms that are built according to the same scheme. To allow for such an implementation, the interface class `SearchAlgorithm` was defined and all particular algorithms inherited from this class. Below, we outline the fields of the class `SearchAlgorithm` and key lines of classes that define the ILRTA* and IMTS algorithms:

```

class SearchAlgorithm
{
protected:
    DistanceType m_enDistanceType;
    EstimationType m_enEstimationType;
    PartitionType m_enPartitionType;

    const PartitionsSpace* m_pPartitionsSpace;
    unsigned long int m_nPartitionsSpaceSize;

    const BinaryPartitions* m_pBinaryPartitions;
    unsigned long int m_nBinaryPartitionsSize;

    const Array* m_pArrayEstimations;
    unsigned long int m_nArraySize;

public:
    //...
}

```

The class `SearchIlrta`, which implements the ILRTA* algorithm, inherits from the class `SearchAlgorithm` and includes the appropriate functions which realize the operations defined by the algorithm. Below, we outline the function `next_partition()`, which provides the choice of the next partition according to the ILRTA* algorithm (Algorithm 4.1):

```
class SearchIlrta : public SearchAlgorithm
{
private:

public:
    //...
    Partition* next_partition(const Partition* pCurrentPartition,
                           const Partition* pFinalPartition) const
    {
        Partition* pTryP = NULL;
        Partition* pNext =
            const_cast<Partition*>(pCurrentPartition);
        unsigned long int i;
        float e, d, eMin;

        if(m_enPartitionType == P_BINARY) {
            eMin = m_pBinaryPartitions->points()->entropy();

            for(i = 0; i < m_nBinaryPartitionsSize; i++) {
                pTryP = m_pBinaryPartitions->partition(i);
                if(pTryP->is_equal_to(pCurrentPartition))
                    continue;

                e = estimation(pTryP->bi_index());
                d = distance(pTryP, pCurrentPartition);

                e = d + e;
                if(e <= eMin) {
                    eMin = e;
                    pNext = pTryP;
                }
            }
        }
        else if(m_enPartitionType == P_BINARY_ONE) {
            eMin = m_pBinaryPartitions->points()->entropy();

            for(i = 0; i < m_nBinaryPartitionsSize; i++) {
                pTryP = m_pBinaryPartitions->partition(i);
                if(pTryP->is_equal_to(pCurrentPartition))
                    continue;

                if(pTryP->atom(0)->size() != 1 &&
                   pTryP->atom(1)->size() != 1)
                    continue;

                e = estimation(pTryP->bi_index());
                d = distance(pTryP, pCurrentPartition);

                e = d + e;
                if(e <= eMin) {
                    eMin = e;
                    pNext = pTryP;
                }
            }
        }
    }
}
```

```

        }
    }
}
else {
    eMin = m_pPartitionsSpace->points()->entropy();

    for(i = 0; i < m_nPartitionsSpaceSize; i++) {
        pTryP = m_pPartitionsSpace->partition(i);
        if(pTryP->is_equal_to(pCurrentPartition))
            continue;

        e = estimation(pTryP->index());
        d = distance(pTryP, pCurrentPartition);

        e = d + e;
        if(e <= eMin) {
            eMin = e;
            pNext = pTryP;
        }
    }
}
return pNext;
}
}

```

Finally, we present a sketch of the class `SearchImts`, which defines the actions of the IMTS algorithm (Algorithm 4.5). As in the definition of the algorithm, the function `next_partition()` is similar to the function used in the ILRTA* algorithm and is defined in the class `SearchIlrta` presented above. Hence, in the class `SearchImts`, we outline the functions which implement updates of the estimated distances for the searcher's and the target's turns:

```

class SearchImts : public SearchAlgorithm
{
private:

public:
    //...
    Partition* next_partition(const Partition* pCurrentPartition,
    const Partition* pFinalPartition, const float fRadius) const
    {...}
    //estimated distances update for the searcher's turn
    float new_estimation(const Partition* pCurrentPartition,
    const Partition* pFinalPartition, const float fRadius) const
    {
        Partition* pTryP = NULL;
        unsigned long int i;
        float e, eMin, d, r, er;

        if(m_enPartitionType == P_BINARY) {
            eMin = m_pBinaryPartitions->points()->entropy();

            for(i = 0; i < m_nBinaryPartitionsSize; i++) {
                pTryP = m_pBinaryPartitions->partition(i);
                if(pTryP->is_equal_to(pCurrentPartition))
                    continue;

```

```

e = estimation(pTryP->bi_index());
d = distance(pTryP, pCurrentPartition);

r = (fRadius < d ? fRadius : d);

if(e > r) continue;

if(e <= eMin) {
    eMin = e;
}
}

else {
    eMin = m_pPartitionsSpace->points()->entropy();

for(i = 0; i < m_nPartitionsSpaceSize; i++) {
    pTryP = m_pPartitionsSpace->partition(i);
    if(pTryP->is_equal_to(pCurrentPartition))
        continue;

    e = estimation(pTryP->index());
    d = distance(pTryP, pCurrentPartition);

    r = (fRadius < d ? fRadius : d);

    if(e > r) continue;

    if(e <= eMin) {
        eMin = e;
    }
}
}

if(m_enPartitionType == P_BINARY)
    e = estimation(pCurrentPartition->bi_index());
else
    e = estimation(pCurrentPartition->index());

er = eMin + r;

if(e > er)
    return e;
else
    return (er < d ? er : d);
}

//estimated distances update for the target's turn
float new_estimation_t(const Partition* pCurrentPartition,
                       const Partition* pFinalPartition,
                       const Partition* pNewFinalPartition,
                       const float fRadius) const
{
    float e, eN, er;

    if(m_enPartitionType == P_BINARY)
        e = estimation(pCurrentPartition->bi_index());

```

```

        else
            e = estimation(pCurrentPartition->index()) ;

        eN = estimation(pCurrentPartition, pNewFinalPartition) ;

        er = eN - fRadius;
        if(e > er)
            return (e > 0 ? e : 0);
        else
            return (er > 0 ? er : 0);
    }
}

```

This outline completes the sketch of definitions of operations and algorithms. Again, as indicated above, the code is not complete and includes only key functions and fields. The complete C++ code can be found on the supporting CD.

4.5 Summary

This chapter introduced the ILRTA* algorithm and the IMTS algorithms. These algorithms follow other stochastic local search (SLS) procedures in general and AI procedures of search over graphs in particular.

The main idea in these proposed procedures is to represent partitions of the search space by the graph vertices, and the use of information theory metrics, particularly the Rokhlin metric, to measure the distance between these partitions. Given the above definitions, and similar to other AI search procedures, the proposed algorithms learn iteratively and find the shortest path between the searcher's partition and the target's partition, and therefore select a path in the graph that corresponds to the proper group tests to follow.

These algorithms are shown to generalize known coding methods, such as the Huffman and the GOTA, and provide relatively good results for the general probabilistic search problem. They are shown to converge to optimal solutions in the limited search problems where these exist. Finally, both the ILRTA* and IMTS enable an offline search scheme that can absorb side information during the search.

References

1. A.N. Shirayev *Probability*. Springer-Verlag: Berlin, etc., 1984.
2. P. Billingsley (1965). *Ergodic Theory and Information*. New York, London, Sydney: John Wiley & Sons, Inc.
3. Y. Nakamura (1969). Measure-theoretic construction for information theory. *Kōdai Mathematical Seminar Reports*, **21**, 133–150.
4. C.E. Shannon (1948). A mathematical theory of communication. *Bell System Technical Journal*, **27**(3), 379–423; **27** (4), 623–656.
5. P. Shields (1998). The interactions between ergodic theory and information theory. *IEEE Transactions on Information Theory*, **44** (6), 2079–2093.
6. V.A. Rokhlin (1967). Lectures on the entropy theory of measure-preserving transformations. *Russian Mathematical Surveys*, **22**, 1–52.
7. D.S. Ornstein (1971). Measure preserving transformations and random processes. *The American Mathematical Monthly*, **78** (8), 833–840.

8. D.S. Ornstein (1974). *Ergodic Theory, Randomness, and Dynamical Systems*. Yale University Press: New Haven and London.
9. W. Parry (1991). Notes on coding problems for finite state processes. *Bulletin of the London Mathematical Society*, **23**, 1–33.
10. Y.G. Sinai (1977). *Introduction to Ergodic Theory*. Princeton University Press: Princeton, NJ.
11. Y.G. Sinai 1995. *Modern Problems of Ergodic Theory*. Nauka, Moscow (Russian).
12. D.S. Ornstein and B. Weiss (2006) Entropy is the Only Finitely Observable Invariant, Discussion Paper 420, Center for the Study of Rationality, The Hebrew University of Jerusalem, May 2006.
13. N. Ruschin-Rimini, I. Ben-Gal, and O. Maimon (2013) Fractal geometry based statistical process control for non-linear auto-correlated processes. *IIE Transactions*, **45** (1.186), 1–19.
14. N.F.G. Martin, J.W. England (1984). *Mathematical Theory of Entropy*. Cambridge University Press: Cambridge.
15. R. López De Mántaras (1991). A distance-based attribute selection measure for decision tree induction. *Machine Learning*, **6**, 81–92.
16. A. Lloris-Ruiz, J.F. Gomez-Lopera, and R. Roman-Roldan (1993) Entropic minimization of multiple-valued logic functions. Proceedings of ISMVL'93, pp. 24–28.
17. S. Jaroszewicz (2003) Information-theoretical and combinatorial methods in data-mining. PhD thesis. University of Massachusetts, Boston, MA.
18. J. Peltonen (2004) Data exploration with learning metrics. PhD thesis. Helsinki University of Technology, Finland.
19. T.M. Cover, J.A. Thomas (1991). *Elements of Information Theory*. John Wiley & Sons, Inc.: New York.
20. D.L. Kreher, D.R. Stinson (1999). *Combinatorial Algorithms. Generation, Enumeration and Search*. CRC Press: Boca Raton, FL.
21. E. Kagan (2009) An informational search after static or moving targets in discrete space. PhD thesis. Tel-Aviv University, Israel.
22. E. Kagan and I. Ben-Gal (2009) Topology of the sample space in the problem of informational search for a moving target. Proceedings of the 4th Israeli Industrial Engineering Research Meeting IIERM'09, Maalot, Israel, March 4–5, 2009, pp. 31–44.
23. E. Kagan and I. Ben-Gal (2013) A Group-Testing Algorithm with Online Informational Learning. To appear in IIE Transactions.
24. R.E. Korf (1987) Real-time heuristic search: first results. Proceedings of AAAI'87, pp. 133–138.
25. R.E. Korf (1988) Real-time heuristic search: new results. Proceedings of AAAI'88, pp. 139–144.
26. R.E. Korf (1990). Real-time heuristic search. *Artificial Intelligence*, **42** (2–3), 189–211.
27. E. Kagan and I. Ben-Gal (2006) An informational search for a moving target. Proceedings of IEEE 24th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, November 15–17, 2006, IEEE Cat No. 06EX1327C, ISBN: 1-4244-0230-1/06.
28. S. Koenig (2001). Mini-max real-time heuristic search. *Artificial Intelligence*, **129** (1–2), 165–197.
29. S. Koenig and R.G. Simmons (1995) Real-time search on non-deterministic domains. Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-95), pp. 1660–1667.
30. I. Ben-Gal, E. Kagan, and N. Shkolnik (2008) Constructing classification trees via data mining and design of experiments concepts. Proceedings of ENBIS'8, Athens, Greece, September 21–25, 2008, e-publ.
31. I. Ben-Gal, N. Shkolnik, and E. Kagan (2007) Greedy learning algorithms and their applications to decision trees. Proceedings of the 7th European Annual Conference ENBIS'07, Dortmund, September 24–26, 2007, p. 36, e-publ.

32. E. Kagan and I. Ben-Gal (2007) A MDP test for dynamic targets via informational measure. Proceedings of Israeli Industrial Engineering Research Meeting IIERM'07, Tel-Aviv, Israel, March 20, 2007, pp. 39–53.
33. C.R.P. Hartmann, P.K. Varshney, K.G. Mehrotra, C.L. Gerberich (1982). Application of information theory to the construction of efficient decision trees. *IEEE Transactions on Information Theory*, **28** (4), 565–577.
34. J. Abrahams (1994). Parallelized Huffman and Hu-Tucker searching. *IEEE Transactions on Information Theory*, **40** (2), 508–510.
35. T.C. Hu, A.C. Tucker (1971). Optimum computer search trees and variable-length alphabetical codes. *SIAM Journal on Applied Mathematics*, **21**, 514–532.
36. R. Aumann (1974). Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, **1**, 67–96; See also: R. Aumann (1987). Correlated equilibrium as an expression of Bayesian rationality. *Econometrica*, **55**, 1–18.
37. E. Kagan (2005) Huffman game of searching for a single static target. *Proceedings of International Conference 'Digital Methods and Technologies DM'05'*, Vol. **2**, Taganrog, Anton Publishers, pp. 17–27.
38. R.H. Buchholz (1992). Perfect pyramids. *Bulletin of the Australian Mathematical Society*, **45** (3), 1–15.
39. T. Ishida and R.E. Korf (1991) Moving target search. Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-91), pp. 204–210.
40. T. Ishida, R.E. Korf (1995). Moving target search: a real-time search for changing goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17** (6), 609–619.
41. E. Kagan and I. Ben-Gal (2013) Moving Target Search Algorithm with Informational Distance Measures. To appear in Entropy.
42. S.M. Pollock (1970). A simple model of search for a moving target. *Operations Research*, **18**, 883–903.
43. S. Singh, V. Krishnamurthy (2003). The optimal search for a moving target when the search path is constrained: the infinite-horizon case. *IEEE Transactions on Automatic Control*, **48** (3), 493–497.

5

Applications and perspectives

In the previous chapters, we considered several models and methods of search that can be applied for different practical problems. Some of these models are purely theoretical, providing both an existing solution framework and a basis for different practical heuristics. Other models, especially those associated with search for static targets, can be directly implemented and require some programming techniques to obtain running algorithms.

In this chapter, we present two examples of heuristic procedures, which illustrate the actions and applications of informational search methods. The first procedure addresses a known problem in the data mining of object classification and presents an algorithm for creating a classification tree. The algorithm is based on the ILRTA* algorithm (see Section 4.2). It illustrates the methods of group-testing search (Section 2.2) and implements the search over graphs (as in Section 2.3).

The second heuristic procedure deals with a problem of search and screening (Section 2.1) and presents a real-time algorithm of search for static and moving targets by single and multiple searchers. The algorithm follows the framework of the MDP models of search (Section 3.3) and the IMTS algorithm (see Section 4.3). It implements informational criteria for estimating the future states of the system.

Finally, we describe several applications of search methods that follow from the presented discourse, such as cellular paging and quality control.

5.1 Creating classification trees by using the recursive ILRTA* algorithm

In general, classification trees can be viewed as decision trees that are constructed for the goal of (supervised) data classification, that is, for grouping input data to predefined classes according to a given target variable [1]. The decision tree applied to the classification

problem is described as a collection of linked decision nodes, extending downward from the root toward the leaves. Each internal node splits the instance space into two or more subspaces according to a certain discrete function of the input attributes' values. Therefore, each node can represent a partition of the sample space. In the most simple and frequent case, each split/test considers a single attribute, such that the instance space is partitioned according to the attribute's value. Each leaf of the tree is assigned to one class representing the most appropriate target variable value.

The goal of the construction of the decision tree is to build a tree such that it fully classifies the input data and has a minimal number of nodes. Such a task can be considered as a search for a specific partition of the sample space, and hence a variant of the ILRTA* algorithm (see Section 4.2) for this task is applicable.

It has been shown that the construction of the decision trees is a *NP*-complete problem [2]. Thus, implementation of optimal decision tree algorithms is feasible only for small problems, while for real-world problems, heuristic methods are often required. Most of the heuristic methods for the construction of decision trees, such as ID3 [3], C4.5 [4], and CART [5], are top-down procedures, starting from the whole sample set, which is represented by the root of the classification tree, and going downward to nodes that represent partitions of the sample space into classes. These algorithms are greedy by nature. At each step, the algorithm considers the *most beneficial* partition of the training set at that stage by using the outcome of a discrete function of the input attributes [6]. These algorithms recursively visit each decision node, selecting the optimal split until no further splits are possible. Both the ID3 and C4.5 algorithms use the concept of *information gain* or *entropy reduction* to select the optimal split.

Look-ahead search is a well-known technique for improving greedy algorithms [7]. When it is applied to decision tree induction, the algorithm tries to look ahead and to take the effect of deeper descendants into account. As a result, look-ahead requires strong computational resources. Below, we present a recursive version of the ILRTA* algorithm [8, 9], which allows one to walk around tradeoffs between greediness and computation complexity.

5.1.1 Recursive ILRTA* algorithm

The ILRTA* algorithm (Section 4.2, Algorithm 4.1) acts on the set of partitions of the sample space and obtains the path over this space, which starts with the initial searcher partition and results in the partition that represents the target location. In the application to classification problems, the target location and corresponding partition are associated with the target variable, and the searcher examines available partitions until the required one is obtained.

Recall the correspondence between the terms that are used by the ILRTA* algorithm and the terms that appear in formulations of classification problems. Consideration of such correspondence in particular is presented in Section 2.2.4; for illustration, see Example 2.18. Assume that the sample space is specified by a set $X = \{0, 1, 2, \dots, n\}$ of integers with equal probabilities, and let A_0, A_1, \dots, A_N be attributes that represent certain characteristics of the set X in any not necessary numerical form. From the point of view of the Generalized Optimal Testing algorithm (see Section 2.2.4, Example 2.18), the attributes can be considered as results of the tests T_1, T_2, \dots, T_N or division rules applied to the set X . An example of correspondence between the sample X and attributes A_0, A_1, \dots, A_N is presented in the following table:

Sample X	Attribute A_0	Attribute A_1	...	Attribute A_N
0	Yes	3	...	1
1	No	2	...	1
2	No	1	...	2
3	Yes	2	...	1
4	Yes	3	...	1
5	No	1	...	2
6	No	2	...	1
:	:	:	:	:
$n - 1$	No	1	...	1
n	Yes	3	...	2

In the table, attribute A_0 specifies terms ‘Yes’ and ‘No’ or, equivalently, Boolean values ‘1’ and ‘0’; attribute A_1 addresses characteristics, which are determined by the values ‘1,’ ‘2,’ and ‘3,’ and so on up to attribute A_N , which specifies the values ‘1’ and ‘2.’

Assume that attribute A_0 represents a class attribute from which the classification starts, and that each attribute A_j , $j = 0, 1, \dots, N$, splits the sample X and forms an appropriate partition. More precisely, attribute A_0 is represented by partition $\alpha_0 = \{A_{00}, A_{01}\}$, where set A_{00} includes all row indices from the sample X , which are related to the term ‘Yes’ in the column of the attribute A_0 , and set A_{01} is the same for all the indices with the term ‘No.’ Thus, $A_{00} = \{0, 3, 4, \dots, n\}$ and $A_{01} = \{1, 2, 5, 6, \dots, n - 1\}$. Similarly, for the partition $\alpha_1 = \{A_{10}, A_{11}, A_{12}\}$ that corresponds to the attribute A_1 we have $A_{10} = \{0, 4, \dots, n\}$, $A_{11} = \{1, 3, 6, \dots\}$, and $A_{12} = \{2, 5, \dots, n - 1\}$; and so on up to the partition $\alpha_N = \{A_{N0}, A_{N1}\}$ with the sets $A_{N0} = \{0, 1, 3, 4, \dots, n - 1\}$ and $A_{N1} = \{2, 5, \dots, n\}$. Since attributes A_j are unambiguously represented by partitions α_j , $j = 0, 1, \dots, N$, we use these terms interchangeably below.

Assume that after starting the classification process with attribute A_0 , the first attribute A_1 is chosen by some criterion. Attribute A_0 splits the sample X and forms partition $\alpha_0 = \{A_{00}, A_{01}\}$, and attribute A_1 forms partition $\alpha_1 = \{A_{10}, A_{11}, A_{12}\}$. Then, selection of both attributes A_0 and A_1 splits the sample X into the following partition:

$$\alpha_0 \vee \alpha_1 = \{A_{00} \cap A_{10}, A_{00} \cap A_{11}, A_{00} \cap A_{12}, A_{01} \cap A_{10}, A_{01} \cap A_{11}, A_{01} \cap A_{12}\},$$

which is obtained by the multiplication of partitions α_0 and α_1 (see Section 4.1).

In the same manner, if after selecting attribute A_1 one selects attribute A_j , then the resulting partition is given by the multiplication $\alpha_0 \vee \alpha_1 \vee \alpha_j$. Selecting all attributes up to attribute A_N results in the final partition $\alpha_{fin} = \alpha_0 \vee \alpha_1 \vee \dots \vee \alpha_j \vee \dots \vee \alpha_N$. As indicated in Section 2.2.1, such a procedure is equivalent to creating a search tree, and a sequence of selected partitions forms a decision policy. Thus, the classification problem is equivalent to the problem of finding the shortest path from an initial partition α_0 to the final partition α_{fin} in the set χ of all the available partitions of the sample. Since each partition α_j represents an attribute A_j , $j = 0, 1, \dots, N$, the elements of the partitions set χ are formed by all the possible multiplications of the partitions:

$$\alpha_i \vee \alpha_j, \quad \alpha_i \vee \alpha_j \vee \alpha_k, \quad \dots, \quad \alpha_0 \vee \alpha_j \vee \alpha_k \vee \dots \vee \alpha_N, \quad i \neq j \neq k, \quad i, j = 0, 1, \dots, N,$$

where, as indicated above, $\alpha \vee \beta = \{A \cap B | A \in \alpha, B \in \beta\}$. Note that such an approach follows a selection procedure with multi-attribute decision trees [10].

A straightforward application of the ILRTA* algorithm (Section 4.2, Algorithm 4.1) to the classification problem requires the creation of set χ of all available partitions. However, note that the solution of the classification problem is given by the most refined partition

$$\alpha_{fin} = \alpha_0 \vee \alpha_1 \vee \cdots \vee \alpha_j \vee \cdots \vee \alpha_N,$$

and it is sufficient to construct this partition only, without creating partitions set χ and applying any classification methods.

Nevertheless, for large samples and a large number of attributes the creation of the partition $\alpha_{fin} = \alpha_0 \vee \alpha_1 \vee \cdots \vee \alpha_j \vee \cdots \vee \alpha_N$ is problematic, since it requires evaluation of all the attribute values before making a classification decision. Such a requirement is costly and in real-life situations often there is a great advantage in classifying by evaluating only a small number of attributes.

To solve the classification problem, we suggest a *recursive* version of the ILRTA* algorithm, which is applied in each stage to the restricted sample according to already selected attributes. The idea of the algorithm is as follows [8, 9].

Let $\alpha = \{A_0, A_1, \dots, A_m\}$ be a partition. Denote by $\alpha|_B$ a restriction of the partition α by a set $B \subset X$ and define it as follows: $\alpha|_B = \{A_0 \cap B, A_1 \cap B, \dots, A_m \cap B\}$. It is clear that, regarding multiplication of the partitions in what follows, $\alpha \vee \beta = \cup_{B \in \beta} \alpha|_B = \cup_{A \in \alpha} \beta|_A$. Assume that on the sample X a probability mass function is defined and that the probabilities of the elements in the case of absence of information are equal, and that the partition α_0 represents a class attribute with which an algorithm starts. In addition, assume that the distances $d(\alpha, \beta)$ between the partitions α and β are specified by the Rokhlin metric (see Section 4.1), and that, similar to Section 4.2.1, distance estimations $\tilde{d}(\alpha, \beta)$ are such that for any two partitions α and β an admissibility assumption, that is, an inequality $\tilde{d}(\alpha, \gamma) \leq d(\alpha, \gamma)$, holds.

Assume that the current partition is $\alpha_{cur} = \{A_{cur,0}, A_{cur,1}, \dots, A_{cur,m}\}$. Then the recursive ILRTA* algorithm creates $(m + 1)$ threads. In each i th thread, $i = 0, 1, 2, \dots, m$, the current partition $\alpha_{cur(i)}$ is a set $\alpha_{cur(i)} = \{A_{cur(i)}, \emptyset\}$, with the normalized probabilities of the elements of $A_{cur(i)}$. Selection of the next partition is conducted from the neighborhood, which includes all the possible partitions by the attributes that have not yet been selected. For example, in the first stage of the algorithm such a neighborhood is defined as

$$N(\alpha_{cur(i)}) = \{\alpha_1|_{A_{cur(i)}}, \alpha_2|_{A_{cur(i)}}, \dots, \alpha_N|_{A_{cur(i)}}\},$$

where the probabilities are normalized and the distance estimations are updated, as well. If the cardinality of the chosen set is not greater than 1, or if another stopping criterion is met, then the thread terminates. Otherwise, the chosen set is considered as the current partition for this thread and the process continues. Formally, the recursive ILRTA* algorithm is outlined as follows.

Algorithm 5.1 (recursive ILRTA* algorithm) [8, 9]. Given partitions $\alpha_0, \alpha_1, \dots, \alpha_N$, admissible metrics \tilde{d} and d , and initial distance estimations $\tilde{d}_0(\alpha_j, \alpha_{fin})$, $j = 0, 1, \dots, N$, do:

1. Define $ResPath$ as a list of partitions.
2. Initialize current partition α_{cur} by initial partition α_0 : Set $\alpha_{cur} = \alpha_0$.

3. For each set $A \in \alpha_{cur}$ such that $|A| > 1$ do:

Start function $Search(A)$.

4. Return $ResPath$.

$Search(A)$. Given partitions $\alpha_0, \alpha_1, \dots, \alpha_N$ do:

1. Initialize current partition α_{cur} by $\{A, \emptyset\}$: Set $\alpha_{cur} = \{A, \emptyset\}$, where set A is an argument of the function.

2. Normalize probabilities of the elements of A .

3. Create neighborhood:

3.1. Set $N(\alpha_{cur}) = \{\alpha_1|_A, \alpha_2|_A, \dots, \alpha_N|_A\}$.

3.2. Normalize probabilities of neighbors.

3.3. Update distances $d(\alpha_{cur}, \alpha|_A)$ and distance estimations $\tilde{d}(\alpha_{cur}, \alpha|_A)$, $\alpha|_A \in N(\alpha_{cur})$, according to updated probabilities.

4. Create local final partition, if needed:

Set $\alpha_{fin} = \alpha_0|_A \vee \alpha_1|_A \vee \dots \vee \alpha_N|_A$.

5. Update estimation:

$$\text{Set } \tilde{d}(\alpha_{cur}, \alpha_{fin}) = \max \left\{ \frac{\tilde{d}(\alpha_{curr}, \alpha_{fin})}{\min_{\alpha \in N(\alpha_{curr})} \{d(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \alpha_{fin})\}} \right\}.$$

6. Choose next partition:

Set $\alpha_{next} = \operatorname{argmin}_{\alpha \in N(\alpha_{curr})} \{d(\alpha_{curr}, \alpha) + \tilde{d}(\alpha, \alpha_{fin})\}$; ties are broken randomly.

7. $ResPath.include(\alpha_{curr})$.

8. For each set $A \in \alpha_{cur}$ such that $|A| > 1$ do:

Start function $Search(A)$.

9. Return.

■

The presented recursive ILRTA* algorithm (i.e., Algorithm 5.1) implements similar techniques to the ILRTA* algorithm (Section 4.2, Algorithm 4.1). However, in contrast to the ILRTA* algorithm, this algorithm creates a required set of restricted partitions recursively following the local neighborhood of the current partition, and does not require creation of the set χ of all available partitions beforehand.

5.1.2 Recursive ILRTA* with weighted distances and simulation results

Modification of the recursive ILRTA* algorithm (Algorithm 5.1) with respect to the definition of distance d and distance estimation \tilde{d} leads to the Double Information Distance (DID) algorithm [11]. In this modification, distances d and distance estimations \tilde{d} were weighted by certain coefficients w_1 and w_2 depending on the considered database. Thus, in

the DID modification of the recursive ILRTA* algorithm (Algorithm 5.1), Step 6 of the function $\text{Search}(A)$ was implemented in the form

$$\alpha_{\text{next}} = \operatorname{argmin}_{\alpha \in N(\alpha_{\text{curr}})} \{w_1 d(\alpha_{\text{curr}}, \alpha) + w_2 \tilde{d}(\alpha, \alpha_{\text{fin}})\},$$

with weighting coefficients w_1 and w_2 specified by previously conducted learning over the analyzed database.

The DID algorithm was trialed over several known databases and compared to the ID3 [3] and C4.5 [4] algorithms. The results of the trials are presented in the following table [11]. The best results are marked in bold.

Dataset	Area	Size		ID3		C4.5		DID		
		No. of instances	No. of attributes	AD	TC (%)	AD	TC (%)	AD	TC (%)	WC
Running example	N/A	12	4	2.083	N/A	2.083	N/A	1.83	N/A	(-2,1)
Monk's-1	Robotics	124 (432)	6	3.21	82	3.32	82	2.66	96.7	(-5,1)
Monk's-2	Robotics	169 (432)	6	4.34	70.4	4.6	75	4.2	66	(-2,1)
Monk's full random set	Robotics	216 (216)	6	1.93	100	2.04	100	1.8	100	(-2,1)
Connect4	Games	67 557	42	5.85	73.8	10.16	79.4	5.64	75	(-5,1)
SPECT heart	Medicine	80 (187)	22	9.6	75.1	10.2	80.3	9.3	76	(-5,1)
Voting	Social	435	16	1.8	96	2.2	96.6	2.1	96	(-1,1)
Balance scale	Social	625	4	3.4	76.3	3.4	78.6	3.3	76.6	(-2,1)
Cars	General	1728	6	2.82	77.1	2.83	77	2.77	78.5	(-1,1)
Tic-Tac-Toe	Games	958	9	4.62	80.6	4.62	80.4	4.6	76.2	(-1,1)
Soybeans	Life	47	35	1.35	100	2.37	97	1.32	97	(-1,1)
Lymphography	Medicine	148	18	2.71	75.1	6.51	77.3	2.6	72.6	(-2,1)

In the table, AD denotes Average Depth, TC True Classification percentage, and WC, Weights Combination with the values (w_1, w_2) . If the dataset is already partitioned for training and test sets, then in the number of instances, a value in brackets represents the number of instances in the given test set.

Following the results obtained, it is indicated in [11] that:

the proposed algorithm outperforms both ID3 and C4.5 classifiers in almost each of the problems above with respect to the average depth of the constructed tree model . . . [However,] with respect to the TC percentage, the DID algorithm is less significantly superior. In most of the problems, the C4.5 tree model gives a better classification rate than the other two algorithms, though this advantage is relatively small. The DID algorithm performs better than the other two algorithms when applied to the Monk's-1 dataset. In most of the other cases, the DID algorithm ranks second, usually only slightly worse than the best successful classifying algorithm.

The simulation results, as well as those presented in the previous chapters on analytical considerations, assure the effectiveness of the ILRTA* algorithm (Section 4.2, Algorithm 4.1) and its recursive version (Algorithm 5.1) in classification problems. In addition, the simulations indicated a relation between the performance of the algorithm and the type of dataset, as follows from the considerations presented in Section 4.1. Nevertheless, note

that since Algorithm 5.1 essentially employs recursive creation of the partitions set, it is not applicable to those problems with changing probabilities of attributes.

5.2 Informational search and screening algorithm with single and multiple searchers

In this section, we consider a real-time algorithm of search for a static or moving target, which formalizes the following task. A target is located in a discrete domain or moving over the domain according to a definite Markov process. The search is conducted by an autonomous mobile agent that starts with the initial target location probabilities, and at each search step obtains information regarding target location in the agent's local neighborhood. The goal is to find the agent's path such that the target is detected in a minimal number of steps.

Such a task follows a general search and screening approach (see Section 2.1). Below, we present an algorithm with estimated future states of the system [12]. As indicated above, the algorithm follows Singh and Krishnamurthy's generalization of the Eagle model (see Section 3.3.1) and the branch-and-bound Procedure 3.1, and, like the IMTS algorithm (see Section 4.3), implements informational estimating criteria. In further developments [13, 14], this algorithm was implemented for the search with erroneous detections.

5.2.1 Definitions and assumptions

Let us recall the definitions and notation which were applied in the search and screening methods (Section 2.1.1). Assume that the sample space $X = \{x_1, x_2, \dots, x_n\}$ represents a rectangular geographical domain of size (n_1, n_2) , $n = n_1 \times n_2$, and the points $x_i \in X$, $i = 1, 2, \dots, n$, represent the cells in which the target can be located at any time $t = 0, 1, 2, \dots$. As indicated above, we index the points of X in a linear order such that for the Cartesian coordinates (j_1, j_2) of the point x_i , its index is given by $i = ((j_1 - 1)n_1 + j_2)$, where $j_1 = 1, \dots, n_1$ and $j_2 = 1, \dots, n_2$.

The target location probabilities at time t , $t = 0, 1, 2, \dots$, are denoted by $p^t(x_i) = \Pr\{x^t = x_i\}$, $i = 1, 2, \dots, n$, where $x^t \in X$ denotes the target location at time t . As usual, we assume that $0 \leq p^t(x_i) \leq 1$ and $\sum_{i=1}^n p^t(x_i) = 1$. The target's movement is governed by a discrete Markov process with transition probability matrix $\rho = [\rho_{ij}]_{n \times n}$, where $\rho_{ij} = \Pr\{x^{t+1} = x_j | x^t = x_i\}$ and $\sum_{j=1}^n \rho_{ij} = 1$ for each $i = 1, 2, \dots, n$. If matrix ρ is a unit matrix, then the target is static, otherwise we say that it is moving. Given target location probabilities $p^t(x)$, $x \in X$, at time t , location probabilities at the next time $t + 1$ are calculated as $p^{t+1}(x_i) = \sum_{j=1}^n \rho_{ij} p^t(x_j)$, $i = 1, 2, \dots, n$.

The searcher moves over the sample space and at each time t , $t = 0, 1, 2, \dots$, looks for a target by testing a neighboring observed area $A^t \subset X$ of a certain radius $r > 0$, which is defined by using the metric of the considered geographical domain. Similar to the Singh and Krishnamurthy model (see Section 3.3.1), we do not restrict the size of the available observed areas, but assume that the areas have equal size and the trajectory of the searcher is continuous, as shown in Figure 2.7.

The observation result is specified by the detection probability function φ , which specifies the probability $\varphi(A^t)$ of detecting the target, while the searcher observes the area A^t . In contrast to the search and screening procedure, in the present algorithm we assume that

the probability $\varphi(A^t)$ is specified by the indicator function $\mathbf{1}(x^t, A^t)$; that is, if at time t the target is located at the point $x^t \in A^t$, then $\varphi(A^t) = 1$, otherwise it is assumed that $\varphi(A^t) = 0$. In other words, detection function φ represents a definite range observation law (see Figure 2.3) and is defined in the form of observation function z , which is implemented in the group-testing search algorithms (see Section 2.2).

Following a general scheme of MDP model search (see Figure 3.2), the search process is conducted as follows. The searcher deals with the estimated location probabilities $\tilde{p}^t(x_i)$ that for the initial time $t = 0$ are equal to the initial location probabilities $\tilde{p}^{t=0}(x_i) = p^{t=0}(x_i)$, $x_i \in X$, $i = 1, 2, \dots, n$. At each time t , the searcher chooses an observed area $A^t \subset X$ and observes it. If an observation results in $\varphi(A^t) = 1$, the search terminates. Otherwise, the agent updates estimated location probabilities $\tilde{p}^t(x_i)$ over the sample space X according to the Bayes rule and obtains the observed probabilities denoted by $\bar{p}^t(x_i)$. The agent applies the target's transition probability matrix $\|\rho_{ij}\|_{n \times n}$ to the observed probabilities, and obtains the estimated probabilities $\tilde{p}^{t+1}(x_i) = \sum_{j=1}^n \rho_{ij} \bar{p}^t(x_j)$, $i = 1, 2, \dots, n$, for the next time $t + 1$. Given the estimated probabilities $\tilde{p}^{t+1}(x)$, $x \in X$, the search agent makes a decision regarding the next observed area $A^{t+1} \subset X$ and the process continues. As indicated above, the goal is to find a continuous path of the searcher such that the search procedure terminates in a minimal average (over possible target moves) number of steps.

As in Example 3.6, in the algorithm we implement a Manhattan metric over the domain, and assume that at each time t , both the search agent and the target can apply one of the following movements:

$$V = \{v_1, v_2, v_3, v_4, v_5\},$$

where v_1 stands for ‘move north,’ v_2 ‘move south,’ v_3 ‘move east,’ v_4 ‘move west,’ and v_5 ‘stay in the current point.’ These are the simplest movements that correspond to the Manhattan metric; for other metrics additional movements can be considered. The target's choice of movement is conducted according to its transition probability matrix ρ , while the agent's choice follows the decision that is obtained according to the result $\varphi(A^t)$ from observation of the area A^t around the agent.

5.2.2 Outline of the algorithm and related functions

As indicated above, the algorithm is based on a probabilistic version of local search with estimated global distances, while decision making utilizes distance estimations and the characteristics of the target's movement. An outline of the suggested search algorithm includes the following steps.

Algorithm 5.2 [12] Given sample space X , initial target location probabilities $p^{t=0}(x_i)$, $i = 1, 2, \dots, n$, and transition probabilities matrix ρ , do:

1. The search agent starts with estimated probabilities $\tilde{p}^{t=0}(x_i) = p^{t=0}(x_i)$, $i = 1, 2, \dots, n$, and an initial observed area $A^{t=0}$.
2. If $\varphi(A^{t=0}) = 1$, then the target is found and the process terminates. Otherwise, the following steps are conducted.
3. While $\varphi(A^t) \neq 1$, do:
 - 3.1. Calculate probabilities $\bar{p}^t(x_i)$, $i = 1, 2, \dots, n$:

- 3.1.1. For all x_i do: If $x_i \in A^t$ then: Set $\tilde{p}^t(x_i) = 0$.
- 3.1.2. For all x_i do: Set $\bar{p}^t(x_i) = \tilde{p}^t(x_i)/\sum_{j=1}^n \tilde{p}^t(x_j)$.
- 3.2. Calculate probabilities $\tilde{p}^{t+1}(x_i)$, $i = 1, 2, \dots, n$:
 - 3.2.1. For all x_i do: Set $\tilde{p}^{t+1}(x_i) = \sum_{j=1}^n \rho_{ij} \bar{p}^t(x_j)$.
- 3.3. Calculate weights w_k for the movements $v_k \in V$:
 - 3.3.1. For all v_k do: Set $w_k = \sum_{x \in \tilde{A}_k} \tilde{p}^{t+1}(x)$, where \tilde{A}_k is an area that will be observed if the agent moves according to movement v_k .
- 3.4. Estimate global probabilities if needed:
 - 3.4.1. If $w_k = 0$ for each k then:
 - a. Set $\tilde{\mathcal{P}} = \text{estimate probabilities}(\tilde{p}^{t+1}, \rho)$.
 - Else
 - b. Set $\tilde{\mathcal{P}} = \tilde{p}^{t+1}$.
- 3.5. Apply local decision making:
 - 3.5.1 Set $A^{t+1} = \text{next observed area}(\tilde{\mathcal{P}})$.
- 3.6. Move to the area A^{t+1} :
 - 3.6.1. Set $t = t + 1$ and obtain observation result $\varphi(A^t)$.
4. Return obtained area A^t . ■

The algorithm follows a general local search approach with forward estimation of the target location probabilities.

The algorithm includes two key functions:

- the global probability estimation function *estimate probabilities()* and
- the local decision-making function *next observed area()*.

The first function resolves an uncertainty for local decision making, while the second conducts local decision making and points to the next observed area.

Let us start with the *next observed area()* function. In the algorithm, we implement local decision making that is based on a discrete version of the maximum gradient over expected probabilities $\tilde{p}(x_i)$ from the current location of the search agent to possible observed areas. In other words, the agent chooses the next feasible observed area according to the defined movements V for which the observation will maximally affect the probabilities $\tilde{p}(x_i)$ as they are determined in Line 3.4 of the algorithm. The distances between the probability mass functions \tilde{p}_k , where k corresponds to the indices of the movements $v_k \in V$, and the probability mass function \tilde{p} are calculated according to the Ornstein distance measure [15] (see Section 4.1), which in the considered case obtains the simple form

$$d(\tilde{p}_k, \tilde{p}) = \sum_{i=1}^n |\tilde{p}_k(x_i) - \tilde{p}(x_i)|.$$

Given the set of possible movements V , the outline of the local decision-making function is as follows.

next observed area(\tilde{p}) :

1. For all movements $v_k \in V$ do:
 - 1.1. Determine A_k according to v_k .
 - 1.2. Set $w_k = \sum_{x \in A_k} \tilde{p}(x)$.
 - 1.3. For all x_i do: If $x_i \notin A_k$ then: Set $\tilde{p}_{k1}(x_i) = 1$.
 - 1.4. For all x_i do: Set $\bar{p}_{k1}(x_i) = \tilde{p}_{k1}(x_i) / \sum_{j=1}^n \tilde{p}_{k1}(x_j)$.
 - 1.5. For all x_i do: If $x_i \in A_k$ then: Set $\tilde{p}_{k0}(x_i) = 0$.
 - 1.6. For all x_i do: Set $\bar{p}_{k0}(x_i) = \tilde{p}_{k0}(x_i) / \sum_{j=1}^n \tilde{p}_{k0}(x_j)$.
 - 1.7. Set $d(A_k) = w_k d(\bar{p}_{k1}, \tilde{p}) + (1 - w_k) d(\bar{p}_{k0}, \tilde{p})$.
2. Set $A = \operatorname{argmax}\{d(A_k)\}$ over all A_k ; ties are broken randomly.
3. Return A .

The calculations of the function lines have the same meaning as in the corresponding lines of the algorithm. In particular, Line 1.2 defines the weights for the movements, while in Lines 1.3–1.6 the observed probabilities for successful and unsuccessful observations are calculated. In addition, in Line 1.7 the Ornstein distance averaged over successful and unsuccessful observations is calculated, and in Line 2 the area, for which this distance is maximal, is chosen.

The local decision-making function acts over given probabilities $\tilde{p}(x_i)$, $i = 1, 2, \dots, n$. If there is no uncertainty as defined in Line 3.4 of Algorithm 5.2, then this function is applied to the probabilities $\tilde{p}^{t+1}(x_i)$. If, on the contrary, for all possible movements $v_k \in V$ the probabilities $\tilde{p}^{t+1}(x_i)$ are equal to 0 (see Line 3.3 of Algorithm 5.2), then to resolve the uncertainty global probability estimation is applied.

The estimation of probabilities is implemented by using the function *estimate probabilities()* that starts with the probabilities $\tilde{p}^{t+1}(x_i)$ and results in the probabilities $\tilde{p}(x_i)$, $i = 1, 2, \dots, n$. Probabilities $\tilde{p}(x_i)$ are calculated by using the estimated target's transition probability matrix \tilde{p} as target location probabilities after a certain number m of steps, while the calculation starts from the agent's expected probabilities $\tilde{p}^{t+1}(x_i)$, $i = 1, 2, \dots, n$.

The indicated number of steps m is defined by using the estimated target location and by the entropy measure [16] (see Equation 2.40) that specifies the lack of information available to the searcher. Since the search agent observes the areas A that include an equal number of points, the entropy is calculated by using the size s of the areas. Then, the required or missing information is defined as follows:

$$I(\tilde{p}^{t+1}) = \log_s(n) - \sum_{i=1}^n \tilde{p}^{t+1}(x_i) \log_s \left(\frac{1}{\tilde{p}^{t+1}(x_i)} \right).$$

Note that the logarithms are calculated on the base size s of the observed areas A . Such a calculation follows the assumption that the areas A have the same size and the observations are perfect.

The estimated target location \tilde{x} is defined by a center of distribution \tilde{p}^{t+1} that is calculated as a ‘center of gravity’ for the probabilities $\tilde{p}^{t+1}(x_i)$, $i = 1, 2, \dots, n$, over a sample space X . For example, over the rectangular domain of size (n_1, n_2) , $n = n_1 \times n_2$, coordinates $(\tilde{J}_1, \tilde{J}_2)$, the estimated target location \tilde{x} , are calculated as follows:

estimated location(\tilde{p}^{t+1}):

1. Set $\tilde{J}_1 = 0$ and $\tilde{J}_2 = 0$.
2. For $j_1 = 1, \dots, n_1$ do:
For $j_2 = 1, \dots, n_2$ do:
Set $\tilde{J}_1 = \tilde{J}_1 + j_1 \times \tilde{p}^{t+1}(x_i)$, $i = ((j_1 - 1)n_1 + j_2)$
3. For $j_2 = 1, \dots, n_2$ do:
For $j_1 = 1, \dots, n_1$ do:
Set $\tilde{J}_2 = \tilde{J}_2 + j_2 \times \tilde{p}^{t+1}(x_i)$, $i = ((j_1 - 1)n_1 + j_2)$.
4. Set $\tilde{J}_1 = \text{round}(\tilde{J}_1)$ and $\tilde{J}_2 = \text{round}(\tilde{J}_2)$.
5. Return $\tilde{x} = (\tilde{J}_1, \tilde{J}_2)$.

Note that the direction of passing the domain in Lines 2 and 3 is different, as I is specified by the different order of the indices.

Denote by y the current location of the search agent. Then, by using information value $I(\tilde{p}^{t+1})$ and the above-defined function for calculating the estimated target location, the function *estimate probabilities*() is outlined as follows.

estimate probabilities(\tilde{p}^{t+1}, ρ):

1. Calculate estimated target location:
Set $\tilde{x} = \text{estimated location}(\tilde{p}^{t+1})$.
2. Calculate distance between \tilde{x} and agent’s location y :
3. Set $d = \text{distance}(\tilde{x}, y)$ and calculate number of estimated steps:
 $m = \text{ceil}(I(\tilde{p}^{t+1})) \times \text{round}(d)$.
4. For all x_i do: Set $\tilde{\rho}(x_i) = \tilde{p}^{t+1}(x_i)$.
5. Do m times:
 - 5.1 For all x_i do: set $\rho(x_i) = \tilde{\rho}(x_i)$.
 - 5.2 For all x_i do: set $\tilde{\rho}(x_i) = \sum_{j=1}^n \tilde{\rho}_{ij} \rho(x_j)$.
6. Return $\tilde{\rho}$.

In the function, *round*(b) stands for calculating the closest integer number of the real number b , and *ceil*(b) stands for calculating the closest integer number that is greater than b . For two points $x = (j_1, j_2)$ and $x' = (j'_1, j'_2)$, function *distance*(x, x') calculates a geographical distance between the points. In the considered search in discrete space, the distance was defined by the Manhattan metric; hence

$$\text{distance}(x, x') = |j_1 - j'_1| + |j_2 - j'_2|.$$

The presented outlines of the algorithm and functions can be implemented directly in the form of computer procedures. Below, we present the results of numerical simulations of the algorithm.

5.2.3 Numerical simulations of search with single and multiple searchers

The feasibility and performance of the suggested algorithm were studied by using simulated trials with different initial distributions of the target location and different types of movement. Below, we present an example of such simulations that have been conducted for search over a square domain of size $50 \times 50 = 2500$ points.

In this example, each trial included 1000 sessions, and for each session initial target location probabilities were generated by a combination of 10 bi-normal distributions with random centers and unit standard deviations. In each session, initial locations of the target were chosen randomly according to the initial location probabilities.

The trials were executed both for a static target and for two types of Markovian target. In each session, the search was executed up to the agent finding the target. The outline of the trial is as follows:

Trial:

1. Specify searcher's starting point and radius of observed areas.
2. Specify characteristics of initial target location probabilities.
3. Initialize transition probability matrix according to the specified type of movement.
4. Repeat 1000 times:

Session:

- 4.1 Initialize target location probabilities and initial location.
- 4.2 Save initial distance between the searcher and target.
- 4.3 Run Algorithm 5.2.
- 4.4 Save the number of search steps which are required for Algorithm 5.2 to find the target.
5. Return 1000 pairs of initial distances and number of search steps obtained.

Note that the initial distance between the searcher and the target (Line 4.2) represents the number of steps which the searcher is required to move to the target, given that the target is static and its location is perfectly known to the searcher.

In the trials, the average distance between the initial target location and the agent's starting point was approximately 37.0 and the standard deviation was approximately 17.0. Like the simulations in Section 4.3.3, the possible movements of the Markovian target are 'move north,' 'move south,' 'move east,' 'move west,' and 'stay in the current point,' while the Brownian target was prohibited from staying in its current position. The results of the trials for three types of target movement and three different sizes of the searcher's observed areas are as follows [12]:

Type of target movement	Radius: $r = 1$ observed area: $3 \times 3 = 9$ points		Radius: $r = 2$ observed area: $5 \times 5 = 25$ points		Radius: $r = 3$ observed area: $7 \times 7 = 49$ points	
	Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation
Static target	125.4	86.6	85.5	57.9	78.2	51.6
Markovian target	510.8	639.1	254.6	294.7	155.4	164.6
Brownian target	563.0	667.3	268.5	344.9	158.8	175.3

As expected, from the simulation results it follows that the average number of search steps strongly depends on the radius of the observed area; however, both for static and for moving targets this dependence is not linearly proportional. In addition, it can be seen that the search for a Markovian target results in a lower average number of steps (at least for small observed areas) than the search for a Brownian target. This property is similar to the property of the IMTS algorithm (see Section 4.3.3).

An example of the program output that presents the status of the search is shown in Figure 5.1.

In the figure, the *square* denotes the current agent's location and the *asterisk* denotes the current target's location. A *star* stands for the estimated location of the target as calculated by the searcher on the basis of the expected probabilities. The *higher probabilities* are denoted in white and the *lower probabilities* in black. The figure in the *upper left corner* shows a map of target location probabilities, and the one in the *lower left corner* shows the agent's trajectory and a map of observed probabilities as updated by the agent. The figure in the *upper right corner* shows a map of the agent's estimations, and the one in the *lower right corner* presents the coordinates of the searcher and target.

Algorithm 5.2 can be applied for the online search both for static and for moving targets, as well as for a target that changes its movement type during the search, and can be generalized for the search by several searchers. An example of the program output that shows the status of the search by three searchers is given in Figure 5.2.

In addition, from numerical studies it is found that for search for a static target, the actions of the algorithm depend on the global estimation at all stages of the search, while for search for a moving target, the global estimations mostly affect the initial search steps.

5.3 Application of the ILRTA* algorithm for navigation of mobile robots

The challenge of mobile robot navigation arose with the development of mobile robot platforms equipped with powerful on-board controllers and a variety of sensors. This challenge has been stated clearly by Siegwart and Nourbakhsh: 'given partial knowledge about its environment and a goal position or series of positions, navigation encompasses the ability of the robot to act based on its knowledge and sensor values so as to reach its goal positions as efficiently and reliably as possible' [17]. The robot navigation process requires consideration of the intellectual behavior of the mobile robot and suitable models for automata acting in a random environment.

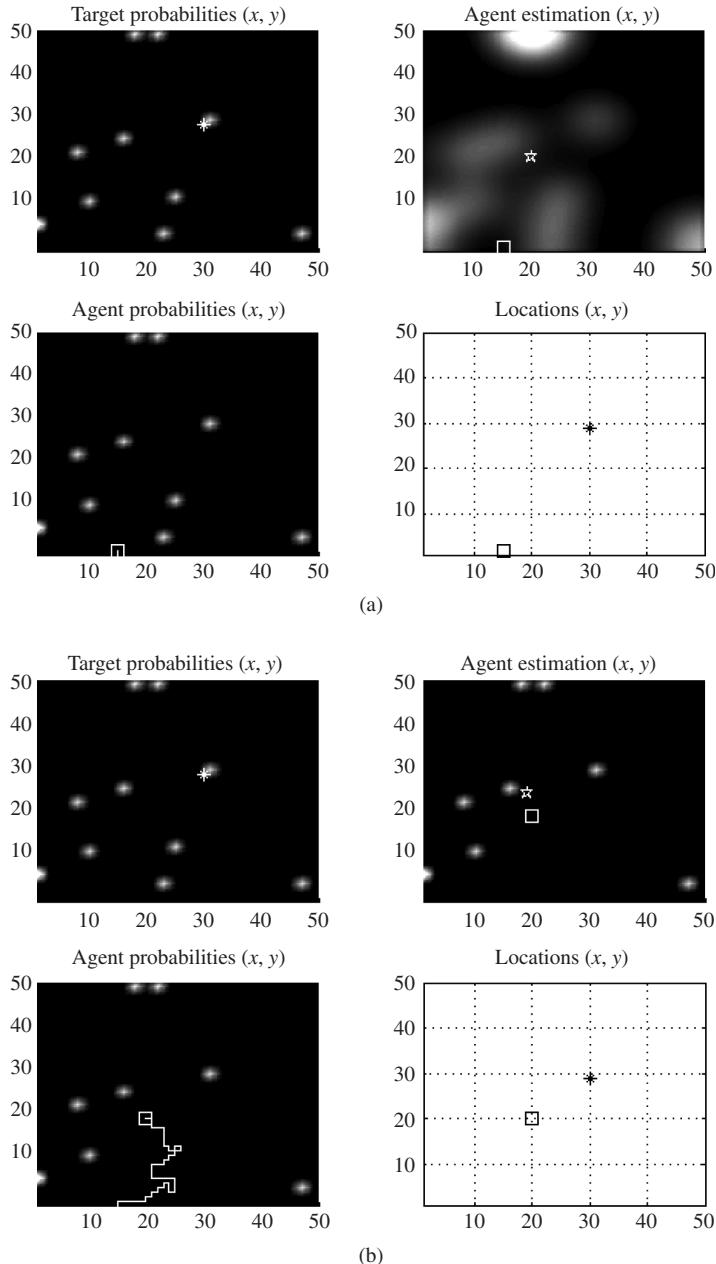


Figure 5.1 Example of the simulation frame during the search by Algorithm 5.2. (a) The states of the searcher and the target and the probabilities at time $t = 1$. (b) The states of the searcher and the target and the probabilities at time $t = 80$.

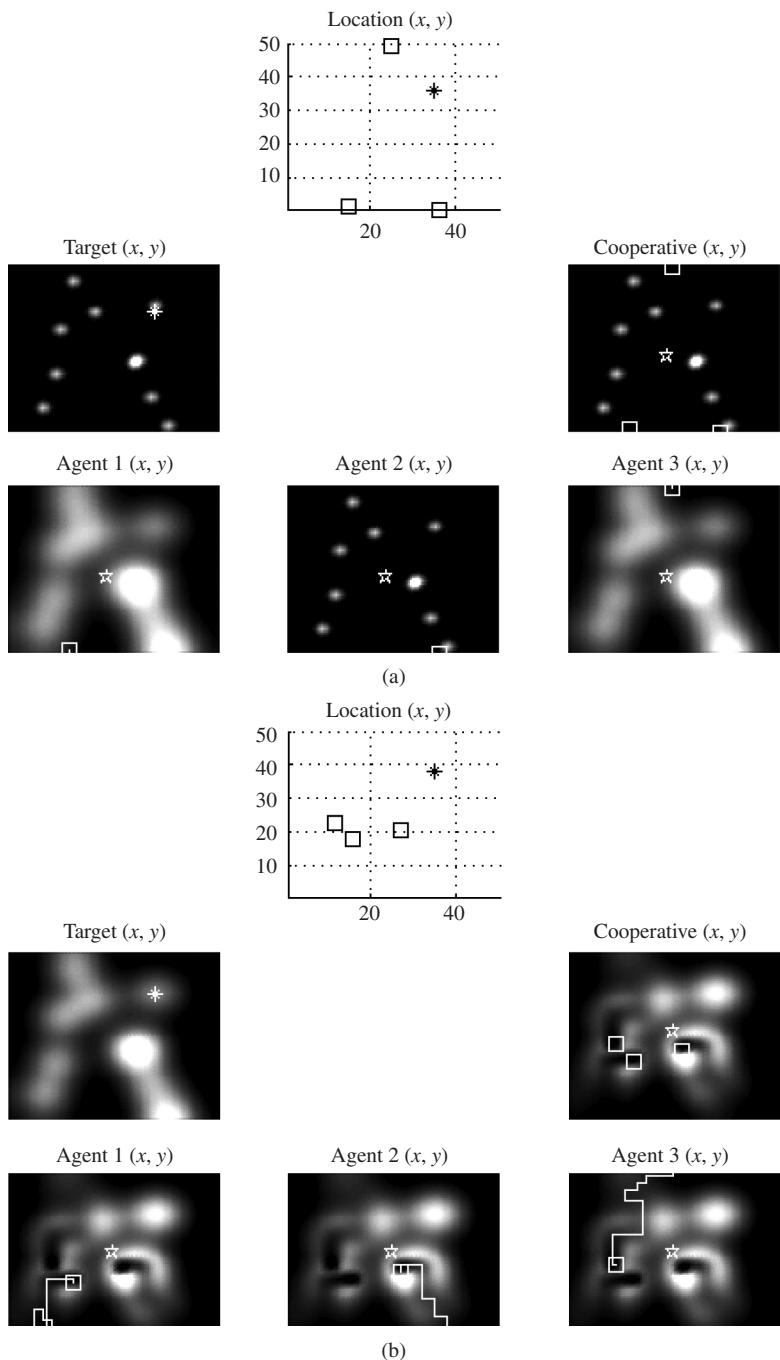


Figure 5.2 Example of the simulation frame during the search by Algorithm 5.2 with three searchers. (a) The states of the searchers and the target and probabilities at time $t = 1$. (b) The states of the searchers and the target and probabilities at time $t = 50$.

Methods for the navigation of mobile robots in a random environment consider controllers of the robots as memory-related automata enabling restoration/remembering the history of their behavior [18]. Such requirements are rather problematic both from technical and from analytical points of view.

Over the years, several methods for mobile robot navigation, including path planning based on the algorithmic manipulation of subsets of the discrete environment, have been suggested [19–22]. It has been shown that the implementation of such an approach results in near-optimal paths of robot navigation to a static target position. For a moving target, the required considerations include algorithmic models that can act on the inner states of the robot's controller, and follow the same search model as the algorithm for the robot's navigation [20].

An application of the search model to mobile robot navigation in a stochastic environment has been motivated by various studies [23–25] that apply automated mobile vehicles for search.

Assume that the controller of a mobile robot is represented by a discrete time Mealy automaton with a set X of inputs x , set S of states s , set Y of outputs y , and characteristic functions f and g . For a deterministic automaton, characteristic functions are defined as $f : X \times S \rightarrow Y$ and $g : X \times S \rightarrow S$; thus $y^t = f(x^t, s^t)$, $s^{t+1} = g(x^t, s^t)$, where $x^t \in X$, $y^t \in Y$, and $s^t, s^{t+1} \in S$, $t = 0, 1, 2, \dots$. In the case of a non-deterministic automaton, the usual approach assumes that the transition function g is a *multi-valued* function, which is equivalent to the mapping $G : X \times S \rightarrow 2^S$. For a mobile robot, inputs x are vectors of sensor values, and y are vectors of outputs to the robot's manipulators; so $x = (x_1, x_2, \dots, x_l)$, where l is the number of sensors, and $y = (y_1, y_2, \dots, y_l)$, where n is the number of manipulators. In the simplest case, $l = 1$, $n = 1$, $x = x_1 = \{0, 1\}$ corresponds to a two-state sensor like the touch sensor, and $y = y_1 = \{0, 1\}$ corresponds to a switch, for example, the motor power switch.

Let us further assume that the characteristic functions f and g are *single-valued* while the *structures* of sets X and S , for example, their metrics, change in time. Changing the structure of X represents a change in the environment as observed by the robot, and changing the structure of S corresponds to learning and adaptation of the robot to the environment.

Since the values of the sensors change along with the robot's movement, the input vectors form a sequence $\langle x^{t=0}, x^1, x^2, \dots \rangle$, starting from the initial vector $x^{t=0}$. Assume that at time t strict values of the sensors are known. Moreover, assume that given a vector x^t there is a probability $P(x^{t+1}|x^t)$ that a certain vector x^{t+1} will be obtained at time $t + 1$. Thus, the inputs form a Markov process $X = \langle x^{t=0}, x^1, x^2, \dots \rangle$ while $\langle x^{t=0}, x^1, x^2, \dots \rangle$ is a realization of this process. Consider the sequence of states $\langle s^{t=0}, s^1, s^2, \dots \rangle$ with the initial state $s^{t=0}$. Since there is a deterministic transition function $g : X \times S \rightarrow S$, while inputs x^t are governed by a Markov process, the states also form a Markov process $S = \langle s^{t=0}, s^1, s^2, \dots \rangle$. In the case of a trivial automaton, the process S with $s^{t+1} = g(x^t)$ is known as the hidden Markov model [26] (see Section 3.2). Note that if, in contrast, $s^{t+1} = g(s^t)$, then the states form a deterministic sequence. Finally, consider a process $Y = \langle y^{t=0}, y^1, y^2, \dots \rangle$ which is formed by the output vectors. According to the definition of the function $f : X \times S \rightarrow Y$, the process with the states (x^t, s^t) is a superposition of two Markov processes X and S , and, in general, it is not Markovian [26]. Thus, the mobile robot's behavior is described by three stochastic processes: an input Markov process X , a hidden Markov process S for which $s^{t+1} = g(x^t, s^t)$, and an output process Y for which $y^t = f(x^t, s^t)$.

The *navigation problem* is formulated as follows. Given a process \widehat{Y} , find a process S and functions f and g such that for each permitted input process X , the output process Y is in some sense equal to \widehat{Y} . In other words, it is required to find such a behavior of the mobile robot that, in spite of the fluctuations in the sensor values and of movement errors, the robot will follow an expected trajectory.

Let us consider the transition function g and define an equivalence relation on the set $X \times S$ with respect to the function g . We can say that the pairs (x, s) and (x', s') from $X \times S$ are equal with respect to g , if g maps them both to the same state, that is, if $g(x, s) = g(x', s')$. Accordingly, since there are a finite number M of distinguished realizations of the inner states and the pairs from $X \times S$ may exist, such that the map g is not defined, the set $X \times S$ is split into a maximum of $M + 1$ subsets called the *equivalence classes*. Denote these classes by A^g . The obtained equivalence classes form a partition of the space $X \times S$. Then, assuming that the function f and its inverse are continuous, the partition of the space $X \times S$ generates a cover of the set Y . We also assume that the inverse function f^{-1} is continuous, and that the cover of Y , generated by the partition of $X \times S$, can also be partitioned.

By the same token, let us define the partitioning of the space $X \times S$ with respect to the function f . We say that the pairs (x, s) and (x', s') from $X \times S$ are equal with respect to f , if f maps both to the same output, that is, if $f(x, s) = f(x', s')$. The equivalence classes with respect to f are denoted by A^f .

Denote by $\alpha^g = \{A_0^g, A_1^g, A_2^g, \dots\}$ and by $\alpha^f = \{A_0^f, A_1^f, A_2^f, \dots\}$ the partitions of $X \times S$ as generated by the characteristic functions g and f , correspondingly. Let $p^t(y) = \Pr\{y^t = y\}$ be the probability that at some time t the output appears as a value $y \in Y$. Then, the probability associated with the equivalence class $A^f(y; t) = f^{-1}(y)$ is $p^t(y)$, that is, $p(A^f(y, t)) = \Pr\{A^f(t) = A^f(y; t)\} = p^t(y)$. Since $A^f(y; t)$ consists of the pairs $(x, s) \in X \times S$, this equivalence implies that $\sum_{(x,s) \in A^f(y;t)} p^t(x, s) = p^t(y)$. In the same manner, let $p^t(s) = \Pr\{s^t = s\}$ be the probability that at time t the inner state is s , $s \in S$. Then, the probability associated with the equivalence class $A^g(s; t) = g^{-1}(s)$ is $p(A^g(s, t)) = \Pr\{A^g(t) = A^g(s; t)\} = p^t(s)$ and $\sum_{(x,s) \in A^g(s;t)} p^t(x, s) = p^t(s)$.

By using the defined probabilities of the equivalence classes $A^f(y; t)$ and $A^g(s; t)$, the difference between the considered processes can be measured. In particular, let T be a finite period, $T > 1$, such that $t = 0, 1, 2, \dots, T - 1$, and denote by α_t^f and α_t^g the corresponding partitions obtained at time t . The difference between the partitions α_t^f and α_t^g over the period T is defined by the Ornstein distance as follows [15]:

$$d_{Orn}(\{\alpha_t^f\}_0^{T-1}, \{\alpha_t^g\}_0^{T-1}) = \frac{1}{T-1} \sum_{t=0}^{T-1} d_{Orn}(\alpha_t^f, \alpha_t^g),$$

where $d_{Orn}(\alpha_t^f, \alpha_t^g)$ for each time t is defined in the usual manner (see Section 4.1).

The Ornstein distance provides another means of specifying the distance between the probability measures that were defined by these partitions. Then, since the probabilities of the partition classes A^f and A^g depend on the probabilities of the inner states $p(s)$, the goal is to find such a process S that the distance $d_{Orn}(\{\alpha_t^f\}_0^{T-1}, \{\alpha_t^g\}_0^{T-1})$ reaches its minimum in a given period T . Note that in the simplest case, where the transition function g is single-valued and continuous, and $\alpha^g = \alpha^f$ for all realizations of the processes X and Y , the output process Y is unambiguously determined by the input process X and does not depend on S .

An application of the search algorithms for solving this navigation problem is now straightforward. If both processes X and Y are stationary, an optimal process S which defines the internal states of the robot can be obtained by using the ILRTA* algorithm, Algorithm 4.1 (see Section 4.2). This algorithm implements the Ornstein distances between the partitions and is aimed at the target partition which is defined by the desired process \widehat{Y} . If, in contrast, processes X or Y or both are non-stationary, the navigation problem can be solved by using the IMTS algorithm, Algorithm 4.5 (see Section 4.3), which acts on a random domain.

It is found that within the framework of robot navigation and mobile robot control, these search algorithms are equivalent to those based on reinforcement learning [27]. Hence, such algorithms form a unified framework for task planning and control of mobile robots.

5.4 Application of the IMTS algorithm for paging in cellular networks

The problem of paging arises when the cellular network server has to redirect an incoming call to its relevant destination – a specific cellular telephone (also called a mobile station). Since the cellular telephone moves constantly over the network cells, the server has imperfect information regarding the telephone's location, and is required at the time of the call to find the cell in which the cellular telephone is located. The process of searching for the cellular telephone over the cellular network is called *paging*. Within the framework of the search problem, the cellular telephone can be considered as a moving target.

A cellular network is composed of a network of cells that are determined by their *base stations*. A base station is the lowest level *stationary* element of the network. It is in direct radio communication with the *mobile stations* – the cellular telephones – in the cell. The cellular telephone moves over the cells and, according to implemented communication protocols, information about its location is known only at certain times. This information is used later by the cellular telephone when it initiates communication. To initiate communication by the network server, it needs to know the exact location of the cellular telephone. The process of obtaining such information is called *location management*.

Since the most costly operation during the location management procedure is the wireless signal transmission, often the main goal of location management schemes is to decrease the time of the wireless communication. In addition, since the scanning process conducted by the cellular telephone consumes battery power, it is desirable to minimize the time of this process.

The problem of *location update* was formulated in 1995 by Bar-Noy *et al.* [28], who considered three variants of location update:

- **Time-based update:** the mobile station initiates the update process for each T time unit.
- **Movement-based update:** the mobile station initiates the update process for each M movement between the cells.
- **Distance-based update:** the mobile station initiates the update process when the distance between the current cell and the cell in which the last update was conducted is equal to a predefined number D .

By considering a realistic Markovian movement of the mobile station over the cells, Bar-Noy *et al.* [28] proved that the best results in the sense of cost reduction are achieved by the distance-based update scheme, while the worst result is obtained by the time-based update scheme.

In real cellular networks, a distance-based update scheme is realized by introducing stationary *location areas*, which are predefined sets of cells served by a certain mobile switch server. Such a scheme requires the mobile station to initiate a location update process when it moves from one location area to another. A location area in this case is considered as a meta-cell and the paging process is initiated based on the information about the location of the mobile station in the location area. A location-updating scheme with movements of a mobile station among the location areas is illustrated schematically in Figure 5.3.

In spite of the satisfying relation between the update and paging costs that is obtained by such a scheme, the realization of location management with stationary location areas requires strong computational capacities of the network and the resources for data storage and processing.

In recent years, several methods have been suggested for intelligent location updating and paging [29–34], which are based on the estimated probabilities of the mobile station locations, though some rely on information theory methods.

As above, denote by $X = \{x_0, x_1, x_2, \dots, x_n\}$ the set of cells in the network, and let x be the current location of a mobile station from the network's point of view. If the mobile

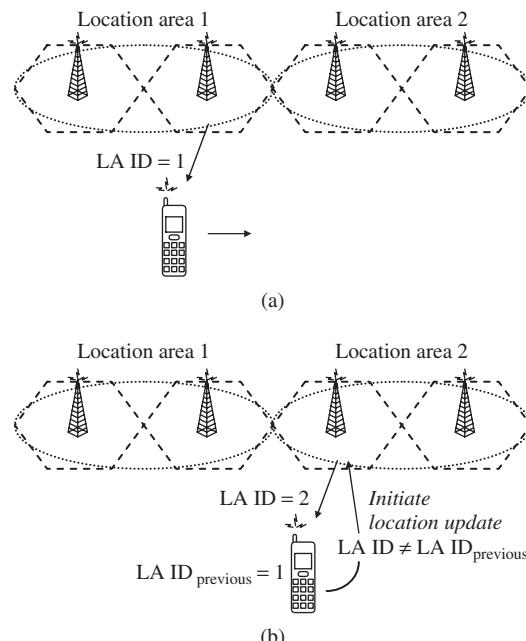


Figure 5.3 Location updating initiated by a mobile station. (a) Mobile station is in location area 1 and moves to location area 2. (b) Location updating when mobile station arrives at location area 2.

station is switched off and does not update its location, the network stores its last location as the current location.

Let $p_i = p(x_i) = \Pr\{x = x_i\}$, $x \in X$, be the location probability distribution of the mobile station over the cells x_i , $i = 0, 1, 2, \dots, n$, and $\sum_{i=0}^n p(x_i) = 1$. If the network management system has certain information regarding the location of the mobile station (at cell x_i), then the corresponding location probability satisfies $p(x_i) = 1$. If, on the contrary, the network management system for some reason does not know the location of the mobile station, then the probabilities are greater than 0. In the case of a static location updating scheme and simultaneous paging, the location probabilities have constant uniform values $p(x_i) = 1/(n+1)$, $i = 0, 1, 2, \dots, n$, where for technical reasons an additional fictive cell x_0 is introduced.

In the case of an intelligent updating scheme, the movement of the mobile station is taken into account as follows. Denote by $t = 0, 1, 2, \dots$ the times at which the location of the mobile station is determined. As above, the location of the mobile station at time t is denoted by $x^t \in X$, and the probability that at time t the mobile station is located in the cell x_i is denoted by $p^t(x_i) = \Pr\{x^t = x_i\}$. In general, it is assumed that the movements of the mobile station are governed by a stationary time-invariant Markov process. Such a representation corresponds with the general accepted assumption that the mobile station's movement maintains a certain 'pattern' in its behavior. For actual cellular networks, such a model is rather realistic, yet it requires computing and storing the transition probabilities and the stationary distributions of the location probabilities of all the telephones. This task may be achieved by applying continuous learning algorithms to all profiles of registered mobile stations. These learning procedures are very costly and are not used in current network management procedures. Alternatively, one can use a method of paging that does not assume any a priori knowledge regarding the location probabilities. The suggested IMTS algorithm, Algorithm 4.5 (see Section 4.3.1), can be considered as an example of such methods.

In the case of cellular networks, Algorithm 4.5 acts on the partition space χ that is already defined by the location areas described above. In fact, denote by $L_j \subset X$, $j = 0, 1, 2, \dots, m \leq n$, location areas that include cells x_i . Since in real networks the location areas L_j are non-overlapping and non-empty, they form a partition $\alpha = \{L_j | L_j \subset X\}$. This partition can be considered as a generic partition of the partition space χ and the other partitions are formed from it by joining location areas L_j and L_k from partition α , and then by joining elements from the obtained partitions.

Additional information regarding the location probabilities and the mobile station's behavior can be obtained by using distance estimations and variable radius r that allows more effective search.

Based on the above considerations of the paging procedure in cellular networks, it is evident that this problem, as a problem of search for a moving target, is suitable for a direct implementation of the suggested IMTS algorithm. The exact procedure and comparative results, with respect to other paging methods, is a subject of a future research.

5.5 Remark on application of search algorithms for group testing

As explained in previous chapters, the formulation of search for a hidden target in a discrete domain can represent many group-testing applications. For example, consider a target (e.g.,

a non-conforming unit) which is located somewhere within a discrete domain, and a searcher (testing agent) that is looking for this target. At each step, the action available to the searcher is to check a subset of units in order to determine whether the target is located somewhere within this subset or not. The procedure terminates if the searcher finds the target in a subset that contains only a single unit. That is, if the searcher finds the target with certainty. The searcher's goal is to choose a sequence of subsets such that the search terminates in a minimal expected number of steps [35, 36]. Such an example is found in Herer and Raz [37] and was motivated by the pasteurized food industry, and pertains to the process of filling plastic cups with cottage cheese drawn from a large container. In case the concentration of curd in the container deviates from the given specifications, all the cottage cheese cups filled after this point in time are regarded as non-conforming units. Moreover, the authors indicate that since it is not practical to wait for the laboratory results in mid-batch, all the product in the container is used to fill up the cups, which are stamped with the processing time and held until the test results are ready. If one or more of the cups sampled show an unacceptable concentration of curd, then both it and all cups that were produced afterward are discarded. If one or more of the cups sampled show that the level of curd is within the acceptable range, then both it and all cups that were produced before it are accepted. Thus the key question in Herer and Raz [37] is how many and which product units (cups) should be sampled to find the first non-conforming unit at minimal cost? The equivalence to group testing is clear since each tested cup represents all the untested cups that were produced before it up to the last tested cup. Note that the testing of several cups can be conducted simultaneously. Moreover, the results can be represented by more than a binary 'conforming' and 'non-confirming' result that can also be prone to observation noise. These cases are considered in the proposed model that was discussed in previous chapters.

References

1. T.D. Larose (2005). *Discovering Knowledge in Data: An Introduction to Data Mining*. John Wiley & Sons, Inc.: Hoboken, NJ.
2. L. Hyafil, R.L. Rivest (1976). Constructing optimal binary decision trees is NP-complete, *Information Processing Letters*, **5** (1), 15–17.
3. J.R. Quinlan (1986). Induction of decision trees, *Machine Learning*, **1**, 81–106.
4. J.R. Quinlan (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers: San Francisco, CA.
5. L. Breiman, J. Friedman, R. Olshen, C. Stone (1984) *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA.
6. P.A. Chou (1991). Optimal partitioning for classification and regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13** (4), 340–354.
7. U.K. Sarkar, P. Chakrabarti, S. Ghose, S.C. DeSarkar (1994). Improving greedy algorithms by look-a-head search. *Journal of Algorithms*, **16**, 1–23.
8. I. Ben-Gal, E. Kagan, and N. Shkolnik (2008) Constructing classification trees via data mining and design of experiments concepts. Proceedings of ENBIS'8, Athens, Greece, September 21–25, 2008, e-publ.
9. I. Ben-Gal, N. Shkolnik, and E. Kagan (2007) Greedy learning algorithms and their applications to decision trees. Proceedings of the 7th European Annual Conference ENBIS'07, Dortmund, Germany, September 24–26, 2007, p. 36, e-publ.

10. J.-Y. Lee, S. Olafsson (2006). Multi-attribute decision trees and decision rules. In E. Triantaphyllou, G. Felici (eds) *Data Mining and Knowledge Discovery Approaches Based on Rule Induction Techniques*, Springer: Heidelberg, 327–358.
11. N. Shkolnik (2008) Constructing of classification trees by the dual information distance method. MSc Thesis. Tel-Aviv University, Israel.
12. E. Kagan, G. Goren, and I. Ben-Gal (2010) Probabilistic double-distance algorithm of search after static or moving target by autonomous mobile agent. Proceedings of IEEE 26th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, November 17–20, 2010, pp. 160–164.
13. G. Chernikhovsky, E. Kagan, G. Goren, and I. Ben-Gal (2012) Path planning for sea vessel search using Wideband Sonar. Proceedings of IEEE 27th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, November 14–17, 2012, DOI 10.1109/EEEI.2012.6377122.
14. E. Kagan, G. Goren, and I. Ben-Gal (2012) Algorithm of search for static or moving target by autonomous mobile agent with erroneous sensor. Proceedings of IEEE 27th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, November 14–17, 2012, DOI 10.1109/EEEI.2012.6377124.
15. D.S. Ornstein (1974). *Ergodic Theory, Randomness, and Dynamical Systems*. Yale University Press: New Haven, CT.
16. T.M. Cover, J.A. Thomas (1991). *Elements of Information Theory*. John Wiley & Sons, Inc.: New York.
17. R. Siegwart, I.R. Nourbakhsh (2004). *Introduction to Autonomous Mobile Robots*. The MIT Press: Cambridge, MA and London.
18. C. Unsal (1998) Intelligent navigation of autonomous vehicles in an automated highway system: learning methods and interacting vehicles approach. PhD Thesis. Virginia Polytechnic Institute and State University, Blacksburg, VI.
19. Z. Bnaya, A. Felner, E. Shimony, G.A. Kaminka, and E. Merdler (2008) A fresh look at sensor-based navigation: navigation with sensing costs. The 2nd Israeli Conference on Robotics, Herzlia, Israel, November 19–20, 2008, e-publ.
20. E. Kagan and I. Ben-Gal (2008) Application of Probabilistic self-stabilization algorithms to the robot's control. Proceedings of 15th Industrial Engineering and Management Conference IE&M'08, Tel-Aviv, Israel, March 10–11, 2008, p. 3e, e-publ.
21. N. Pochter, A. Zohar, and J.S. Rosenschein (2008) Using swamps to improve optimal path-finding. The 2nd Israeli Conference on Robotics, Herzlia, Israel, November 19–20, 2008, e-publ.
22. X. Zhuang (2005) The strategy entropy of reinforcement learning for mobile robot navigation in complex environments. Proceedings of IEEE Conference on Robotics and Automation, Barcelona, Spain, 2005, pp. 1742–1747.
23. M. Goldenberg, A. Kovarsky, X. Wu, and J. Schaeffer (2003) Multiple Agents Moving Target Search, Department of Computer Science, University of Alberta, Edmonton.
24. H. Lau, S. Huang, and G. Dissanayake (2005) Optimal Search for Multiple Targets in a Built Environment, ARC Centre of Excellence for Autonomous Systems (CAS), University of Technology, Sydney.
25. A. Roy, S.K. Das Bhaumik, A. Bhattacharya, K. Basu, D.J. Cook, S.K. Das (2003) Location aware resource management in smart homes. Proceedings of the First International Conference on Pervasive Computing and Communications (PerCom'03).
26. Y. Ephraim, N. Merhav (2002). Hidden Markov processes. *IEEE Transactions on Information Theory*, **48** (6), 1518–1569.
27. M.M. Lomas (2006) Reinforcement learning for mobile robot controllers: theory and experiments. PhD Thesis. University of Pennsylvania, Philadelphia, PE.

28. A. Bar-Noy, I. Kessler, M. Sidi (1995). Mobile users: to update or not to update? *Wireless Networks*, **1** (2), 175–185.
29. A. Bhattacharya and S.K. Das (1999) LeZi-update: an information-theoretic approach to track mobile users in PCS networks. Proceedings of the 5th Annual ACM/IEEE Conference on Mobile Computing and Networking MobiCom'99, pp. 1–12.
30. G.D. Caro (2004) Ant colony optimization and its application to adaptive routing in telecommunication networks. PhD Thesis. Université Libre de Bruxelles, Belgium.
31. J. Cowling (2004) Dynamic location management in heterogeneous cellular networks. BSc Thesis. University of Sydney, Sydney.
32. D. Kouvatssos, S.A. Assi, I.-H. Mkwawa, V.C. Giner, H. DeMeer, A. Houyou (2005) An information-theoretic approach to mobility location management: an overview. The 3rd International Working Conference HET-NETs'05, Ilkley, West Yorkshire, July 18–20, 2005.
33. B. Krishnamachari, R.-H. Gau, S.B. Wicker, Z.J. Haas (2004). Optimal sequential paging in cellular networks. *Wireless Networks*, **10**, 121–131.
34. Y. Xiao, Y. Pan, J. Li (2004). Design and analysis of location management for 3G cellular networks. *IEEE Transactions on Parallel and Distributed Systems*, **15** (4), 339–349.
35. E. Kagan and I. Ben-Gal (2013) A Group-Testing Algorithm with Online Informational Learning. To appear in IIE Transactions.
36. E. Kagan and I. Ben-Gal (2013) Moving Target Search Algorithm with Informational Distance Measures. To appear in Entropy.
37. Y.T. Herer, T. Raz (2000). Optimal parallel inspection for finding the first nonconforming unit in a batch – an information theoretic approach. *Management Science*, **46** (6), 845–857.

6

Final remarks

The problem of optimal probabilistic search for targets is as ancient as the human race: it started with prehistoric hunter-gatherers trying to figure out the best location to catch their prey. The problem was first formulated mathematically in the eighteenth century in its simplest form and since then has been revised and regenerated in various forms, orientations, complexities, and applications. It was formulated both as a search in discrete space and time as discussed in this book, and as a search in continuous space and time (see, e.g., [1–3]) or as a visibility analysis problem [4, 5]. Yet, despite the huge amount of research that has been invested and the large body of literature that addresses it, this problem has remained unsolved to date: even for a marginal search space there is no clear answer for what the optimal search policy should be to guarantee a minimal search time until the target is found. Moreover, for each set of assumptions, such as incomplete information, ability to group-test, a noisy observation, or the possibility of a moving target, different suboptimal algorithms are used. Taking into account the computational complexity involved in the problem, one can speculate that the possibility to find optimal solutions to practical realistic search problems decreases over the years as the search applications grow rapidly in complexity and size.

The information age is characterized by the ability of individuals to gain instant access to information that would have been impossible to find previously. The huge amount of data, for example, on Internet users, e-commerce and financial transactions, mobile telephones, location-based services, sensors, and radio frequency identification (RFID) tags, increases the potential search space and poses huge computational obstacles and challenges for various search applications. At the same time, with the overall developments in storage capacity, sensor usage, and IT, there is a growing need for new monitoring and search techniques that can cope with complex and data-rich environments. Big Data analytics, Business Intelligence, Client Targeting, Client Segmentation, Online Advertisement, Promotion Management, Next Best Offering, and Complex Event Processing are a few examples of new business methodologies and concepts that monitor a large amount of data and can use good search algorithms for their implementation. To follow one such application, in

Client Targeting the search problem focuses on finding the right targeted customer over a multi-dimensional space, where coordinates represent various user features that can profile and segment the potential customer. Some of these features are age, habits, demographic information, purchase history, ratings and grading, social and credit information, and more. Although in this book we mainly considered search over a two-dimensional (often geographic) space, many of the presented algorithms, including the proposed Informational Learning Real-Time A* (ILRTA*) and the Informational Moving Target Search (IMTS) algorithms, can be applied to search over such a multi-dimensional space by applying minor changes to them. This is an important feature to mention, as it applies to many of the modern applications that are discussed above.

The theory of group testing can also be implemented in new exciting research areas, such as gene network analysis, data mining, and predictive maintenance in cloud computing applications. For example, mining data items or records with specific features in large databases is an active research and practical area that can benefit from group-testing approaches. In these cases, the tested group is represented by those items (records) that share some similar features, such that a database query can identify all the items in a group, while the searched target can be a record with specific features (see, e.g., [6, 7]). Sensor-based location management in service applications provides a lot of ground to find objects with the same sensor signature. Similarly, in modern cloud computing environments, a group-testing approach could be of great benefit for predictive maintenance. Cloud computing is the use of computer resources (hardware and software) that are delivered as a service over a network (typically the Internet) remotely. Thus, group testing can be used to identify specific faulty modules (sometimes associated with remote services) that are stored in the cloud by using I/O testing of different groups of modules, until the faulty module is identified.

The explosion of data and specifically the use of the Internet and social networks create a revolutionary phenomenon where the statistics gathered on people's activities are no longer sporadic or based on small polls or questionnaires. Automated Big Data applications continuously collect huge amounts of information that provide a better ground for implementing successful search policies inspired also by continuous space and time search methods [8–11]. In other words, although the optimal probabilistic search problem remains computationally intractable, the implementation of good search policies on exponentially growing populations can guarantee better and better results.

Perhaps one analogy can be made to statistical mechanics, as a branch of physics that applies probability theory to deal with a large population of particles. There, the deterministic thermodynamic behavior of systems that are composed of a large enough number of particles is guaranteed, despite the probabilistic nature of the behavior of each single particle by itself. Similarly, in modern probabilistic search, deterministic results could be guaranteed when it is applied to large enough populations. Indeed, exciting times are yet to come!

References

1. O. Hellman (1972). On the optimal search for a randomly moving target. *SIAM Journal on Applied Mathematics*, **22**, 545–552.
2. M. Lukka (1977). On the optimal searching tracks for a moving target. *SIAM Journal of Applied Mathematics*, **32** (1), 126–132.

3. A. Ohsumi (1991). Optimal search for a Markovian target. *Naval Research Logistics*, **38**, 531–554.
4. M. Yadin, S. Zacks (1988). Visibility probabilities on line segments in three-dimensional spaces subject to random poisson fields of obscuring spheres. *Naval Research Logistics*, **35**, 555–569.
5. S. Zacks, M. Yadin (1994). The survival probability function of a target moving along a straight line in a random field of obscuring elements. *Naval Research Logistics*, **41**, 689–70.
6. I. Ben-Gal (2004). An upper bound on the weight-balanced testing procedure with multiple testers. *IIE Transactions*, **36**, 481–493.
7. J.S. Vitter (1987). Design and analysis of dynamic Huffman codes. *Journal of the ACM*, **34** (4), 825–845.
8. G. Chernikhovsky, E. Kagan, G. Goren, and I. Ben-Gal (2012) Path planning for sea vessel search using Wideband Sonar. Proceedings of IEEE 27th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, November 14–17, 2012, DOI 10.1109/EEEI.2012.6377122.
9. M. Israel, E. Khmelnitsky, and K. Kagan (2012) Search for a mobile target by ground vehicle on a topographic terrain. Proceedings of IEEE 27th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, November 14–17, 2012, DOI 10.1109/EEEI.2012.6377123.
10. E. Kagan, G. Goren, and I. Ben-Gal (2012) Algorithm of search for static or moving target by autonomous mobile agent with erroneous sensor. Proceedings of IEEE 27th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, November 14–17, 2012, DOI 10.1109/EEEI.2012.6377124.
11. E. Kagan, G. Goren, and I. Ben-Gal (2010) Probabilistic double-distance algorithm of search after static or moving target by autonomous mobile agent. Proceedings of IEEE 26th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, November 17–20, 2010, pp. 160–164.

Index

- Action, 6, 8–9, 12–14, 58, 68, 101, 104, 146–9, 152, 155, 162–6, 191, 193, 202–203, 210, 238, 242, 313
- Actions
set of, 12–13, 101, 147–9, 150, 152–3, 162, 201
- Algorithm
 A^* , 116–18, 121
binary splitting, 60–61, 67–68, 70, 84–85
Brown (moving target search), 47, 49, 55, 184
'division by half' search, 86, 242
 FRA^* (fringe-retrieving A^*), 139
GOTA (generalized optimal testing algorithm), 103–5, 241
over partitions space, 241
Graff and Roeloffs
(group-testing), 79, 81
Hwang (generalized binary splitting), 67–68, 70
Huffman-Zimmermann search, 95
over partitions space, 237
by multiple searchers, 251
 $ILRTA^*$ (informational $LRTA^*$), 230, 231, 233, 310
IMTS (informational MTS), 255, 258–9, 310, 312
- $LRTA^*$ (learning real-time A^*), 125–7
min-max $LTRA^*$, 213, 227
MTS (moving target search), 132–4
on-line search for static and moving targets, 300, 304–5, 307
recursive $ILRTA^*$, 296, 299
 RTA^* (real-time A^*), 124
Sobel and Groll (group-testing), 74, 76, 81
Stone (allocation for static target search), 40–41, 44, 46–47, 185
Washburn (moving target search), 53, 184
- Allocation, 31–42, 44–48, 52, 54, 67, 100, 186
- Code, 9, 90, 95–97
length, 9, 96–97
word, 95–97, 140
- Control
function, 57, 58
set, 57, 58
- Cost
evaluated, 113–120, 122–4, 126–130, 138
evaluation function, 110, 112–13, 115–17, 122–3, 125, 132, 137, 228

- Cost** (*continued*)
 expected, 76–78, 175–180, 195–6,
 202, 206, 215
 function, 33, 195, 241
 total, 108, 242
- Decision rule**, 152, 156, 269
- Density**,
 search, 24, 31–32, 36–38, 41–50,
 52–55, 193
 target location, 24–26, 28–30, 37,
 38, 47, 48, 53–55, 147,
 153, 193
- Detection function**, 6–9, 21–24, 30, 32,
 35, 39, 42–43, 46, 49, 52–53,
 185, 186, 189, 300
- Division rule**, 101–108, 294
- Distance**, 13–15, 198, 206–7, 210–211,
 218, 231, 239, 248–9, 255–6,
 283–5, 287–290, 297, 303–4,
 310–311
 actual, 228, 230, 233, 237
 estimated, estimations, 6, 115,
 229–230, 233, 237–8,
 255–6, 258–260, 263,
 266–8, 296–7, 300, 312
 Euclidean, 135
 Hamming, 220, 223–4, 279, 283–5
 informational, 221
 Manhattan, 198, 303
 Ornstein, 14, 220, 222–5, 283–5,
 301–2, 309
 Rokhlin, 13–15, 218–230, 235, 240,
 248–250, 254–5, 283–5
 weighted, 244
- Entropy**, 5, 57, 60, 63, 88, 97–99, 156,
 157, 160, 166, 214, 219, 220,
 226, 229, 236, 242, 246, 255,
 269, 271, 287–9, 295, 302
 of partition, 99–100, 105, 108,
 220–221, 226, 235, 246,
 248–9, 250
 conditional, 102, 105–7, 220,
 221, 223, 226, 236,
 238
 joint, 222
- History of detections (observations), 162,
 183
- Information**,
 mutual, 157, 222
 vector, 163–5, 202
- Neighborhood**, 5, 6, 11–13, 160, 283,
 284, 299
 in the Best-First* (BF*)
 algorithm, 110, 111
- in the Branch and Bound
 algorithm, 185, 188, 190,
 191
- in the Informational Learning
 Real-Time A* (ILRTA*)
 algorithm, 228–232, 234,
 255, 296–7
- in the Informational Moving Target
 Search algorithm, 255–8,
 260, 266–7
- in the Real-Time search
 algorithm, 211–13
- GOTA**, 241–5
- Huffman**, 160, 237–240
- Observation**
 function, 21, 23, 56, 57, 152, 193,
 194, 300
 result, 21, 43, 55–60, 62–65, 67, 70,
 72–73, 75–76, 78–81,
 83–84, 95, 152–5, 158–9,
 161, 166–7, 176, 183, 194,
 201, 203–5, 214, 246,
 299–301
- Observations**
 expected number of,
 in the Dorfman group-testing
 search, 63, 64, 66
 in generalized binary splitting
 (Hwang's
 algorithm), 68, 70–71
 in the Pollock model of
 search, 167–71,
 173–5
 in the Ross model of search, 175
 in the Sobel and Groll group
 testing search, 73

- in the Sterrett group-testing search, 65, 66
- Observed area, 16, 21–24, 30, 31, 37, 56–59, 63, 65, 66, 71–77, 79–81, 83–85, 100, 146, 152, 153, 155–8, 167, 174, 183–5, 193, 201, 203–5, 214, 299, 300–302, 305
- Partition**
 - initial, 230, 232, 234, 237–41, 244, 248, 255, 257, 263, 267–8, 295–6
 - final, 13, 101, 103–107, 147, 228, 230–232, 237, 239–241, 244, 284, 295, 297
- Partitions space (or set), 15, 218, 227–8, 230–231, 233–4, 237, 239–42, 244, 248–51, 254–5, 270–271, 278, 280, 282–4
- Path, 1, 4, 5, 7–8, 10, 12–13, 95, 101, 108–118, 120–134, 138–140, 180, 208–11, 213–14, 228, 229, 233, 240, 244–5, 290, 294
 - optimal, 8, 108–110, 113, 115–16, 118, 121, 123, 125–8, 131, 138–9, 188, 211, 214, 228
 - shortest, 206, 295
 - longest, 206
- Path-planning, 2, 108–110, 113, 115, 122, 140, 184, 188–9, 192, 215, 308
- Policy, 2, 10
 - greedy, 41
 - in the Ross model, 175–6
 - of Markov Decision Process, 146, 148–9, 151, 155, 163–66
 - of the searcher, search policy, 20, 23, 55, 59, 155–6, 183–4, 211, 316
 - in cumulative search-evasion game, 194–96, 198
 - in probabilistic pursuit-evasion game, 201–3
 - in pursuit-evasion game on graph, 209–210
 - in min-max LRTA*, 211–13, 214
- of the target, 211
 - in cumulative search-evasion game, 195–8
 - in probabilistic pursuit-evasion game, 202–3
 - in pursuit-evasion game on graph, 209, 210
 - in min-max LRTA*, 211–14
- Probability**
 - estimated location, 24, 30
 - of detecting the target, 7, 20–22, 24, 26, 29, 31
 - in the Branch and Bound algorithm, 190
 - in the Dorfman group-testing search, 66
 - in the Eagle model of search, 180, 183
 - in the Pollock model of search, 167–9, 171–2
 - in the Ross model of search, 174
 - in the Sobel and Groll group testing search, 72
 - in the Sterrett group-testing search, 66
 - using allocation, 32, 35
- of not detecting the target, 26–28, 43, 46, 50–52, 55, 63, 73, 74, 77, 81, 186, 188–9
- steady state or limiting, 149
- target location, 26, 30, 41, 155
- transition, 22, 26, 147, 152–4, 159, 162, 171, 174–7, 180–182, 186, 204, 261, 299–300, 302, 304
- Probabilities**
 - estimated location, 21, 30, 152, 174, 175, 180, 300
 - observed location, 152–3, 174
 - target location, 22, 28–30, 32, 35–37, 41–42, 47, 49, 53, 84, 156, 167, 170, 174, 176, 178, 180, 183–6, 191, 199–200, 204–5, 299–302, 304–5

- Probabilities (*continued*)
 transition, 24–25, 50, 52, 62, 147, 149–150, 153, 157, 159, 167, 170, 176, 201, 203, 261, 266, 300, 312
- Procedure
 Best First* (BF*)
 algorithm, 110–114, 139
 branch and bound search, 187, 189, 190
 division by half, 85–86, 94, 99–102, 108
 Dorfman group-testing search, 63
 group-testing search, 57, 67
 myopic heuristic search, 190–191
 Sterrett group-testing search, 64–65
- Reward, 148–9, 206, 247
 cumulative, 196
 discounted, 165
 expected, 149, 151, 156, 164, 184, 196, 202, 206
 immediate, instantaneous, 149, 164, 165, 184, 197, 202, 204, 205
 function, 196, 248
- Rendezvous
 expected number of, 190, 193, 194, 195
- Sample space, 1, 14, 21, 23–27, 30–31, 33–36, 39, 40, 43–45, 52, 84–86, 90–91, 93–96, 98–108, 146, 152–7, 159–160, 167–8, 174–7, 179–80, 182, 185–7, 189–90, 193, 198, 201, 203, 206, 215, 219, 221–9, 231, 234, 235, 237, 239–49, 251, 254–5, 260–263, 266–271, 276, 278, 294, 299–300, 303
- Search effort(s), 1–3, 5, 20, 22–24, 28–43, 47–49, 52–55, 185–190, 193, 195–7
 distribution, 23, 24, 29–31, 52, 55, 189
- Search space, 2, 3, 5–6, 9–11, 13–14, 19, 21, 88, 140, 152–4, 159–60, 174, 180, 227, 276, 278, 280, 284, 290, 316
- Search tree, 59–62, 84, 89–90, 94–95, 98–102, 244, 295
 of generalized binary splitting
 (Hwang's algorithm), 68, 70–71
 of the division by half procedure, 85, 242
 of the Generalized Optimal Testing Algorithm (GOTA), 102–8, 147, 235, 242, 245
 of the Graff and Roeloffs group-testing search, 81–82
 of the Informational Learning Real-Time A* (ILRTA*) algorithm, 238, 241
 of the optimal binary combination (Zimmermann procedure), 93
 of the optimal binary search (Huffman-Zimmermann algorithm), 96, 178, 235–242, 246
 of the Sobel and Groll group testing search, 74–75
- States
 set of, 12, 118, 123, 137, 148–150, 152, 162–3, 201–3, 214, 228
- System dynamics, 146–7, 161–2, 219–20
 history of, 147, 149
- Trajectory, 7
 of the search agent, 30–32, 109, 186–9, 191, 193, 299
 of the target, 43–44, 48–49, 186, 189, 193
 cumulative search-evasion game, 194, 198, 201
 of the pursuit-evasion game on graph, 207–8, 210