Ben-Gurion University of the Negev
Kreitman School for Advanced Graduate Studies

Rational Metareasoning in Problem-Solving Search

PhD Thesis

by David Tolpin
Under the supervision of Professor Solomon Eyal Shimony

May 16, 2013

An important class of Artificial Intelligence problems is problem-solving search. In problem-solving search, a single agent acts in a neutral environment to reach the goal. Many problems, such as routing and path-finding problems, finite-domain constraint satisfaction, function optimization, fit within the problem-solving search abstraction. General search algorithms capable of solving large sets of search problems are well known. Algorithm performance can be significantly improved by tuning the algorithm to a particular problem domain; however, such fine-tuned algorithms exhibit good performance only on small sets of search problems, and the effort invested in the algorithm design cannot be reused in other problem domains.

Specialized versions of general search algorithms are often created by combination and selective application of search heuristics. A human expert decides which heuristics to use with the problem domain, and specifies how the search algorithm should apply the heuristics to solve a particular problem instance. A search algorithm that rationally selects and applies heuristics would decrease the need for the costly human expertise. Principles of rational metareasoning can be used to design rational search agents.

Some rational search algorithms were designed and shown to compare to or even outperform manually tuned algorithms. However, wide adoption of rational metareasoning algorithms for problem solving search is hindered both by theoretical difficulties and by lack of problem domain specific case studies. This research aims at lifting some of the theoretical difficulties in application of the rational methodology. In particular, the problem of efficiency estimating the value of information of computational actions is considered. In addition, this research proposes rational metareasoning versions of algorithms for applications of problem-solving search in the areas of parameter tuning, constraint satisfaction, Monte-Carlo tree search, and optimal planning.

Contents

Chapter 1

Introduction

A common approach in Artificial intelligence (AI) solves computational problems by designing *agents*. Agents act in an environment, exploring and possibly modifying the environment in order to reach the *goal*: a solution of the problem. Agent behavior is determined by the *agent function* that maps the agent's knowledge about the environment to actions. AI classifies problems according to the number of agents, the type of interaction among the agents and between the agents and the environment, as well as by the *problem domain*—properties of the environment which help design efficient problem-specific agents.

A simple yet large class of problems is *problem-solving search*. In problem-solving search, a single agent acts in a neutral environment to reach the goal. Many problems, such as routing and path-finding problems, finite-domain constraint satisfaction, function optimization, fit within the problem-solving search abstraction. The problem-solving search agent's behavior is described by a *search algorithm*. The same search algorithms can be used to solve different search problems, but a general search algorithm often requires adaptation to solve a particular problem efficiently.

Computer scientists approach problem-solving search in two ways. On the one hand, general search algorithms capable of solving large sets of search problems are designed, such as A* search for path-finding problems, or the Maintaing Arc Consistency (MAC) algorithm for finite-domain constraint satisfaction [RN03]. Theoretical performance bounds can often be proved for these algorithms, and the same algorithm can efficiently solve many problem instances with little or no adaptation. On the other hand, the algorithm performance can be significantly improved by tuning the algorithm to a particular problem domain. For example, specialized algorithms for task scheduling outperform general constraint satisfaction algorithms by orders of magnitude [Wol04]. However, such fine-tuned algorithms exhibit good performance only on small sets of search problems, and the effort invested in the algorithm design cannot be reused in other problem domains.

Specialized versions of general search algorithms are often created by combination and selective application of *search heuristics*—domain-specific modifications or extensions of search algorithms. A human expert decides which heuristics to use with the problem domain, and specifies how the search algorithm should apply the heuristics to solve a particular problem instance. A search algorithm that rationally select and applies heuristics would decrease the need for the costly human expertise.

The principles of rational metareasoning [RW91] can be used to design rational search agents. A rational agent deliberates before acting. The deliberation is composed of *computational actions*. The purpose of the deliberation is to select the best *base-level action*. A base-level action affects the state of the search algorithm looking for a solution of a problem instance, for example, by a exploring the search space, or pruning a part of the search that does not contain a solution. Computational actions are selected according to their net value of information, the difference between the expected benefit and the cost of the action, and the agent deliberates only if there is a computational action with positive value of information.

Some rational search algorithms were designed and shown to compare to or even outperform manually tuned algorithms [RW91]. However, wide adoption of rational metareasoning algorithms for problem solving search is hindered both by theoretical difficulties and by lack of problem domain specific case studies. This research aims at lifting some of the theoretical difficulties in application of the rational methodology. In particular, the problem of efficiency of estimating the value of information of computational actions is considered. In addition, this research proposes rational metareasoning extensions for several search algorithms, and theoretically and empirically analyses the gain in performance due to the extensions.

The rest of the thesis is organized as follows. Chapter 2 provides the necessary background information about the rational metareasoning approach as well as about search problems and algorithms. Chapter 3 describes application of the rational metareasoning approach to search problems and discusses difficulties arising in design of search algorithms based on the approach.

Chapter 4, based on [TS12b], introduces rational computation of value of information—an important issue in design of efficient search algorithms based on rational metareasoning. Case studies of the rational metareasoning approach in several search problems are presented and evaluated in Chapter 5 (based on [TS11, TS12a, HRTS12, TBS$^+$13]). Chapter 6 concludes this thesis with a discussion of achieved results and further research directions.

Chapter 2

Background

2.1 Rational Metareasoning

While ideally programs, or agents, should *act rationally* [RN03], absolute rationality is not feasible. The computational power of the agent approaching the problem must be taken into account. Rational metareasoning [RW91] is an approach to building *bounded optimal* agents, agents which find a solution that is optimal given its computational resources [Hor87]. The approach describes a method of choosing *meta-level actions* and is based on notions of *value of information* and *time cost.*

2.1.1 Meta-level Actions

The agent performs a sequence of base-level actions drawn from a known set $\{A_i\}$. Before committing to each action, the agent deliberates, and the deliberation can be represented as a sequence of meta-level actions from a set $\{S_j\}$. At any given time there is a base-level action, A_α, that appears best to the agent. The goal of subsequent meta-level actions is to refine the choice of A_α before performing the base-level action.

The agent selects α by numericaly estimating the *utilities* of action outcomes; according to decision theory, an optimal action A_α is one which maximizes the agent's expected utility:

$$\begin{aligned}\mathbb{E}[U(A_i)] &= \sum_k P(W_k)U(A_i, W_k) && (2.1)\\ \alpha &= \arg\max_i \mathbb{E}[U(A_i)] && (2.2)\end{aligned}$$

where $\{W_k\}$ is the set of possible world states, and $P(W_k)$ is the belief probability that the world is currently in state W_k.

2.1.2 Value of Information

A meta-level action affects the choice of the base-level action A_α by changing the belief distribution over the world states. The *value* of a meta-level action is measured by the resulting increase in the utility of A_α. Since neither the outcomes of meta-level actions nor the true utility of A_α are known in advance, a meta-level action is selected according to its expected influence on the expected utility of A_α.

S_j affects the internal state of the agent and the effect of a possible further meta-level action sequence $\mathbf{T}$. Thus, the expected utility of S_j is:

$$\mathbb{E}[U(S_j)] = \sum_{\mathbf{T}} P(\mathbf{T})\mathbb{E}[U(A_\alpha^{\mathbf{T}}, S_j \cdot \mathbf{T})] \tag{2.3}$$

where $S_j \cdot \mathbf{T}$ denotes the meta-level action S_j followed by possible further meta-level action sequence $\mathbf{T}$. The *value of information* of a meta-level action S_j is the expected difference between the expected utility of S_j and the expected utility of the current A_α.

$$V(S_j) = \mathbb{E}(\mathbb{E}(U(S_j)) - \mathbb{E}(U(A_\alpha))) \tag{2.4}$$

While a perfectly rational agent would always choose the most valuable computation sequence, an agent with only limited rationality makes decisions based on an approximation of the utility.

2.1.3 Benefit and Time Cost

The general dependence on the overall state complicates the analysis. Under certain assumptions, it is possible to capture the dependence of utility on time in a separate notion of *time cost* C. Then, the utility of an action A_i taken after a meta-level action S_j is the utility of A_i taken now less the cost of time for performing S_j:

$$U(A_i, S_j) = U(A_i) - C(A_i, S_j) \tag{2.5}$$

It is customary to call the current utility of a future base-level action its *intrinsic utility.* The separation into intrinsic utility and time cost allows to estimate the utility of a base-level action in a time-independent manner, and then refine the net utility estimate according to the time pressure represented by C.

In many cases, the time cost of an internal action is independent of the subsequently taken base-level action. When C depends only on S_j, (2.4) can be rewritten with the cost and the intrinsic value of information of a computation as separate terms.

$$\begin{align}
V(S_j) &= \mathbb{E}\left(\mathbb{E}(U(A_\alpha^j, S_J)) - \mathbb{E}(U(A_\alpha))\right) \nonumber \\
&= \mathbb{E}\left(\mathbb{E}(U(A_\alpha^j)) - \mathbb{E}(U(A_\alpha))\right) - C(S_j) \tag{2.6} \\
&= \Lambda(S_j) - C(S_j) \tag{2.7}
\end{align}$$

where

$$\Lambda(S_j) = \mathbb{E}\left(\mathbb{E}(U(A_\alpha^j)) - \mathbb{E}(U(A_\alpha))\right) \tag{2.8}$$

denotes the intrinsic value of information, that is, the expected difference between the intrinsic expected utilities of the new and the old selected base-level action, computed after the meta-level action was taken.

For any particular outcome of the meta-level action, one of the following cases takes place:

1. the selected base-level action A_α stays the same; consequently, $\mathbb{E}(U(A_\alpha^j)) - \mathbb{E}(U(A_\alpha))$ is zero;

2. a different base-level action A_α^j is selected, with **higher** expected utility than the expected utility of A_α before the meta-level action was taken;

3. a different A_α^j is selected, and its expected utility is **the same** as or **lower** than the expected utility of A_α before the meta-level action was taken.

In the last two cases, the difference is positive—although the expected utility of the final choice can decrease, the latter choice appears to be better than the earlier one due to the updated knowledge about all actions. Thus, while the net value of information can be either positive or negative depending on the cost of the action, the intrinsic value of information is always non-negative.

2.2 Problem-Solving Search

Problem-solving search is characterized by a single agent acting in a neutral environment[RN03]. The ultimate goal of the agent is to select a single member from a given set of feasible solutions. The value of an *evaluation function*, possibly randomized, can be efficiently computed for any member; however, evaluating all of the members is infeasible because the set is too large, often exponential in the size of the problem instance, or even infinite.

The two common selection criteria are

Satisfaction: the evaluation function represents a *goal test* determining whether the member satisfies constraints imposed by the problem definition. The agent may choose any member satisfying the constraints.

Optimization: the evaluation function is an *utility* function returning a numeric value, and the agent must choose a member that maximizes the utility.

One strives to design an agent that arrives at the final choice in as little time as possible, or at least within reasonable time bounds. If an agent that solves any instance of the problem efficiently cannot be designed, for example, because no polynomial-time algorithm is known, the agent that is the fastest one in expectation for a certain instance distribution is preferred. An optimizing agent may approximate the solution by selecting a member which is 'good enough' while not necessarily the best one, achieving a compromise between the running time and the solution quality.

2.2.1 Problem Examples

Sliding tile puzzle: an $N \times N$ board with $N^2 - 1$ sequentially numbered tiles is given. A tile adjacent to the blank space can slide into the space. The goal is to arrange the tiles in the ascending order in as few moves as possible [RN03]. This can be either an optimization problem, in which the shortest route from the initial state to the goal state must be found, or a satisfaction problem, in which either any route from the initial state to the goal state must be found, or a proof that no such route exists must be provided.

N-queens puzzle: N chess queens must be placed on an $N \times N$ chessboard such that none of them is able to capture any other using the standard chess queen's moves [RN03]. The queens must be placed in such a way that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. N-queens puzzle is a satisfaction problem, any placement of queens satisfying the goal test is a valid solution.

Traveling salesman problem: given a set of cities, and known distances between each pair of cities, a tour that visits each city exactly once and that minimizes the total distance traveled must be found [RN03]. This is an optimization problem: a member of the set of all permutations of the cities with the shortest sum of distances between the consequent cities must be chosen.

Multi-armed bandit is the problem for a gambler to decide which arms of a K-slot machine to pull to maximize his total reward in a series of trials of a given length [VM05]. When pulled, each lever provides a reward drawn from a distribution associated to that specific lever. Initially, the gambler has no knowledge about the levers, but through repeated trials, can focus on the most rewarding levers. This is an optimization problem with trade-off between *exploration* and *exploitation*: the gambler both attempts to pull the most rewarding lever so far and tries different levers to discover a better lever.

2.2.2 Search Algorithms

Search problems are often solved by enumerating members of the set of feasible solutions until a member satisfying the goal test (for satisfaction problems), or maximizing the evaluation function (for optimization problems) is found. The two common enumeration strategies are *complete-state* and *partial-state* traversal. A search algorithm passes between states by performing *search actions* starting from some *initial state.* The algorithm stops when a state satisfying the goal test is reached.

Commonly, in the complete-state strategy a state is a complete feasible solution, and in the partial-state strategy a state is a partially built solution, when some of the structure of the final solution is left undefined. A slightly different view of the same classification is provided here to facilitate the discussion in the context of rational metareasoning.

In the complete-state strategy, a state corresponds to a single member of the set of feasible solutions: a placement of all N queens in the N-queens puzzle or a permutation of cities in the traveling salesman problem. State transitions are usually based on the structure of the set members: in the N-queens puzzle, states that differ from the current state in the position of a single queen can be viewed as neighbors of the state. The initial state can be chosen arbitrarily.

In the partial-state strategy, a state corresponds to a subset of the set of feasible solutions, and the initial state is the complete set. The search proceeds by considering states-subsets of the current state until a state consisting of a single element satisfying the goal test is found. The strategy is called 'partial-state' because each state is based on partial structure of the solution shared by all members of the subset corresponding to the set. For example, a state in a partial-state strategy for the traveling salesman problem can be the set of all permutations in which two particular cities are visited one immediately after the other.

Problem-independent, *uninformed* search algorithms, such as breadth-first search, iterative deepening search, and others, can be applied to any search problem without modifications. Unfortunately, these algorithms are incredibly inefficient in most cases: a complete-state algorithm may have to evaluate all of the set members; in a partial-state algorithm the number of possible states can be larger than exponential in the problem size.

Problem-specific knowledge can help find solutions more efficiently. *Heuristics* are used to encode the knowledge and to direct the search in such a way that the number of explored states is significantly decreased. The increase in performance depends on the quality of the heuristic. Good heuristics can sometimes be constructed by relaxing the problem definition, by precomputing solution costs for subproblems, or by learning from experience with the problem class.

The exact role of heuristics in a search algorithm varies for different algorithm families, of which best-first search, backtracking search, local search are most widely known and used. Besides, online search algorithms, in which computation and action are interleaved, are important for exploration problems and dynamic environments.

The following description of informed search algorithms is of necessity very skimpy, and does not and cannot cover all of the state-of-the art algorithms. Only some widely adopted schemes referenced in the thesis are briefly described.

Best-First Search

Best-first search (Algorithm 1) is used to solve route-finding or touring problems, in which a shortest path between the initial and the goal state must be found. The sliding tile puzzle and the traveling salesman problem are examples of such problems.

Best-first search repeatedly expands the best node in the fringe, according to a given evaluation function, and adds the node's children to the fringe (line 11). The algorithm terminates when either the goal is reached (line 7) or the fringe becomes empty (line 4). To avoid loops, visited nodes are kept in memory (line 10).

Algorithm 1 Best-First Search

```
1:  $ClosedNodes \leftarrow \emptyset$
2:  $Fringe \leftarrow \{InitialState\}$
3:  loop
4:      if $Empty(Fringe)$ then return failure
5:      end if
6:      $node \leftarrow RemoveBestNode(Fringe)$
7:      if $GoalTest(node)$ then return $node$
8:      end if
9:      if $node \notin ClosedNodes$ then
10:         $ClosedNodes \leftarrow ClosedNodes \cup \{node\}$
11:         $Fringe \leftarrow Fringe \cup \{Expand(node)\}$
12:     end if
13: end loop
```

The most widely used version of best-first search is A* search. A* evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the estimated cost to get from the node to the goal:

$$f(n) = g(n) + h(n) \tag{2.9}$$

Thus, $f(n)$ is the estimated cost of the cheapest solution through node n.

Provided that $h(n)$ is *admissible* and *consistent*, A* is optimal; even more, A* is *optimally efficient*: no other optimal algorithm is guaranteed to expand fewer nodes than A* [RN03]. Variations of A* with lower space requirements were developed. However, many search problems are NP-hard, which means that finding an optimal solution may take exponential time; approximating algorithms must be used to find a solution in polynomial time.

RTA* (Real-Time A*) [Kor90] is an approximating version of A* with limited lookup horizon. RTA* explores the search graph upto a given depth, and then moves to a node with the lowest estimated distance to the goal, updating $g(n)$ for all nodes to the distances from the current node. RTA* is not optimal unless the goal is within the lookup horizon, but the solution quality increases with the lookup depth. RTA* is complete in a space with positive edge hosts and finite heuristic values, in which the goal is reachable from any state. Variants of RTA* have been developed [RW91], [BLS+08] with improved control over the compromise between the computation time and the solution quality.

Backtracking Search

Backtracking search is used to solve *constraint satisfaction problems.* A constraint satisfaction problem (CSP) is defined by a set of variables, X_1, X_2, ..., X_n, and a set of constraints, C_1, C_2, ..., C_n. Each variable X_i has a non-empty domain D_i of possible values. Each constraint C_i involves some subset of the variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables; and an assignment that does not violate any constraints is called a consistent assignment. The eight queens puzzle can be formulated as a constraint satisfaction problem and solved using backtracking search.

Backtracking search is a depth-first search algorithm that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. A version of backtracking search is presented in Algorithm 2. The algorithm calls function *Backtracking(assignment, csp)* recursively, (line 8) trying values consistent with previous assignments (line 5) for each variable (line 4) until either a solution is found (line 8) or all combinations are exhausted (line 10). With each new assignment, the problem is updated according to constraints imposed by the assignment in order to prune the search space (line 7).

Algorithm 2 Backtracking Search

```
 1: procedure BACKTRACKING(assignment, csp)
 2:     if $Complete(assignment)$ then return $assignment$
 3:     end if
 4:     $var \leftarrow SelectUnassignedVariable(assignment, csp)$
 5:     for all $value$ in $OrderValues(var, assignment, csp)$ do
 6:         $assignment' \leftarrow assignment \cup \{var = value\}$
 7:         $csp' \leftarrow PropagateConstraints(csp, var = value)$
 8:         if $Backtracking(assignment', csp') \neq$ failure then return $assignment'$
 9:         end if
10:     end forreturn failure
11: end procedure
```

The algorithm performance depends heavily on the efficiency of variable-ordering and value-ordering heuristics (*SelectUnassignedVariable* in line 4 and *OrderValues* in line 5), as well on the amount of pruning due to the propagation of constraints (*PropagateConstraints* in line 7). Some variants of the backtracking search also involve *intelligent backtracking*, when additional values for recently assigned variables are not considered after a failure if it can be proved that that they would not fix the inconsistency.

Local Search

Local search is a family of complete-state algorithms, as opposite to best-first search and backtracking search. The search operates using a single *current state* and recursively explores neighbor states until a maximum of the evaluation function is found. Local search can be used to solve both satisfaction and optimization problems. To solve satisfaction problems, a heuristic function estimating proximity of a state to the goal is used instead of the evaluation function. The eight queens puzzle and the multi-armed bandit are examples of problems that can be solved using local search.

The simplest variant of local search, *hill-climbing* [RN03], moves only to higher-valued neighbors of the current state and is prone to getting stuck on local maxima. Therefore, variants that can escape a local maximum and eventually find a global maximum if enough time is given have been developed, such as *stochastic hill-climbing*, *random-restart hill climbing*, *simulated annealing* [RN03], and *genetic algorithms* [ES08]. These variants heuristically alternate between searching for a local maximum and exploring different areas of the search space.

Online Search

When the search space is dynamic or only partially known, an agent that builds a complete solution before committing to an action is often infeasible, because a contingency plan becomes prohibitively large. Consequently, online search algorithms come into use. In an online search algorithm, the agent interleaves computation and action, accounting for action outcomes in further computations.

Online versions of best-first search explore the search space up to a limited horizon or through sampling, use the current state of the search to choose an action, perform the action, and continue the search from the new state. Real-time A* is formulated as an offline search algorithm, but can be used unchanged by online search agents.

The simplest form of local search, hill climbing, can also be viewed as an online search algorithm. Other variants of local search involve ‘jumps’ in the search space which are free offline but come at a cost online. Consequently, random restarts or simulated annealing are impractical, and a *random walk* is used instead to escape local maxima. A random walk simply selects at random one of available actions, giving preference to actions that have not yet been tried.

Learning plays an important role in online search. The agent can keep information about visited states and outcomes of the performed actions, and use the gained experience to compute heuristics. An example of learning in online search is the LRTA* algorithm [Kor90], an extension of RTA*. LRTA* continuously updates cost estimates of visited states and used the current cost estimates to choose the “apparently best” move.

2.3 Related Work

Principles of rational metareasoning were formulated by Russell and Wefald [RW91], following Horvitz [Hor87]. In a case study of application of rational metareasoning to problem-solving search, Russell and Wefald [RW91] described the design of the DTA* search algorithm, a rational version of RTA* search by Korf [Kor90].

Horvitz and Klein [HK95] applied rational metareasoning to theorem proving. In particular, the authors showed how decision-theoretic methods can be used to determine the value of continuing to deliberate versus taking immediate action in time-critical situations. In a later work, Horvitz et al [HRG$^+$01] described methods of the decision-theoretic control of hard search and reasoning algorithms, illustrating the approach with a focus on the task of predicting run time for general and domain-specific solvers on a hard class of structured constraint satisfaction problems.

Zilberstein [Zil93] employed limited rationality techniques to analyze any-time algorithms. Radovilsky and Shimony [RS08] applied principles of rational metareasoning to the design of any-time algorithms for observation subset selection. Principles of rational metareasoning are used in some multi-armed bandit algorithms [VM05].

Gomes and Selman [GS01] analyzed dynamic algorithm portfolios to hard combinatorial search problems and proposed techniques for online algorithm selection and resource reallocation based on algorithm performance profiles. Domshlak, Karpas and Markovitch [DKM10] presented a method of reducing the cost of combining heuristics for optimal planning search by choosing the best heuristic to compute at each search state.

Domain-specific search algorithms and heuristics are discussed in Chapter 5: Case Studies. Work relevant to each of the problem domains is cited in the corresponding sections.

Chapter 3

Rational Metareasoning in Search Algorithms

An appropriate domain heuristic qualitatively improves performance of search algorithms, allowing to easily solve problems which were considered hard [AM96]. However, heuristics are inherently simplifications, and no heuristic is infallible even within a single problem class. When *heuristic portfolios* are available, some heuristics can be identified to work better than others on certain problem instances, and applied accordingly [AM96]; heuristics can also be combined [DKM10]. For certain problem instances, uninformed search may actually be the best option [KDG04].

Second-level heuristics are used to select, combine, or apply heuristics [AM96][BLS$^+$08]; however, these meta-heuristics are also domain-specific, and thus are difficult to generalize. Rational metareasoning offers a systematic approach for selection and application of heuristic computations in informed search. The techniques developed according to the approach are often applicable to many different problem domains and algorithm families.

When considering rational metareasoning in the context of a search algorithm, state transitions correspond to base level actions, and heuristic computations used to select among state transitions correspond to computational actions. Selection of a base-level action according to the rational metareasoning approach (Section 2.1) is presented in Algorithm 3. In the algorithm, the VOI of all computational actions available in the current state is repeatedly computed

Algorithm 3 Selecting base-level action

```
 1: loop
 2:     for all computational actions $S_i$ do
 3:         Compute $VOI(S_i)$
 4:     end for
 5:     $i_{\max} \leftarrow \arg\max_i VOI(S_i)$
 6:     if $VOI\,(S_{i_{\max}}) > 0$ then
 7:         Perform $S_i$
 8:     else
 9:         break
10:     end if
11: end loop
12: for all base-level actions $A_j$ do
13:     Compute $U(A_j)$
14: end for
15: $j_{max} \leftarrow \arg\max_j U(A_j)$
16: Select $A_{j_{max}}$
```

(lines 1–11), and if the maximum VOI is positive, a computational action $S_{i_{max}}$ with the maximum VOI is performed (line 7). Otherwise, a base-level action with the maximum utility is selected (line 16).

3.1 Search Algorithms with a Metareasoning Layer

The metareasoning layer of an informed search algorithm decides

- whether to apply an heuristic computation or commit to an action;
- which of the heuristics or what combination thereof to use;
- if the heuristic computation is parametrized, what parameter values result in the best performance.

The metareasoning layer estimates the value of information and the cost of heuristic computations, and makes decisions according to the principles of metareasoning (Section 2.1). This way, the layer of domain-specific heuristics is placed between the domain-independent layers of the search algorithm and the metareasoning.

In real-time or online **best-first search** an important parameter is the lookup horizon. Both the intrinsic value of information and the computation cost usually grow with the lookup horizon, and there is often a non-trivial optimal value maximizing the net value of information.

Another important parameter is the subset of nodes to expand and the order of expansion. The computations are node expansions with application of heuristic estimates to the leaves; the base-level actions are state transitions.

In **backtracking search** for constraint satisfaction problems the base-level actions are variable assignments, and the computations are heuristics. There are many different heuristics [Tsa93]:

- for variable ordering (most constrained variable, minimal width, minimal bandwidth, maximum cardinality etc.);
- for value ordering (minimum conflicts, least impact etc.);
- for constraint propagation (forward checking, lookahead, maintaining arc consistency etc.).

Some of the heuristics can be quite expensive, and should be used only if the anticipated benefit is greater than the computation cost. In addition, new heuristics can be invented based on the principles of metareasoning (Section 5.2).

Local search is itself an heuristic. The local search heuristic suggests that the agent should search for an improvement of the evaluation function in the proximity of the best state found so far; when no improvement can be achieved, the agent should turn to unexplored parts of the search space. The metareasoning approach offers a generalization of the heuristic based on the value of information of evaluating a state: the next state to evaluate is the state with the greatest expected influence on the final choice, given the current beliefs. The state can be either close to explored states with high values, or in an unexplored region with high uncertainty about values (exploration). According to this view, there is a single base level action—the final choice, and the computational action is evaluation of a state.

Still, in many cases it is not clear what reasoning model of a search algorithm suits the metareasoning approach the best. Classification and analysis of search algorithms along the lines of the metareasoning approach would advance both theoretical research and applications of problem-solving search.

3.2 Assigning Utility and Computation Cost

According to the rational metareasoning approach, the agent must be able to estimate utilities of base-level actions and costs of computational actions (Section 2.1.3). In some cases, the estimates are obvious, in others, the task of finding appropriate estimates is complicated.

For estimating **utility**, the simplest case is solving an optimization problem using a complete-state algorithm. The utility of the only base-level action—the selection of an element from the set—is the value of the evaluation function on the selected element. If a complete-state algorithm is used for solving a satisfaction problem, e.g. the *min-conflicts* algorithm [RN03], then the evaluation function that estimates 'closeness' of the state to a consistent state serves as the utility function.

In partial-state algorithms, such as best-first or backtracking search, both the solution quality and the pending base-level and computational actions contribute to the utility of a base-level action. Solution quality can often be assumed independent of the search cost. Under this assumption, the utility of an action is the solution quality less the amortized cost of the search:

$$U_{action} = Q_{solution} - \alpha C_{search} \tag{3.1}$$

where α is the amortization factor signifying the importance of obtaining a solution fast: if the solution is used only once, such as in the case of an online path search, $\alpha = 1$; for reusable solutions, α goes down. In satisfaction problems, when any solution of the problem has the same quality, $Q_{solution}$ can be omitted from (3.1), along with the amortization factor α, and the action utility is simply the search cost taken with the negative sign:

$$U_{action} = -C_{search} \tag{3.2}$$

C_{search} is composed of the cost of both computational and base-level actions. In some algorithms, such as backtracking search, the total search effort can be estimated [Knu74][Ref04]; in others, such as best-first search, the cost of pending computations is ignored, and the estimated cost of base-level actions only is used [RW91].

The **cost** of a computational action can be estimated directly in some cases [RW91]; in other cases, the cost is learned during the same or earlier invocations of the search algorithm, and can depend on the size of the problem instance. In some algorithms (see Section 5.2 for an example), just the ratio between costs of the base-level actions and the computational actions, and not their absolute values, determines the algorithm behavior.

3.3 Representing and Revising Beliefs

Value of information of a computational action depends on the belief distribution of outcomes of the action (Section 2.1.2). Often, the belief distribution is represented as an error distribution around the true value of the quantity to compute (e.g. the path length in best-first search, the value of a state in local search); of course, the true value itself is also unknown. The error distribution can be chosen heuristically, learned from earlier invocations of the algorithm on other problem instances from the same domain, or gradually refined during the search using Bayesian inference, starting with a prior belief.

There are thus, in the general case, several simultaneous processes of Bayesian inference during the search:

- beliefs about quantities computed by the computational actions are updated based on outcomes of the actions;
- beliefs about other quantities dependent on the computed quantities are revised according to the dependencies;
- beliefs about error distributions of outcomes of computational actions are revised.

Proper handling of base-level and meta-level beliefs in search algorithms is crucial for algorithm performance and allows to replace the guessing and *ad hoc* heuristics with uniform techniques applicable to a wide range of problems. Chapter 4 presents a problem of maintaining beliefs about error distributions of computational actions and discusses solution approaches and initial results.

Chapter 4

Rational Computation of Value of Information

4.1 Introduction

Problems of decision-making under uncertainty frequently contain cases where information can be obtained using some costly actions [WP09], called measurement actions. In order to act rationally in the decision-theoretic sense, measurement plans are typically optimized based on some form of value of information (VOI). Computing value of information can also be computationally intensive. Since an exact VOI is often not needed in order to proceed (e.g. it is sufficient to determine that the VOI of a certain measurement is much lower than that of another measurement, at a certain point in time), significant computational resources can be saved by controlling the resources used for estimating the VOI. This tradeoff is examined via a case study of measurement selection.

In general, computing the VOI, even under the commonly used simplifying myopic assumption, involves multidimensional integration of a general function [RW91]. For some problems, the integral can be computed efficiently [RW89]; but when the utility function is computationally intensive or when a non-myopic estimate is used, the time required to compute the VOI can be significant [HHM93] [BG07] and the computation time cost must be taken into account. This study presents and analyzes an extension of the known greedy algorithm. The extension decides when to recompute the VOI of each of the measurements based on the principles of limited rationality (Section 2.1).

Although it may be possible to use this idea in more general settings, here the content is on-line most informative measurement selection [KG07] [BG07], an approach which is commonly used to solve problems of optimization under uncertainty [ZRB05] [KLG+08]. Since this approach assumes that the computation time required to select the most informative measurement is negligible compared to the measurement time, it is important in this setting to ascertain that VOI estimation indeed does not consume excessive computational resources.

4.2 The Measurement Selection Problem

Let us examine the following optimization problem: given a set of items of unknown utility (but a distribution of which is known), an item with as high a utility as possible must be selected. Measurements (possibly noisy) of item features are allowed prior to a final selection of an item, at known costs. The objective is to optimize the overall decision process of measurement and selection. Formally, the measurement selection problem is a 6-tuple (S, Z, P_0, M, u, C) where:

- $S = \{s_1, s_2, \ldots s_{N_s}\}$ is a set of N_s items.
- $Z = \{z_1, z_2, \ldots, z_{N_f}\}$ is a set of N_f item features; each feature z_i has a domain $\mathcal{D}(z_i) \subseteq \mathbb{R}$.
- P_0 is a joint distribution over the features of the items in S. That is, a joint distribution over the random variables $\{z_1(s_1), z_2(s_1), \ldots, z_1(s_2), z_2(s_2), \ldots\}$.
- $M = \{m_k = (c, p)_k \,|\, k \in 1..N_m\}$, is a set of measurement types, with potentially different intrinsic measurement cost $c \in \mathbb{R}$ and observation probability distribution p of the observed feature values, conditional on the true feature values, for each measurement type. Repeated measurements are assumed independent given feature values.
- $u(\mathbf{z})\colon \mathbb{R}^{N_f} \to \mathbb{R}$ is a known utility function of an item over feature values.
- C is a measurement budget.

A policy of measurement and selection for a selection problem is a mapping π (either explicit or implicit) from belief states (distributions over item features, or alternately histories of observations) to actions, which are either measurements or selection of a final item. A policy applied to the initial distribution P_0 results in a (stochastically generated) sequence of measurements and final selection

$$Q = \{q_i = (k_i, s_i) \,|\, i \in 1..N_q\}.s_\alpha \tag{4.1}$$

where k_i is the type of measurement q_i, s_i is the item measured by q_i, and s_α is the final selected item. The goal is to find such a policy that obeys the budget constraint and maximizes the expectation of the reward R over all possible measurement outcomes, with a distribution based on the initial belief P_0 and the information received from measurements according to the observation distribution model. The objective function is:

$$\max \mathbb{E}[R] \colon R \triangleq u(\mathbf{z}(s_\alpha)) - \sum_{i=1}^{N_q} c_{k_i} \quad \text{s.t.:} \sum_{i=1}^{N_q} c_{k_i} \leq C \tag{4.2}$$

Problems from different application areas can be viewed as instances of the measurement selection problem. Equation 4.2 assumes that costs and utilities are commensurable. Often, this is indeed the case, e.g. when both the utility and the cost are computation times. Otherwise, a mapping of the quantities to the same units must be provided, as illustrated by the following examples:

> **Water reservoir monitoring:** A water reservoir is monitored for contamination sources. Water probes can be taken in a number of predefined spots, and the goal is to predict the location of a contamination source based on analysis of water quality. The contamination must be identified quickly, before it distributes too far or affects the consumers. In this problem, the *features* are concentrations of possible contaminants, the *utility function* is the time to find the contamination source given the predicted location, and the *measurement cost* is the time required to perform a probe.
>
> **SVM parameter optimization:** Classification accuracy of a support vector machine (SVM) depends on one or more parameters (see Section 4.6.2 for a case study). While there are heuristics for selecting good parameter values for particular kernel types and data sets, several combinations of parameters must be tried before a good one can be chosen. A good setting must be found under certain time constraints. Here, the only *feature* is the classification accuracy α, the *utility function*

is the identity function $u(\alpha) = \alpha$, and the *measurement cost* is proportional to the computation time with a factor relfecting the time constraints.

The above selection problem is intractable, and is therefore commonly solved approximately using a greedy heuristic algorithm. The greedy algorithm selects a measurement $q_{j_{\max}}$ with the greatest net VOI $V_{j_{\max}}$. The *net VOI* (or just 'VOI') is the difference between the intrinsic VOI and the measurement cost.

$$V_j = \Lambda_j - c_{k_j} \tag{4.3}$$

The *intrinsic VOI* Λ_j is the expected difference in the true utility of the finally selected item s_α after and before the measurement:

$$\Lambda_j = \mathbb{E}_{q_j}(\mathbb{E}_{z|q_j}[u(\mathbf{z}(s_{\alpha^j}))] - \mathbb{E}_{z|q_j}[u(\mathbf{z}(s_\alpha))]) \tag{4.4}$$

where expectation $\mathbb{E}_{q_j}$ is computed according to the belief distribution about outcomes of the jth measurement, and $\mathbb{E}_{z|q_j}$ — according to the belief distribution of the features given an outcome of the measurement. Exact computation of Λ_j is intractable, and various estimates are used, including the myopic estimate [RW91] and semi-myopic schemes [TS12c].

The pseudocode for the greedy algorithm is presented as Algorithm 4. The algorithm maintains a persistent data structure which holds beliefs about feature values of the items. The beliefs are initialized to the prior beliefs (line 2), and then updated according to measurement outcomes (line 17). The main loop (lines 3–21) continues as long as there are measurements with the positive VOI (line 15) that fit within the budget (line 8). Otherwise, the loop terminates (line 20), and the algorithm returns an item with the maximum expected utility (line 22). Variable *budget* is initialized to the total budget C, and decreased by the cost of each performed measurement. Thus, the algorithm is guaranteed to terminate if the costs of all measurements are strictly positive and bounded away from zero.

At each step, the algorithm recomputes the VOI of every measurement (line 9). For the myopic scheme, the computation involves evaluation of a multi-dimensional integral of a general function (Equation 4.4), repeated for each measurement. For semi-myopic VOI computation [TS12c], lines 7–13 of Algorithm 4 are replaced by Algorithm 5, and the multi-dimensional integral must be evaluated for each measurement batch (line 4 of Algorithm 5). Depending on the particular semi-myopic scheme, there can be many more batches than measurements;

Algorithm 4 Greedy measurement selection

```
$budget \leftarrow C$
Initialize beliefs
loop
    for all items $s_i$ do
        Compute $\mathbb{E}(U_i)$
    end for
    for all measurements $q_j$ do
        if $c_j \le budget$ then
            Compute $V_j$
        else
            $V_j \leftarrow 0$
        end if
    end for
    $j_{\max} \leftarrow \arg\max_j V_j$
    if $V_{j_{\max}} > 0$ then
        Perform measurement $q_{j_{\max}}$
        Update beliefs
        $budget \leftarrow budget - c_{j_{\max}}$
    end if
    else break
end loop
$\alpha \leftarrow \arg\max_j \mathbb{E}(U_i)$ return $s_\alpha$
```

for example, for the blinkered scheme [TS12c] with measurement cost c the number of batches is $O\left(K \log\left(\frac{C}{c}\right)\right)$. The assumptions behind the greedy algorithm are justified when the cost of selecting the next measurement is negligible compared to the measurement cost. However, optimization problems with hundreds and thousands of items are common [TS12c]; and even if the VOI of a single measurement can be computed efficiently [RW89], the cost of estimating the VOI of all measurements may become comparable, or even outgrow the cost of performing a measurement.

Recomputing the VOI for every measurement is often unnecessary. When there are many different measurements, the VOI of most measurements is unlikely to change abruptly due to an outcome of just one other measurement. With an appropriate uncertainty model, it can be shown that the VOI of only a few of the measurements must be recomputed after each measurement, thus decreasing the computation time and ensuring that the greedy algorithm exhibits a more rational behavior w.r.t. computational resources. The focus here is to explore an improvement in the algorithm due to selective VOI recomputation.

Algorithm 5 Semi-myopic VOI computation

```
forall measurements $q_j$ do $V_j \leftarrow 0$
for all batches $b_k$ satisfying constraint $\mathcal{C}$ do
    if $cost(b_k) \leq budget$ then
        compute $V_k^b$
        for all measurements $q_j \in b_k$ do
            if $V_j < V_k^b$ then $V_j \leftarrow V_k^b$
        end for
    end if
end for
```

4.3 Rational Computation of the Value of Information

For the selective VOI recomputation, the VOI of each measurement is modeled as *known with uncertainty.* The belief BEL(V_j) about the VOI of measurement q_j is represented by a belief distribution. In particular, the normal distribution with mean Λ_j and variance ς_j^2 is used, although other distributions models could be used as necessary.

$$\mathrm{BEL}(V_j) = \mathcal{N}(V_j, \varsigma_j^2) \tag{4.5}$$

After a measurement is performed, and the beliefs about the item features are updated (line 17 of Algorithm 4), the belief about V_j becomes less certain. Under the assumption that the influence of each measurement on the VOI of other measurements is independent of influence of any other measurement, the uncertainty is expressed by adding noise to the belief distribution. For the normal belief distribution, the noise is also modeled by the normal distribution with zero mean and variance τ^2. Since the sum of two independent normally distributed random variables $X = \mathcal{N}(\mu_x, \sigma_x^2)$ and $Y = \mathcal{N}(\mu_y, \sigma_y^2)$ is a normally distributed random variable $Z = \mathcal{N}(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2)$, the variance of the noise distribution τ^2 is added to the variance of the belief distribution ς_j^2:

$$\varsigma_j^2 \leftarrow \varsigma_j^2 + \tau^2 \tag{4.6}$$

When V_j of measurement q_j is computed, BEL(V_j) becomes exact ($\varsigma_j^2 \leftarrow 0$). At the beginning of the algorithm, the beliefs about the VOI of measurements are computed from the initial beliefs about item features.

In the algorithm that recomputes the VOI selectively, the initial beliefs about the VOI are computed immediately after line 2 in Algorithm 4, and lines 7–14 of Algorithm 4 are replaced

Algorithm 6 Rational computation of the VOI

```
 1: for all measurements $q_j$ do
 2:     if $c_j \le budget$ then
 3:         $V_j \leftarrow \Lambda_j - c_j$
 4:         $\varsigma_j \leftarrow \sqrt{{\varsigma_j}^2 + \tau^2}$
 5:     else
 6:         $V_j \leftarrow 0$
 7:         $\varsigma_j \leftarrow 0$
 8:     end if
 9: end for
10: loop
11:     for all measurements $q_k$ do
12:         if $c_k \le budget$ then
13:             Compute $W_k$                                  ▷ Equation (4.7)
14:         else
15:             $W_k \leftarrow 0$
16:         end if
17:     end for
18:     $k_{\max} \leftarrow \arg\max_k W_k$
19:     if $W_{k_{\max}} \le 0$ then break
20:     Compute $V_{k_{\max}}$
21:     $\varsigma_{k_{\max}} \leftarrow 0$
22: end loop
23: $j_{\max} \leftarrow \arg\max_j V_j$
24: Compute $V_{j_{\max}}$
25: $\varsigma_{j_{\max}} \leftarrow 0$
```

by Algorithm 6. While the number of iterations in lines 11–18 of Algorithm 6 is the same as in lines 7–13 of Algorithm 4, W_k is efficiently computable, and the subset of measurements for which the VOI is computed in line 20 of Algorithm 6 is controlled by the computation cost c_V:

$$\begin{aligned} W_k &= -c_V + \begin{cases} \frac{1}{\sqrt{2\pi}\varsigma_k} \int\limits_{-\infty}^{V_\beta} (V_\beta - x) e^{\left(-\frac{(x-V_k)^2}{2\varsigma_k^2}\right)} dx & \text{if } V_k = V_\alpha \\ \frac{1}{\sqrt{2\pi}\varsigma_k} \int\limits_{V_\alpha}^{\infty} (x - V_\alpha) e^{\left(-\frac{(x-V_k)^2}{2\varsigma_k^2}\right)} dx & \text{if } V_k \le V_\beta \end{cases} \\ &= -c_V + \frac{\varsigma_k}{\sqrt{2\pi}} e^{\left(-\frac{(V_k-V_\gamma)^2}{2\varsigma_k^2}\right)} - |V_k - V_\gamma| \, \Phi\left(-\frac{|V_k - V_\gamma|}{\varsigma_k}\right) \end{aligned} \tag{4.7}$$

where

- $\Phi(x)$ is the normal cumulative probability function,

- V_α is the highest, and V_β is the next to highest net VOI estimate,

- $V_\gamma = \begin{cases} V_\beta & \text{if } V_k = V_\alpha \\ V_\alpha & \text{if } V_k \leq V_\beta \end{cases}$.

4.4 Obtaining Uncertainty Parameters

Uncertainty variance τ^2 depends on the current beliefs about item features. Beliefs are changed with each measurement, and the dependency of the uncertainty variance on the beliefs is complicated. The total cost of the performed measurements may serve as a scalar measure of influence of observations on the beliefs. The variance τ^2 is established as a function of the total cost of the measurements performed since the beginning of the run and of the cost of the last measurement:

$$\tau_k^2 = f\left(\sum_{j=1}^{k-1} c_{i_j}, c_{i_k}\right) \tag{4.8}$$

When the cost of any single measurement is significantly smaller than the total budget, $c_{\max} \ll C$, τ_k^2 can be approximated as the product of c_k and of a function independent of c_k:

$$\begin{aligned} \tau_k^2 &= c_{i_k} g\left(\sum_{j=1}^{k-1} c_{i_j}\right) \\ g(c) &= \frac{\delta f(c, \eta)}{\delta c}, \quad 0 \leq \eta \leq c_{\max} \end{aligned} \tag{4.9}$$

Dependency $g(c)$ can be obtained in one of the following ways:

- assumed fixed, $g(c) = G$, with constant G determined from earlier runs or derived from an heuristic;
- learned off-line from earlier runs on other problem instances;
- learned on-line from the effects of earlier measurements on the change in the VOI in the same problem instance.

The first two options depend on availability of training data for the problem class, and require offline learning of the dependency prior to application of the rational recomputing algorithm. However, learning $g(c)$ online from earlier VOI recomputations during the same run proved to be robust and easy to implement by gradually updating τ^2 after each VOI recomputation.

4.5 Computation Time

Influence of the rational VOI recomputation on the computation time can be estimated under the assumption that there are potentially many measurements—the cost of a single measurement c is negligible compared to the budget C: $c \ll C$.

When the computation cost c_V increases, the VOI for a smaller number of measurements is recomputed at each step, and the computation time of the rational recomputing algorithm decreases. Let η be the ratio between the expected computation time of the rational recomputing algorithm T_r to the computation time of the original algorithm T: $\eta = \mathbb{E}\left(\frac{T_r}{T}\right)$.

According to the assumption, τ changes slowly. Given τ, ς_j^2 of measurement q_i in the measurement sequence Q (Equation 4.1) is proportional to the *VOI age*—the total cost of measurements performed since the last recomputation of the VOI of the measurement. Assuming that the time to compute the VOI of a single measurement is constant, the expected VOI age over all measurements and all steps of the algorithm is $\frac{1}{\eta}$: $\mathbb{E}_{i,j}(age_j^i) = \frac{1}{\eta}$.

VOI variance ς^2 (Equation 4.6) is proportional to the VOI age given τ, i.e. inversely-proportional to η:

$$\varsigma^2 = \Theta\left(\frac{1}{\eta}\right) \tag{4.10}$$

As follows from substitution of (4.10) into (4.7), the expected computation time of the rational recomputing algorithm decreases with the logarithm of the computation cost:

$$\eta = \Theta\left(1 - \alpha \log c_V\right) \tag{4.11}$$

where α is a constant. Empirical evaluations (Section 4.6) confirm this estimate.

4.6 Empirical Evaluation

Experiments in this section compare performance of the algorithm that recomputes the VOI selectively with the original algorithm in which the VOI of every measurement is recomputed at every step. Two of the problems evaluated in [TS12c] are considered: *noisy Ackley function maximization* and *SVM parameter search*. For these problems, the performance of the greedy algorithm was analyzed in [TS12c], and the semimyopic (blinkered) scheme, while giving better results than the myopic scheme, caused an increase in the computation time that, while still polynomial in the size of the problem, can be prohibitive. According to blinkered scheme, the value of information of a measurement is estimated by considering batches of multiple measurements of a single item.

For each of the optimization problems, four plots are presented:

- the number of VOI recomputations,
- the reward (Equation 4.2),
- the intrinsic utility,
- and the total cost of measurements.

The cost of VOI recomputation is not subtracted from the final reward, since the chosen greedy algorithm terminates the recomputation based on the difference between the intrinsic VOI and the cost of a single VOI computation, rather than of all VOI computations made since the last measurement; consequently, the time costs of VOI recomputations and of measurements are on different scales. An improved termination condition would allow a more accurate analysis of the reward; however, the algorithm behavior would stay the same, with the appropriate scaling of the computation cost.

The results are averaged for multiple (100) runs of each experiment, such that the standard deviation of the reward is $\approx 5\%$ of the mean reward. In the plots,

- the solid line corresponds to the rational recomputing algorithm,
- the dashed line corresponds to the original algorithm,
- the dotted line corresponds to the Monte Carlo algorithm that selects measurements randomly and performs the same number of measurements as the rational recomputing algorithm for the given computation cost c_V.

Since, as can be derived from (4.7), the computation time T_r of the rational recomputing algorithm decreases with the logarithm of the computation cost c_V (Equation 4.11), the computation cost axis is scaled logarithmically.

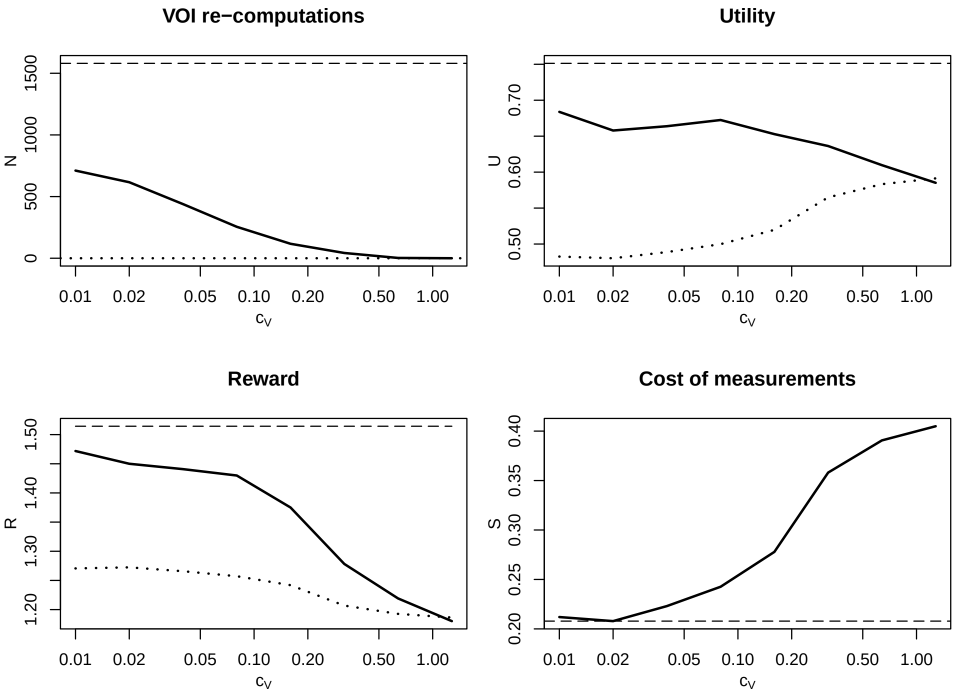

Figure 4.1: The Ackley function, myopic scheme.

4.6.1 The Ackley Function

The Ackley function [Ack87] is a popular optimization benchmark. The two-argument form of the Ackley function is used in the experiment; the function is defined by the expression (4.12):

$$A(x,y) = 20 \cdot \exp\left(-0.2\sqrt{\frac{x^2+y^2}{2}}\right) + \exp\left(\frac{cos(2\pi x)+cos(2\pi y)}{2}\right) \tag{4.12}$$

In the optimization problem, the utility function is $u(z) = \tanh(2z)$, the measurements are normally distributed around the true values with variance $\sigma_m^2 = 0.5$, and the measurement cost is 0.01. There are uniform dependencies with $\sigma_w^2 = 0.5$ in both directions of the coordinate grid with a step of 0.2 along each axis. The results are presented in Figure 4.1 for the myopic scheme, and in Figure 4.2 for the blinkered scheme [TS12c].

4.6.2 SVM Parameter Search

An SVM (Support Vector Machine) classifier based on the radial basis function has two parameters: C and γ. A combination of C and γ with high expected classification accuracy should be chosen, and an efficient algorithm for determining the optimal values is not known. A trial

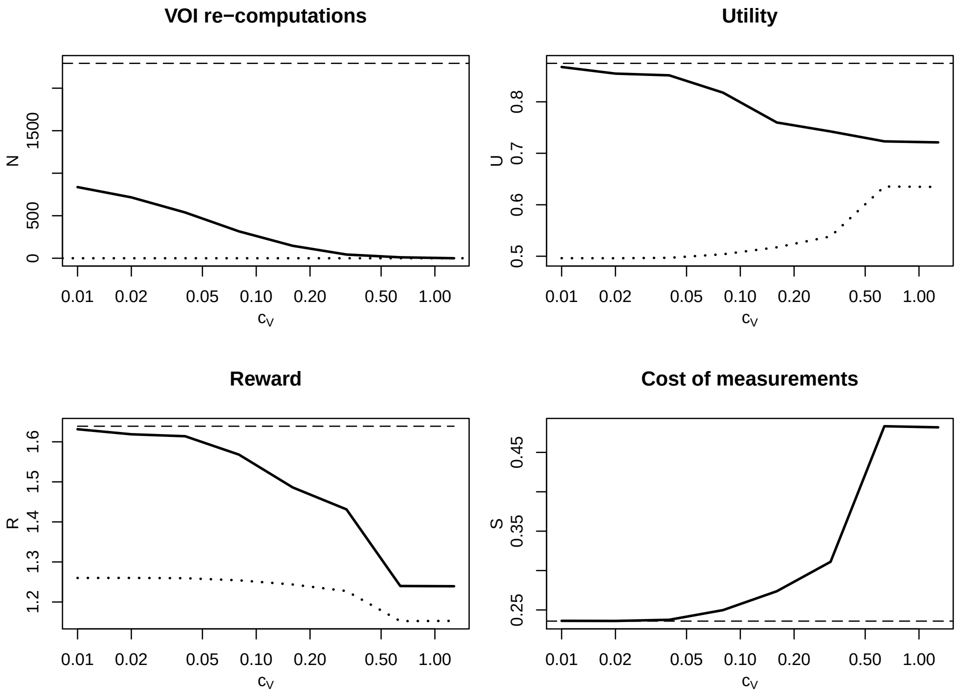

Figure 4.2: The Ackley function, blinkered scheme.

for a combination of parameters determines estimated accuracy of the classifier through cross-validation. The SVMGUIDE2 [HCL10] dataset is used for the case study. The utility function is $u(z) = \tanh(4(z-0.5))$, the $\log C$ and $\log \gamma$ axes are scaled for uniformity to ranges [1..21] and there are uniform dependencies along both axes with $\sigma_w^2 = 0.4$. The measurements are normally distributed with variance $\sigma_m^2 = 0.25$ around the true values, and the measurement cost is $c_m = 0.01$. The results are presented in Figure 4.3 for the myopic scheme, and in Figure 4.4 for the blinkered scheme [TS12c].

4.6.3 Discussion of Results

In all experiments, a significant decrease in computation time is achieved with only a slight degradation of the reward; performance of the rational recomputing algorithm decreases slowly with the computation cost and exceeds performance of the algorithm that makes random measurements even when VOI estimates for only a small fraction of measurements is recomputed at each step. Exact dependency of performance of the rational recomputing algorithm on the overhead caused by the VOI recomputations varies among problems and depends both on the problem properties and on the VOI estimate used in the algorithm.

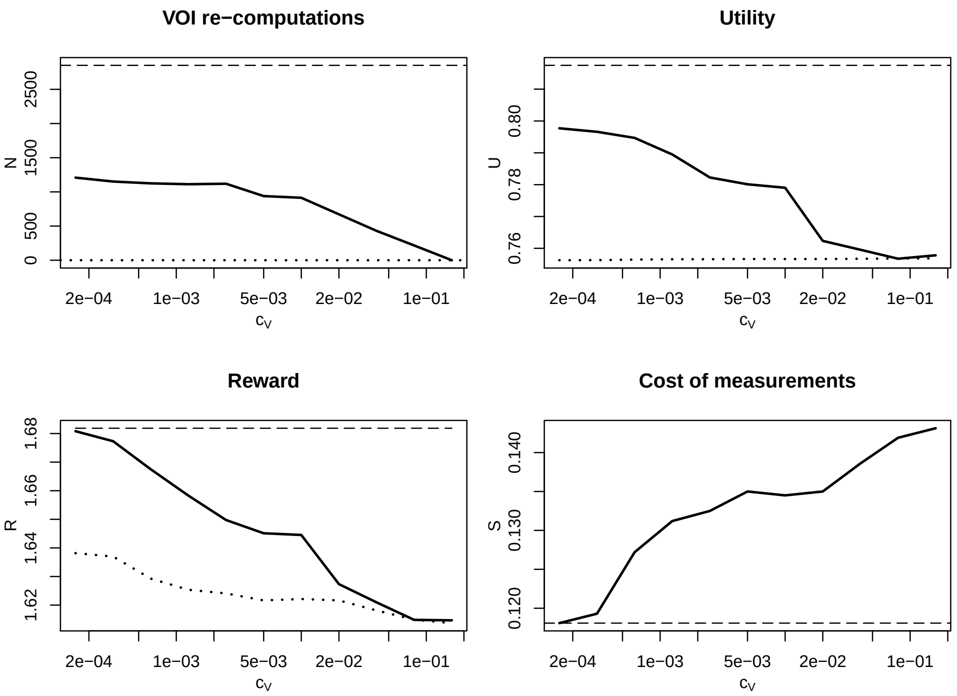

Figure 4.3: SVM parameter search, myopic scheme.

4.7 Conclusion and Further Research

This chapter proposes an improvement to a widely used class of VOI-based optimization algorithms. The improvement allows to decrease the computation time while only slightly affecting the reward. The proposed algorithm rationally reuses computations of VOI estimates and recomputes the estimates only for measurements for which a change in the VOI is likely to affect the choice of the next measurement.

The proposed scheme of rational VOI computation can be further improved.

- the normal distribution is chosen rather arbitrarily to model uncertainty about the VOI. Often, the VOI is non-increasing [KG07], and a skewed distribution from the exponential family should be chosen for better results.
- The termination condition of VOI recomputation should be improved such that the VOI recomputation cost could be accounted for in the final reward.
- The scheme is applied to the greedy algorithm for measurement selection, but can be extended to other VOI-based algorithms.

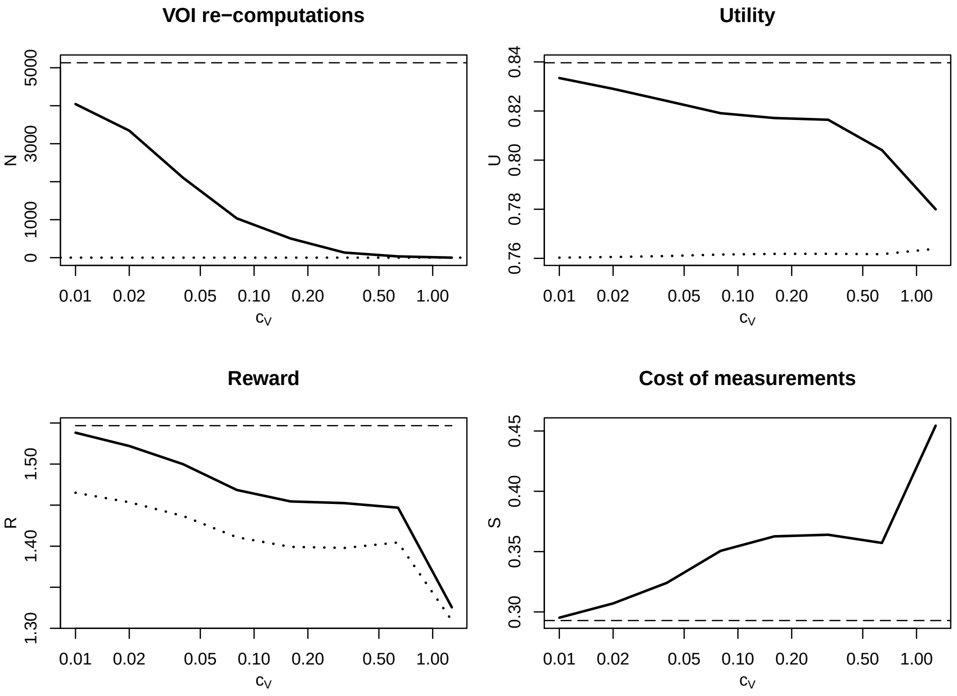

Figure 4.4: SVM parameter search, blinkered scheme.

Chapter 5

Case Studies

5.1 Introduction

In this chapter three case studies of applications of rational metareasoning are presented. The studies explore rational metareasoning in the context of satisfaction problems (Section 5.2), as well as of approximating (Section 5.3) and optimal (Section 5.3) search in optimization problems. In addition to providing a metareasoning-based solution to a particular problem, the studies outline both techniques which are common to different problem types (identifying base-level and computational actions, defining the utility and the value of information, etc.) and those which exploit features present only in some of the problems, such as estimating the VOI based on distribution-independent bounds (Section 5.3).

A straightforward application of rational metareasoning often faces difficulties because some of the assumptions of the approach do not hold. For example, estimating the value of information depends on the belief distribution of action outcomes; however, a reasonable choice of the distribution is not always possible. Section 5.2 derives the distribution model from the properties of the search algorithm rather than from the set of problem instances. Another example is the myopic assumption [RW91] which suggests that an action can be chosen based on its anticipated immediate effect. In some search algorithms, the information is obtained through *sampling*—recurring actions with probabilistic outcomes—and the myopic VOI of a single sample is often zero. Section 5.3 addresses such a case in the context of Monte-Carlo tree search and suggests a semi-myopic VOI estimate.

Advanced search algorithms are often parameterized and require tuning to achieve the best performance. Building an algorithm upon the principles of rational metareasoning allows to keep the number of tunable parameters to a minimum—a single parameter needs to be tuned in Sections 5.2, 5.4. In addition, as Section 5.2 demonstrates, parameters of a metareasoning-based algorithm are likely to reflect implementation details of the algorithm and the heuristics rather than features of a particular set of problem instances.

Finally, rational metareasoning is commonly applied to approximating search where decision quality is traded for computation time. Optimal search algorithms aim at finding the optimal solution in the shortest possible time, and seemingly little can be achieved through application of rational metareasoning. Section 5.4 presents metareasoning-based improvements to a variant of the A^* algorithm which result in finding the optimal solution in a shorter time using the same heuristic functions; the employed technique is general enough to extend to other optimal search algorithms.

5.2 Rational Deployment of CSP Heuristics

5.2.1 Introduction

Large search spaces are common in artificial intelligence, heuristics being of major importance in limiting search efforts. The role of a heuristic, depending on type of search algorithm, is to decrease the number of nodes expanded (e.g. in A* search), the number of candidate actions considered (planning), or the number of backtracks in constraint satisfaction problem (CSP) solvers. Nevertheless, some sophisticated heuristics have considerable computational overhead, significantly decreasing their overall effect [HH00, KDG04], even causing *increased* total runtime in pathological cases. It has been recognized that control of this overhead can be essential to improve search performance; e.g. by selecting which heuristics to evaluate in a manner dependent on the state of the search [WF92, DKM10].

We propose a rational metareasoning approach to decide when and how to deploy heuristics, using CSP backtracking search as a case study. The heuristics examined are various *solution count estimate* heuristics for value ordering [MSS97, HH00], which are expensive to compute, but can significantly decrease the number of backtracks. These heuristics make a good case study, as their overall utility, taking computational overhead into account, is sometimes detrimental; and yet, by employing these heuristics adaptively, it may still be possible to achieve an overall runtime improvement, even in these pathological cases. Following the metareasoning approach, the value of information (VOI) of a heuristic is defined in terms of total search time saved, and the heuristic is computed such that the expected net VOI is positive.

We begin with background and related work on CSP (Section 5.2.2), followed by a restatement of value ordering in terms of rational metareasoning (Section 2.1), allowing a definition of VOI of a value-ordering heuristic. This scheme is then instantiated to handle our case-study of backtracking search in CSP (Section 5.2.4), with parameters specific to value-ordering heuristics based on solution-count estimates. Empirical results (Section 5.2.5) show that the proposed mechanism successfully balances the tradeoff between decreasing backtracking and heuristic computational overhead, resulting in a significant overall search time reduction. Other aspects of such tradeoffs are also analyzed empirically. Finally, possible future extensions of the proposed mechanism are discussed (Section 5.2.6).

5.2.2 Background and Related Work

A constraint satisfaction problem (CSP) is defined by a set of variables $\mathcal{X} = \{X_1, X_2, ...\}$, and a set of constraints $\mathcal{C} = \{C_1, C_2, ...\}$. Each variable X_i has a non-empty domain D_i of possible values. Each constraint C_i involves some subset of the variables—the scope of the constraint—and specifies the allowable combinations of values for that subset. An assignment that does not violate any constraints is called *consistent* (or *a solution*). There are numerous variants of CSP settings and algorithmic paradigms. This study focuses on binary CSPs over discrete-values variables, and backtracking search algorithms [Tsa93].

A basic method used in numerous CSP search algorithms is that of maintaining arc consistency (MAC) [SF97]. There are several versions of MAC; all share the common notion of *arc consistency.* A variable X_i is arc-consistent with X_j if for every value a of X_i from the domain D_i there is a value b of X_j from the domain D_j satisfying the constraint between X_i and X_j. MAC maintains arc consistency for all pairs of variables, and speeds up backtracking search by pruning many inconsistent branches.

CSP backtracking search algorithms typically employ both variable-ordering [Tsa93] and value-ordering heuristics. The latter type include *minimum conflicts* [Tsa93], which orders values by the number of conflicts they cause with unassigned variables, *Geelen's promise* [Gee92] — by the product of domain sizes, and *minimum impact* [Ref04], which orders values by relative impact of the value assignment on the product of the domain sizes.

Some value-ordering heuristics are based on solution count estimates [DP87, MSS97, HH00, KDG04]: solution counts for each value assignment of the current variable are estimated, and assignments (branches) with the greatest solution count are searched first. The heuristics are based on the assumption that the estimates are correlated with the true number of solutions, and thus a greater solution count estimate means a higher probability that a solution be found in a branch, as well as a shorter search time to find the first solution if one exists in that branch. [MSS97] estimate solution counts by approximating marginal probabilities in a Bayesian network derived from the constraint graph; [HH00] propose the *probabilistic arc consistency* heuristic (pAC) based on iterative belief propagation for a better accuracy of relative solution count estimates; [KDG04] adapt Iterative Join-Graph Propagation to solution counting, allowing a tradeoff between accuracy and complexity.

These methods vary by computation time and precision, although all are rather computationally heavy. As a means to improve the performance of search algorithms based on computationally expensive heuristics, runtime selection of heuristics has lately become of interest, e.g. deploying heuristics for planning [DKM10]. The approach taken is usually that of *learning* which heuristics to deploy based on features of the search state.

5.2.3 Rational Value-Ordering

The role of (dynamic) value ordering is to determine the order of values to assign to a variable X_k from its domain D_k, at a search state where values have already been assigned to $(X_1, ..., X_{k-1})$. We make the standard assumption that the ordering may depend on the search state, but is not re-computed as a result of backtracking from the initial value assignments to X_k: a new ordering is considered only after backtracking up the search tree above X_k.

Value ordering heuristics provide information on future search efforts, which can be summarized by 2 parameters:

- T_i—the expected time to find a solution containing assignment $X_k = y_{ki}$ or verify that there are no such solutions;
- p_i—the "backtracking probability", that there will be no solution consistent with $X_k = y_{ki}$.

These are treated as the algorithm's subjective probabilities about future search in the current problem instance, rather than actual distributions over problem instances. Assuming correct values of these parameters, and independence of backtracks, the expected remaining search time in the subtree under X_k for ordering ω is given by:

$$T^{s|\omega} = T_{\omega(1)} + \sum_{i=2}^{|D_k|} T_{\omega(i)} \prod_{j=1}^{i-1} p_{\omega(j)} \tag{5.1}$$

In terms of rational metareasoning, the "current" optimal base-level action is picking the ω which optimizes $T^{s|\omega}$. Based on a general property of functions on sequences [MS79], it can be shown that $T^{s|\omega}$ is minimal if the values are sorted by increasing order of $\frac{T_i}{1-p_i}$.

A candidate heuristic H (with computation time T^H) generates an ordering by providing an updated (hopefully more precise) value of the parameters T_i, p_i for value assignments $X_k = y_{ki}$, which may lead to a new optimal ordering ω_H, corresponding to a new base-level action. The total expected remaining search time is given by:

$$T = T^H + E[T^{s|\omega_H}] \tag{5.2}$$

Since both T^H (the “time cost” of H in metareasoning terms) and $T^{s|\omega_H}$ contribute to T, even a heuristic that improves the estimates and ordering may not be useful. It may be better not to deploy H at all, or to update T_i, p_i only for some of the assignments. According to the rational metareasoning approach (Section 2.1), the intrinsic VOI Λ_i of estimating T_i, p_i for the ith assignment is the expected decrease in the expected search time:

$$\Lambda_i = \mathbb{E}\left[T^{s|\omega_-} - T^{s|\omega_{+i}}\right] \tag{5.3}$$

where ω_- is the optimal ordering based on priors, and ω_{+i} on values after updating T_i, p_i. Computing new estimates (with overhead T^c) for values T_i, p_i is beneficial just when the net VOI is positive:

$$V_i = \Lambda_i - T^c \tag{5.4}$$

To simplify estimation of Λ_i, the expected search time of an ordering is estimated as though the parameters are computed only for $\omega_-(1)$, i.e. for the first value in the ordering; essentially, this is the metareasoning subtree independence assumption. Other value assignments are assumed to have the prior (“default”) parameters $T_{\text{def}}, p_{\text{def}}$. Assuming w.l.o.g. that $\omega_-(1) = 1$:

$$T^{s|\omega_-} = T_1 + p_1 \sum_{i=2}^{|D_k|} T_{\text{def}} p_{\text{def}}^{i-2} = T_1 + p_1 T_{\text{def}} \frac{1 - p_{\text{def}}^{(|D_k|-1)}}{1 - p_{\text{def}}} \tag{5.5}$$

and the intrinsic VOI of the ith deliberation action is:

$$\Lambda_i = \mathbb{E}\left[G(T_i, p_i) \,\middle|\, \frac{T_i}{1-p_i} < \frac{T_1}{1-p_1}\right] \tag{5.6}$$

where $G(T_i, p_i)$ is the search time gain given the heuristically computed values T_i, p_i:

$$G(T_i, p_i) = T_1 - T_i + (p_1 - p_i) T_{\text{def}} \frac{1 - p_{\text{def}}^{(|D_k|-1)}}{1 - p_{\text{def}}} \tag{5.7}$$

In some cases, H provides estimates only for the expected search time T_i. In such cases, the backtracking probability p_i can be bounded by the Markov inequality as the probability for the given assignment that the time t to find a solution or to verify that no solution exists is at least the time T_i^{all} to find all solutions: $p_i = P\left(t \ge T_i^{all}\right) \le \frac{T_i}{T_i^{all}}$, and the bound can be used to estimate the probability:

$$p_i \approx \frac{T_i}{T_i^{all}} \tag{5.8}$$

Furthermore, note that in harder problems the probability of backtracking from variable X_k is proportional to $p_{\text{def}}^{(|D_k|-1)}$, and it is reasonable to assume that backtracking probabilities above X_k (trying values for $X_1, ..., X_{k-1}$) are still significantly greater than 0. Thus, the "default" backtracking probability p_{def} is close to 1, and consequently:

$$T_i^{all} \approx T_{\text{def}}, \quad \frac{1 - p_{\text{def}}^{(|D_k|-1)}}{1 - p_{\text{def}}} \approx |D_k| - 1 \tag{5.9}$$

By substituting (5.8), (5.9) into (5.7), estimate (5.10) for $G(T_i, p_i)$ is obtained:

$$\begin{aligned} G(T_i, p_i) &\approx T_1 - T_i + (\frac{T_1}{T_1^{all}} - \frac{T_i}{T_i^{all}})T_{\text{def}}\frac{1 - p_{\text{def}}^{(|D_k|-1)}}{1 - p_{\text{def}}} \\ &\approx T_1 - T_i + (\frac{T_1}{T_{\text{def}}} - \frac{T_i}{T_{\text{def}}})T_{\text{def}}\frac{1 - p_{\text{def}}^{(|D_k|-1)}}{1 - p_{\text{def}}} \\ &\approx T_1 - T_i + (T_1 - T_i)(|D_k| - 1) \\ &\approx (T_1 - T_i)|D_k| \end{aligned} \tag{5.10}$$

Finally, since (5.8), (5.9) imply that $T_i < T_1 \Leftrightarrow \frac{T_i}{1-p_i} < \frac{T_1}{1-p_1}$,

$$\Lambda_i \approx \mathbb{E}\left[(T_1 - T_i)|D_k| \,\middle|\, T_i < T_1\right] \tag{5.11}$$

5.2.4 VOI of Solution Count Estimates

The estimated solution count for an assignment may be used to estimate the expected time to find a solution for the assignment under the following assumptions[1]:

1. Solutions are roughly evenly distributed in the search space, that is, the distribution of time to find a solution can be modeled by a Poisson process.

[1] We do not claim that this is a valid model of CSP search; rather, we argue that even with such a crude model one can get significant runtime improvements.

2. Finding all solutions for an assignment $X_k = y_{ki}$ takes roughly the same time for all assignments to the variable X_k. Prior work [MSS97, KDG04] demonstrates that ignoring the differences in subproblem sizes is justified.

3. The expected time to find all solutions for an assignment divided by its solution count estimate is a reasonable estimate for the expected time to find a single solution.

Based on these assumptions, T_i can be estimated as $\frac{T^{all}}{|D_k|n_i}$ where T^{all} is the expected time to find all solutions for all values of X_k, and n_i is the solution count estimate for y_{ki}; likewise, $T_1 = \frac{T^{all}}{|D_k|n_{\max}}$, where $n_{\max}$ is the currently greatest n_i. By substituting the expressions for T_i, T_1 into (5.11), obtain as the intrinsic VOI of computing n_i:

$$\Lambda_i = T^{all} \sum_{n=n_{\max}}^{\infty} \left(\frac{1}{n_{\max}} - \frac{1}{n} \right) P(n, \nu) \tag{5.12}$$

where $P(n, \nu) = e^{-\nu} \frac{\nu^n}{n!}$ is the probability, according to the Poisson distribution, to find n solutions for a particular assignment when the mean number of solutions per assignment is $\nu = \frac{N}{|D_k|}$, and N is the estimated solution count for all values of X_k, computed at an earlier stage of the algorithm.

Neither T^{all} nor T^c, the time to estimate the solution count for an assignment, are known. However, for relatively low solution counts, when an invocation of the heuristic has high intrinsic VOI, both T^{all} and T^c are mostly determined by the time spent eliminating non-solutions. Therefore, T^c can be assumed approximately proportional to $\frac{T^{all}}{|D_k|}$, the average time to find all solutions for a single assignment, with an unknown factor $\gamma < 1$:

$$T^c \approx \gamma \frac{T^{all}}{|D_k|} \tag{5.13}$$

Then, T^{all} can be eliminated from both T^c and Λ. Following Equation (5.4), the solution count should be estimated whenever the net VOI is positive:

$$V(n_{\max}) \propto |D_k| e^{-\nu} \sum_{n=n_{\max}}^{\infty} \left(\frac{1}{n_{\max}} - \frac{1}{n} \right) \frac{\nu^n}{n!} - \gamma \tag{5.14}$$

The infinite series in (5.14) rapidly converges, and an approximation of the sum can be computed efficiently. As done in Section 5.2.5, γ can be learned offline from a set of problem instances of a certain kind for the given implementation of the search algorithm and the solution counting heuristic.

Algorithm 7 implements rational value ordering. The procedure receives problem instance *csp* with assigned values for variables $X_1, ..., X_{k-1}$, variable X_k to be ordered, and estimate N of the number of solutions of the problem instance (line 1); N is computed at the previous step of the backtracking algorithm as the solution count estimate for the chosen assignment for X_{k-1}, or, if $k = 1$, at the beginning of the search as the total solution count estimate for the instance. Solution counts n_i for some of the assignments are estimated (lines 4–10) by selectively invoking the heuristic computation EstimateSolutionCount (line 8), and then the domain of X_k, ordered by non-increasing solution count estimates of value assignments, is returned (lines 12–13).

Algorithm 7 Value Ordering via Solution Count Estimation

```
 1: procedure ValueOrdering-SC($csp, X_k, N$)
 2:     $D \leftarrow D_k, \quad n_{\max} \leftarrow \frac{N}{|D|}$
 3:     for all $i$ in $1..|D|$ do $n_i \leftarrow n_{\max}$
 4:     while $V(n_{\max}) > 0$ do                          ▷ using Equation (5.14)
 5:         choose $y_{ki} \in D$ arbitrarily
 6:         $D \leftarrow D \setminus \{y_{ki}\}$
 7:         $csp' \leftarrow csp$ with $D_k = \{y_{ki}\}$
 8:         $n_i \leftarrow$ EstimateSolutionCount($csp'$)
 9:         if $n_i > n_{\max}$ then $n_{\max} \leftarrow n_i$
10:     end while
11:     end while
12:     $D_{ord} \leftarrow$ sort $D_k$ by non-increasing $n_i$
13:     return $D_{ord}$
14: end procedure
```

5.2.5 Empirical Evaluation

Specifying the algorithm parameter γ is the first issue. γ should be a characteristic of the implementation of the search algorithm, rather than of the problem instance; it is also desirable that the performance of the algorithm not be too sensitive to fine tuning of this parameter.

Most of the experiments were conducted on sets of random problem instances generated according to Model RB [XL00]. The empirical evaluation was performed in two stages. In the first stage, several benchmarks were solved for a wide range of values of γ, and an appropriate value for γ was chosen. In the second stage, the search was run on two sets of problem instances with the chosen γ, as well as with exhaustive deployment, and with the minimum conflicts heuristic, and the search time distributions were compared for each of the value-ordering heuristics.

The AC-3 version of MAC was used for the experiments, with some modifications [SF97]. Variables were ordered using the maximum degree variable ordering heuristic.[2] The value-ordering heuristic was based on a version of the solution count estimate proposed in [MSS97]. The version used here was optimized for better computation time for overconstrained problem instances. As a result, Equation (5.13) is a reasonable approximation for this implementation. The source code is available from `http://ftp.davidashen.net/vsc.tar.gz`.

Benchmarks

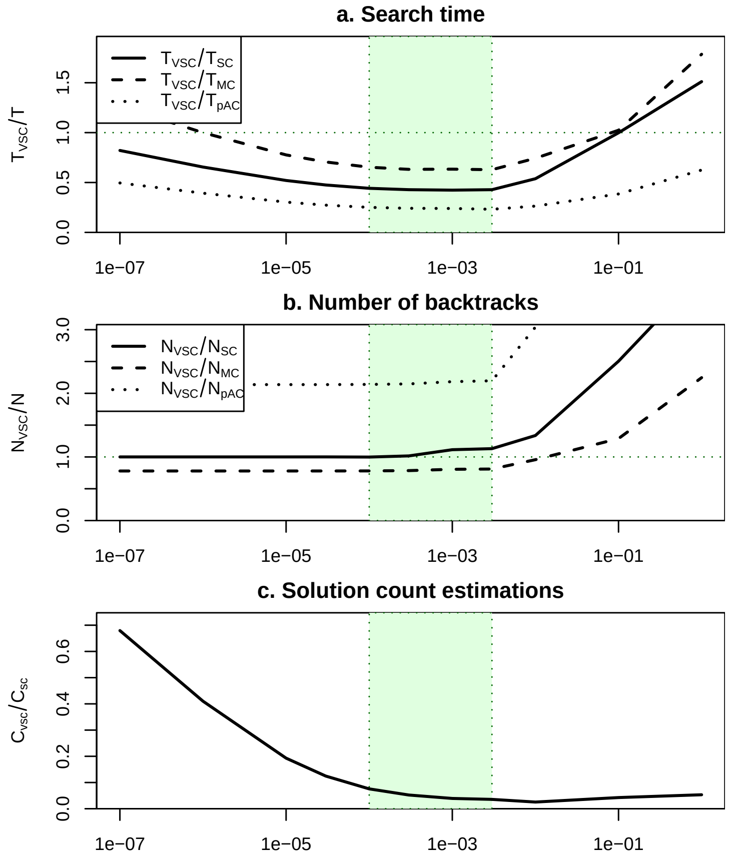

Figure 5.1: Influence of γ in CSP benchmarks

CSP benchmarks from CSP Solver Competition 2005 [BHL05] were used. 14 out of 26

[2] A dynamic variable ordering heuristic, such as dom/deg, may result in shorter search times in general, but gave no significant improvement in our experiments; on the other hand, static variable ordering simplifies the analysis.

benchmarks solved by at least one of the solvers submitted for the competition could be solved with 30 minutes timeout by the solver used for this empirical study for all values of γ: $\gamma = 0$ and the exponential range $\gamma \in \{10^{-7}, 10^{-6}, ..., 1\}$, as well as with the minimum-conflicts heuristic and the pAC heuristic.

Figure 5.1.a shows the mean search time of VOI-driven solution count estimate deployment T_{VSC} normalized by the search time of exhaustive deployment T_{SC} ($\gamma = 0$), for the minimum conflicts heuristic T_{MC}, and for the pAC heuristic T_{PAC}. The shortest search time on average is achieved by VSC for $\gamma \in [10^{-4}, 3 \cdot 10^{-3}]$ (shaded in the figure) and is much shorter than for SC (mean $\left(\frac{T_{VSC}(10^{-3})}{T_{SC}}\right) \approx 0.45$); the improvement was actually close to the "ideal" of getting all the information provided by the heuristic without paying the overhead at all. For all but one of the 14 benchmarks the search time for VSC with $\gamma = 3 \cdot 10^{-3}$ is shorter than for MC. For most values of γ, VSC gives better results than MC ($\frac{T_{VSC}}{T_{MC}} < 1$). pAC always results in the longest search time due to the computational overhead.

Figure 5.1.b shows the mean number of backtracks of VOI-driven deployment N_{VSC} normalized by the number of backtracks of exhaustive deployment N_{SC}, the minimum conflicts heuristic N_{MC}, and for the pAC heuristic $\overline{N}_{pAC}$. VSC causes less backtracking than MC for $\gamma \leq 3 \cdot 10^{-3}$ ($\frac{N_{VSC}}{N_{MC}} < 1$). pAC always causes less backtracking than other heuristics, but has overwhelming computational overhead.

Figure 5.1.c shows C_{VSC}, the number of estimated solution counts of VOI-driven deployment, normalized by the number of estimated solution counts of exhaustive deployment C_{SC}. When $\gamma = 10^{-3}$ and the best search time is achieved, the solution counts are estimated only in a relatively small number of search states: the average number of estimations is ten times smaller than in the exhaustive case (mean $\left(\frac{C_{VSC}(10^{-3})}{C_{SC}}\right) \approx 0.099$, median $\left(\frac{C_{VSC}(10^{-3})}{C_{SC}}\right) \approx 0.048$).

The results show that although the solution counting heuristic may provide significant improvement in the search time, further improvement is achieved when the solution count is estimated only in a *small fraction of occasions* selected using rational metareasoning.

Random instances

Based on the results on benchmarks, we chose $\gamma = 10^{-3}$, and applied it to two sets of 100 problem instances. Exhaustive deployment, rational deployment, the minimum conflicts heuristic, and probabilistic arc consistency were compared.

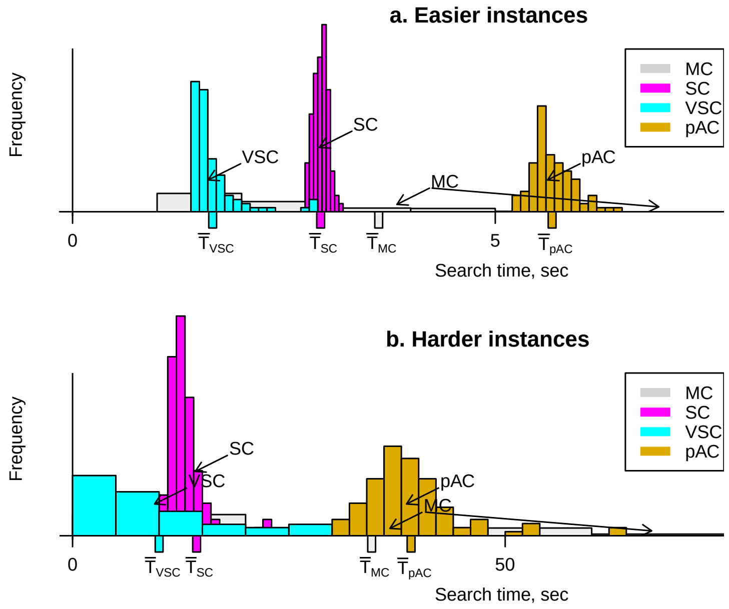

Figure 5.2: Search time comparison on sets of random instances (using Model RB)

The first, easier, set was generated with 30 variables, 30 values per domain, 280 constraints, and 220 nogood pairs per constraint ($p = 0.24$, $p_{crit} = 0.30$). Search time distributions are presented in Figure 5.2.a. The shortest mean search time is achieved for rational deployment, with exhaustive deployment next ($\frac{\overline{T}_{SC}}{\overline{T}_{VSC}} \approx 1.75$), followed by the minimum conflicts heuristic ($\frac{\overline{T}_{MC}}{\overline{T}_{VSC}} \approx 2.16$) and probabilistic arc consistency ($\frac{\overline{T}_{pAC}}{\overline{T}_{VSC}} \approx 3.42$). Additionally, while the search time distributions for solution counting are sharp ($\frac{\max T_{SC}}{\overline{T}_{SC}} \approx 1.08$, $\frac{\max T_{VSC}}{\overline{T}_{VSC}} \approx 1.73$), the distribution for the minimum conflicts heuristic has a long tail with a much longer worst case time ($\frac{\max T_{VSC}}{\overline{T}_{VSC}} \approx 5.67$).

The second, harder, set was generated with 40 variables, 19 values, 410 constraints, 90 nogood pairs per constraint (exactly at the phase transition: $p = p_{crit} = 0.25$). Search time distributions are presented in Figure 5.2.b. As with the first set, the shortest mean search time is achieved for rational deployment: $\frac{\overline{T}_{SC}}{\overline{T}_{VSC}} \approx 1.43$, while the relative mean search time for the minimum conflicts heuristic is much longer: $\frac{\overline{T}_{MC}}{\overline{T}_{VSC}} \approx 3.45$. The probabilistic arc consistency heuristic resulted again in the longest search time due to the overhead of computing relative solution count estimates by loopy belief propagation: $\frac{\max T_{VSC}}{\overline{T}_{VSC}} \approx 3.91$.

Thus, the value of γ chosen based on a small set of hard instances gives good results on a set of instances with different parameters and of varying hardness.

Generalized Sudoku

Randomly generated problem instances have played a key role in the design and study of heuristics for CSP. However, one might argue that the benefits of our scheme are specific to model RB. Indeed, real-world problem instances often have much more structure than random instances generated according to Model RB. Hence, we repeated the experiments on randomly generated Generalized Sudoku instances [ABF$^+$06], since this domain is highly structured, and thus a better source of realistic problems with a controlled measure of hardness.

The search was run on two sets of 100 Generalized Sudoku instances, with 4x3 tiles and 90 holes and with 7x4 tiles and 357 holes, with holes punched using the doubly balanced method [ABF$^+$06]. The search was repeated on each instance with the exhaustive solution-counting, VOI-driven solution counting (with the same value of $\gamma = 10^{-3}$ as for the RB model problems), minimum conflicts, and probabilistic arc consistency value ordering heuristics. Results are summarized in Table 5.1 and show that relative performance of the methods on Generalized Sudoku is similar to the performance on Model RB.

	$\overline{T_{SC}}$, sec	$\overline{\left(\frac{T_{VSC}}{T_{SC}}\right)}$	$\overline{\left(\frac{T_{MC}}{T_{SC}}\right)}$	$\overline{\left(\frac{T_{pAC}}{T_{SC}}\right)}$
4x3, 90 holes	1.809	0.755	1.278	22.421
7x4, 357 holes	21.328	0.868	3.889	3.826

Table 5.1: Generalized Sudoku

Deployment patterns

One might ask whether trivial methods for selective deployment would work, such as estimating solution counts for a certain number of assignments in the beginning of the search. We examined deployment patterns of VOI-driven SC with ($\gamma = 10^{-3}$) on several instances of different hardness. For all instances, the solution counts were estimated at varying rates during *all stages* of the search, and the deployment patterns differed between the instances, so a simple deployment scheme seems unlikely.

VOI-driven deployment also compares favorably to random deployment. Table 5.2 shows performance of VOI-driven deployment for $\gamma = 10^{-3}$ and of uniform random deployment, with total number of solution count estimations equal to that of the VOI-driven deployment. For both schemes, the values for which solution counts were not estimated were ordered randomly, and the search was repeated 20 times. The mean search time for the random deployment is ≈ 1.6 times longer than for the VOI-driven deployment, and has ≈ 100 times greater standard deviation.

	mean(T), sec	median(T), sec	sd(T), sec
VOI-driven	19.841	19.815	0.188
random	31.421	42.085	20.038

Table 5.2: VOI-driven vs. random deployment

5.2.6 Conclusion and Further Research

This study suggests a model for adaptive deployment of value ordering heuristics in algorithms for constraint satisfaction problems. The approach presented here does not attempt to introduce new heuristics or solution-count estimates; rather, an "off the shelf" heuristic is deployed selectively based on value of information, thereby significantly reducing the heuristic's "effective" computational overhead, with an improvement in performance for problems of different size and hardness. As a case study, the model was applied to a value-ordering heuristic based on solution count estimates, and a steady improvement in the overall algorithm performance was achieved compared to *always* computing the estimates, as well as to other simple deployment tactics. The experiments showed that for many problem instances the optimum performance is achieved when solution counts are estimated only in a relatively small number of search states.

The methods introduced in this study can be extended in numerous ways. First, generalization of the VOI to deploy different types of heuristics for CSP, such as variable ordering heuristics, as well as reasoning about deployment of more than one heuristic at a time, are natural non-trivial extensions. Second, an explicit evaluation of the quality of the distribution model is an interesting issue, coupled with a better candidate model of the distribution. Such distribution models can also employ more disciplined statistical learning methods in tandem, as suggested above. Finally, applying the methods suggested here to search in other domains can be attempted, especially to heuristics for planning. In particular, examining whether the metareasoning scheme can improve reasoning over deployment of heuristics based solely on learning methods is an interesting research issue.

5.3 VOI-aware Monte-Carlo Tree Search

5.3.1 Introduction

MCTS, and especially UCT [KS06] appears in numerous search applications, such as [EKH10]. Although these methods are shown to be successful empirically, most authors appear to be using UCT "because it has been shown to be successful in the past", and "because it does a good job of trading off exploration and exploitation". While the latter statement may be correct for the Multi-armed Bandit problem and for the UCB1 algorithm [ACBF02], we argue that a simple reconsideration from basic principles can result in schemes that outperform UCT.

The core issue is that in MCTS for adversarial search and search in "games against nature" the goal is typically to find the best first action of a good (or even optimal) policy, which is closer to minimizing the simple regret, rather than the cumulative regret minimized by UCB1. However, the simple and the cumulative regret cannot be minimized simultaneously; moreover, [BMS11] shows that in many cases the smaller the cumulative regret, the greater the simple regret.

We begin with background definitions and related work. VOI estimates for arm pulls in MAB are presented, and a VOI-aware sampling policy is suggested, both for the simple regret in MAB and for MCTS. Finally, the performance of the proposed sampling policy is evaluated on sets of Bernoulli arms and on Computer GO, showing the improved performance.

5.3.2 Background and Related Work

Monte-Carlo tree search was initially suggested as a scheme for finding approximately optimal policies for Markov Decision Processes (MDP). MCTS explores an MDP by performing *rollouts*—trajectories from the current state to a state in which a termination condition is satisfied (either the goal or a cutoff state).

Taking a sequence of samples in order to minimize the regret of a decision based on the samples is captured by the Multi-armed Bandit problem (MAB) [VM05]. In MAB, we have a set of K arms. Each arm can be pulled multiple times. When the ith arm is pulled, a random reward X_i from an unknown stationary distribution is encountered. In the *cumulative setting*, all encountered rewards are collected. UCB1 [ACBF02] was shown to be near-optimal in this respect. UCT, an extension of UCB1 to MCTS is described in [KS06], and shown to outperform many state of the art search algorithms in both MDP and adversarial search [GW06, EKH10]. In the *simple regret setting*, the agent gets to collect only the reward of the last pull.

Definition. *The **simple regret** of a sampling policy for MAB is the expected difference between the best expected reward μ_* and the expected reward μ_j of the empirically best arm $\overline{X}_j = \max_i \overline{X}_i$:*

$$\mathbb{E}r = \sum_{j=1}^{K} \Delta_j \Pr(\overline{X}_j = \max_i \overline{X}_i) \tag{5.15}$$

where $\Delta_j = \mu_* - \mu_j$.

Strategies that minimize the simple regret are called pure exploration strategies [BMS11].

A different scheme for control of sampling can use the principles of rational metareasoning (Section 2.1). In search, one maintains a current best action α, and finds the expected gain from finding another action β to be better than the current best.

5.3.3 Upper bounds on Value of Information

In many practical applications of the selection problem, such as search in the game of Go, prior distributions are unavailable.[3] In such cases, one can still bound the value of information of myopic policies using *concentration inequalities* to derive distribution-independent bounds on the VOI. We obtain such bounds under the following assumptions:

1. Samples are iid given the value of the arms (variables), as in the Bayesian schemes such as Bernoulli sampling.

2. The expectation of a selection in a belief state is equal to the sample mean (and therefore, after sampling terminates, the arm with the greatest sample mean will be selected).

When considering possible samples in the blinkered semi-myopic setting, two cases are possible: either the arm α with the highest sample mean $\overline{X}_\alpha$ is tested, and $\overline{X}_\alpha$ becomes lower than $\overline{X}_\beta$ of the second-best arm β; or, another arm i is tested, and $\overline{X}_i$ becomes higher than $\overline{X}_\alpha$.

Our bounds below are applicable to any bounded distribution (without loss of generality bounded in $[0, 1]$). Similar bounds can be derived for certain unbounded distributions, such as the normally distributed prior value with normally distributed sampling. We derive a VOI

[3] The analysis is also applicable to some Bayesian settings, using “fake” samples to simulate prior distributions.

bound for testing an arm a fixed N times, where N can be the remaining budget of available samples or any other integer quantity. Denote by Λ_i^b the intrinsic VOI of testing the ith arm N times, and the number of samples already taken from the ith arm by n_i.

Theorem 1. *Λ_i^b is bounded from above as*

$$\begin{aligned}\Lambda_{i|i\neq\alpha}^b &\le \frac{N(1-\overline{X}_\alpha^{n_\alpha})}{n_i}\Pr(\overline{X}_i^{n_i+N}\ge\overline{X}_\alpha^{n_\alpha})\\ \Lambda_\alpha^b &\le \frac{N\overline{X}_\beta^{n_\beta}}{n_\alpha}\Pr(\overline{X}_\alpha^{n_\alpha+N}\le\overline{X}_\beta^{n_\beta})\end{aligned}\tag{5.16}$$

Proof. For the case $i\neq\alpha$, the probability that the ith arm is finally chosen instead of α is $\Pr(\overline{X}_i^{n_i+N}\ge\overline{X}_\alpha^{n_\alpha})$. $X_i\le 1$, therefore $\overline{X}_i^{n_i+N}\le\overline{X}_\alpha^{n_\alpha}+\frac{N(1-\overline{X}_\alpha^{n_\alpha})}{N+n_i}$. Hence, the intrinsic value of blinkered information is at most:

$$\begin{aligned}&\frac{N(1-\overline{X}_\alpha^{n_\alpha})}{N+n_i}\Pr(\overline{X}_i^{n_i+N}\ge\overline{X}_\alpha^{n_\alpha})\\ &\qquad\le\frac{N(1-\overline{X}_\alpha^{n_\alpha})}{n_i}\Pr(\overline{X}_i^{n_i+N}\ge\overline{X}_\alpha^{n_\alpha})\end{aligned}\tag{5.17}$$

Proof for the case $i=\alpha$ is similar. □

The probabilities can be bounded from above using the Hoeffding inequality [Hoe63]:

Theorem 2. *For any $A\ge\overline{X}_{i|i\neq\alpha}^{n_i}$ and $B\le\overline{X}_\alpha^{n_\alpha}$ the probabilities $\Pr(\overline{X}_{i|i\neq\alpha}^{n_i+N}\ge A)$, $\Pr(\overline{X}_\alpha^{n_\alpha+N}\le B)$ are bounded from above as*

$$\begin{aligned}\Pr(\overline{X}_{i|i\neq\alpha}^{n_i+N}\ge A)&\le 2\exp\left(-\varphi(A-\overline{X}_i^{n_i})^2n_i\right)\\ \Pr(\overline{X}_\alpha^{n_\alpha+N}\le B)&\le 2\exp\left(-\varphi(\overline{X}_\alpha^{n_\alpha}-B)^2n_\alpha\right)\end{aligned}\tag{5.18}$$

where $\varphi=\min\left(2(\frac{1+n/N}{1+\sqrt{n/N}})^2\right)=8(\sqrt{2}-1)^2>1.37$.

Proof. Equation (5.18) follows from the observation that $\overline{X}_i^{n_i+N}>A$ if and only if the mean $\overline{X}_i^N$ of N samples from n_i+1 to n_i+N is at least $A+(A-\overline{X}_i^{n_i})\frac{n_i}{N}$.

For any δ, the probability that $\overline{X}_i^{n_i+N}$ is greater than A is less than the probability that $\mathbb{E}[X_i]\ge\overline{X}_i^n+\delta$ *or* $\overline{X}_i^N\ge\mathbb{E}[X_i]+A-\overline{X}_i^{n_i}-\delta+(A-\overline{X}_i^{n_i})\frac{n_i}{N}$, thus, by the union bound, less than the sum of the probabilities:

$$\begin{aligned}
&\Pr(\overline{X}_i^{n_i+N} \ge A) \\
&\quad \le \Pr(\mathbb{E}[X_i] - \overline{X}_i^{n_i} \ge \delta) \qquad (5.19)\\
&\quad + \Pr\left(\overline{X}_i^{N} - \mathbb{E}[X_i] \ge A - \overline{X}_i^{n_i} - \delta + (A - \overline{X}_i^{n_i})\frac{n_i}{N}\right)
\end{aligned}$$

Bounding the probabilities on the right-hand side using the Hoeffding inequality, obtain:

$$\begin{aligned}
&\Pr(\overline{X}_i^{n_i+N} \ge A) \\
&\quad \le \exp(-2\delta^2 n_i) \\
&\quad + \exp\left(-2\left((A - \overline{X}_i^{n_i})\left(1 + \frac{n_i}{N}\right) - \delta\right)^2 N\right) \qquad (5.20)
\end{aligned}$$

Find δ for which the two terms on the right-hand side of Equation (5.20) are equal:

$$\exp(-\delta^2 n) = \exp\left(-2\left((A - \overline{X}_i^{n_i})(1 + \frac{n_i}{N}) - \delta\right)^2 N\right) \qquad (5.21)$$

Solve Equation (5.21) for δ: $\delta = \frac{1+\frac{n_i}{N}}{1+\sqrt{\frac{n_i}{N}}}(A - \overline{X}_i^{n_i}) \ge 2(\sqrt{2}-1)(A - \overline{X}_i^{n_i})$. Substitute δ into Equation (5.20) and obtain

$$\begin{aligned}
&\Pr(\overline{X}_i^{n_i+N} \ge A) \\
&\quad \le 2\exp\left(-2\left(\frac{1+\frac{n_i}{N}}{1+\sqrt{\frac{n_i}{N}}}(\overline{X}_\alpha^{n_\alpha} - \overline{X}_i^{n_i})\right)^2 n_i\right) \\
&\quad \le 2\exp(-8(\sqrt{2}-1)^2(A - \overline{X}_i^{n_i})^2 n_i) \\
&\quad = 2\exp(-\varphi(A - \overline{X}_i^{n_i})^2 n_i) \qquad (5.22)
\end{aligned}$$

Derivation for the case $i = \alpha$ is similar. □

Corollary 3. *An upper bound on the VOI estimate Λ_i^b is obtained by substituting Equa-*

tion (5.18) into (5.16).

$$\begin{aligned}\Lambda^b_{i|i\neq\alpha} \leq \hat{\Lambda}^b_i &= \frac{2N(1-\overline{X}^{n_\alpha}_\alpha)}{n_i}\exp\left(-\varphi(\overline{X}^{n_\alpha}_\alpha - \overline{X}^{n_i}_i)^2 n_i\right) \\ \Lambda^b_\alpha \leq \hat{\Lambda}^b_\alpha &= \frac{2N\overline{X}^{n_\beta}_\beta}{n_\alpha}\exp\left(-\varphi(\overline{X}^{n_\alpha}_\alpha - \overline{X}^{n_\beta}_\beta)^2 n_\alpha\right)\end{aligned} \tag{5.23}$$

More refined bounds can be obtained through tighter estimates on the probabilities in Equation (5.16), for example, based on the empirical Bernstein inequality [MP09], or through a more careful application of the Hoeffding inequality, resulting in Theorem 4:

Theorem 4. Λ^b_i *is bounded from above as*

$$\begin{aligned}\Lambda^b_{i|i\neq\alpha} &\leq \frac{\sqrt{\pi}}{\sqrt{\varphi n_i}}\left[\operatorname{erf}\left(\left(\overline{X}^{n_\alpha}_\alpha + \frac{N}{N+n_i}(1-\overline{X}^{n_\alpha}_\alpha) - \overline{X}^{n_i}_i\right)\sqrt{\varphi n_i}\right) - \operatorname{erf}\left((\overline{X}_\alpha - \overline{X}^{n_i}_i)\sqrt{\varphi n_i}\right)\right] \\ \Lambda^b_\alpha &\leq \frac{\sqrt{\pi}}{\sqrt{\varphi n_\alpha}}\left[\operatorname{erf}\left(\left(\overline{X}^{n_\alpha}_\alpha - \overline{X}^{n_\beta}_\beta + \frac{N}{N+n_\alpha}\overline{X}^{n_\beta}_\beta\right)\sqrt{\varphi n_\alpha}\right) - \operatorname{erf}\left((\overline{X}^{n_\alpha}_\alpha - \overline{X}^{n_\beta}_\beta)\sqrt{\varphi n_\alpha}\right)\right]\end{aligned} \tag{5.24}$$

Proof. By definition,

$$\Lambda^b_i = \int\limits_{\overline{X}_\alpha}^{\overline{X}^{n_\alpha}_\alpha + \frac{N}{N+n_i}(1-\overline{X}^{n_\alpha}_\alpha)} (x - \overline{X}^{n_\alpha}_\alpha)\Pr\left(\overline{X}^{n_i+N}_i = x\right)dx \tag{5.25}$$

Integrating by parts, obtain

$$\begin{aligned}\Lambda^b_i =& -(x-\overline{X}^{n_\alpha}_\alpha)\Pr\left(\overline{X}^{n_i+N}_i \geq x\right)\Big|^{\overline{X}^{n_\alpha}_\alpha + \frac{N}{N+n_i}(1-\overline{X}^{n_\alpha}_\alpha)}_{\overline{X}^{n_\alpha}_\alpha} + \int\limits_{\overline{X}_\alpha}^{\overline{X}^{n_\alpha}_\alpha + \frac{N}{N+n_i}(1-\overline{X}^{n_\alpha}_\alpha)} \Pr\left(\overline{X}^{n_i+N}_i \geq x\right)dx \\ =& \int\limits_{\overline{X}_\alpha}^{\overline{X}^{n_\alpha}_\alpha + \frac{N}{N+n_i}(1-\overline{X}^{n_\alpha}_\alpha)} \Pr\left(\overline{X}^{n_i+N}_i \geq x\right)dx\end{aligned} \tag{5.26}$$

Substituting the bound on $\Pr\left(\overline{X}_i^{n_i+N} \geq x\right)$ from Theorem 2 into (5.26), finally obtain:

$$
\begin{aligned}
\Lambda_i^b \leq & \int\limits_{\overline{X}_\alpha}^{\overline{X}_\alpha^{n_\alpha}+\frac{N}{N+n_i}(1-\overline{X}_\alpha^{n_\alpha})} 2\exp\left(-\varphi(x-\overline{X}_i^{n_i})^2 n_i\right)dx \\
= & \frac{\sqrt{\pi}}{\sqrt{\varphi n_i}}\left[\operatorname{erf}\left(\left(\overline{X}_\alpha^{n_\alpha}+\frac{N}{N+n_i}(1-\overline{X}_\alpha^{n_\alpha})-\overline{X}_i^{n_i}\right)\sqrt{\varphi n_i}\right)-\operatorname{erf}\left((\overline{X}_\alpha-\overline{X}_i^{n_i})\sqrt{\varphi n_i}\right)\right]
\end{aligned}
\tag{5.27}
$$

Derivation for the case $i = \alpha$ is similar. □

Selection problems usually separate out the decision of *whether* to sample or to stop (called the stopping policy), and *what* to sample. We'll examine the first issue here, along with the empirical evaluation of the above approximate algorithms, and the second in the following section.

Assuming that the sample costs are constant, a semi-myopic policy will decide to test the arm that has the best current VOI estimate. When the distributions are unknown, it makes sense to use the upper bounds established in Theorem 1, as we do in the following. This evaluation assumes a fixed budget of samples, which is completely used up by each of the candidate schemes, making a stopping criterion irrelevant.

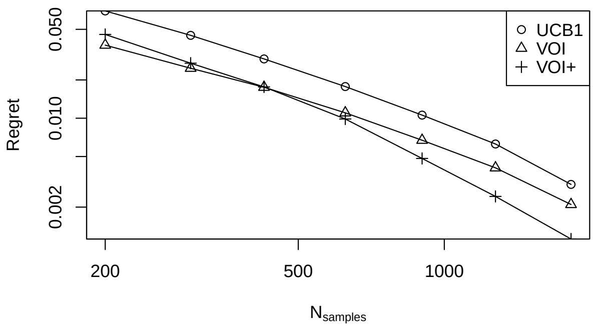

Figure 5.3: Average regret of various policies as a function of the fixed number of samples in a 25-action Bernoulli sampling problem, over 10000 trials.

The sampling policies are compared on random Bernoulli selection problem instances. Figure 5.3 shows results for randomly-generated selection problems with 25 Bernoulli arms, where the mean rewards of the arms are distributed uniformly in $[0, 1]$, for a range of sample budgets 200..2000, with multiplicative step of 2, averaging over 10000 trials. We compare UCB1 with the policies based on the bounds in Equation (5.23) (VOI) and Equation (5.24) (VOI+).

UCB1 is always considerably worse than the VOI-aware sampling policies.

5.3.4 Sampling in trees

The previous section addressed the selection problem in the flat case. Selection in trees is more complicated. The goal of Monte-Carlo tree search [CBSS08] at the root node is usually to select an action that appears to be the best based on outcomes of *search rollouts*. But the goal of rollouts at non-root nodes is different than at the root: here it is important to better approximate the value of the node, so that selection at the root can be more informed. The exact analysis of sampling at internal nodes is outside the scope of this study. At present we have no better proposal for internal nodes than to use UCT there.

We thus propose the following hybrid sampling scheme [TS12a]: at the *root node*, sample based on the VOI estimate; at *non-root nodes*, sample using UCT.

Strictly speaking, even at the root node the stationarity assumptions underlying our belief-state MDP for selection do not hold exactly. UCT is an adaptive scheme, and therefore the values generated by sampling at non-root nodes will typically cause values observed at children of the root node to be non-stationary. Nevertheless, sampling based on VOI estimates computed as for stationary distributions works well in practice. As illustrated by the empirical evaluation (Section 5.3.4), estimates based on upper bounds on the VOI result in good sampling policies, which exhibit performance comparable to the performance of some state-of-the-art heuristic algorithms.

Stopping criterion

When a sample has a known cost commensurable with the value of information of a measurement, an upper bound on the intrinsic VOI can also be used to stop the sampling if the intrinsic VOI of any arm is less than the total cost of sampling C: $\max_i \Lambda_i \le C$.

The VOI estimates of Equations (5.16) and (eqn:mcts-bound-blnk-hoeffding) include the remaining sample budget N as a factor, but given the cost of a single sample c, the cost of the remaining samples accounted for in estimating the intrinsic VOI is $C = cN$. N can be dropped on both sides of the inequality, giving a reasonable stopping criterion:

$$\begin{aligned}\frac{1}{N}\Lambda_\alpha^b \le & \frac{\overline{X}_\beta^{n_\beta}}{n_\alpha}\Pr(\overline{X}_\alpha^{n_\alpha+N} \le \overline{X}_\beta^{n_\alpha}) \le c\\ \frac{1}{N}\max_i \Lambda_i^b \le & \max_i \frac{(1-\overline{X}_\alpha^{n_\alpha})}{n_i}\Pr(\overline{X}_i^{n_i+N} \ge \overline{X}_\alpha^{n_\alpha}) \le c\\ & \forall i : i \neq \alpha \end{aligned} \tag{5.28}$$

The empirical evaluation (Section 5.3.4) confirms the viability of this stopping criterion and illustrates the influence of the sample cost c on the performance of the sampling policy. When the sample cost c is unknown, one can perform initial calibration experiments to determine a reasonable value, as done in the following.

Sample redistribution in trees

The above hybrid approach assumes that the information obtained from rollouts in the current state is discarded after an real-world action is selected. In practice, many successful Monte-Carlo tree search algorithms reuse rollouts generated at earlier search states, if the sample traverses the current search state during the rollout; thus, the value of information of a rollout is determined not just by the influence on the choice of the action at the current state, but also by its potential influence on the choice at future search states.

One way to account for this reuse would be to incorporate the 'future' value of information into a VOI estimate. However, this approach requires a nontrivial extension of the theory of metareasoning for search. Alternately, one can behave myopically with respect to the search tree depth:

1. Estimate VOI as though the information is discarded after each step,
2. Stop early if the VOI is below a certain threshold (see Section 5.3.4), and
3. Save the unused sample budget for search in future states, such that if the nominal budget is N, and the unused budget in the last state is N_u, the search budget in the next state will be $N + N_u$.

In this approach, the cost c of a sample in the current state is the VOI of increasing the budget of a future state by one sample. It is unclear whether this cost can be accurately estimated, but supposing a fixed value for a given problem type and algorithm implementation would work. Indeed, the empirical evaluation (Section 5.3.4) confirms that stopping and sample redistribution based on a learned fixed cost substantially improve the performance of the VOI-based sampling policy in game tree search.

Playing Go against UCT

The hybrid policies were compared on the game Go, a search domain in which UCT-based MCTS has been particularly successful [GW06]. A modified version of Pachi [BLG11], a state of the art Go program, was used for the experiments:

- The UCT engine of Pachi was extended with VOI-aware sampling policies at the first step.
- The stopping criterion for the VOI-aware policy was modified and based solely on the sample cost, specified as a constant parameter. The heuristic stopping criterion for the original UCT policy was left unchanged.
- The time-allocation model based on the fixed number of samples was modified for *both the original UCT policy and the VOI-aware policies* such that
  - Initially, the same number of samples is available to the agent at each step, independently of the number of pre-simulated games;
  - If samples were unused at the current step, they become available at the next step.

While the UCT engine is not the most powerful engine of Pachi, it is still a strong player. On the other hand, additional features of more advanced engines would obstruct the MCTS phenomena which are the subject of the experiment. The engines were compared on the 9x9 board, for 5000, 7000, 1000, and 15000 samples (game simulations) per ply, each experiment repeated 1000 times. Figure 5.4 depicts a calibration experiment, showing the winning rate of the VOI-aware policy against UCT as a function of the stopping threshold c (if the maximum VOI of a sample is below the threshold, the simulation is stopped, and a move is chosen). Each curve in the figure corresponds to a certain number of samples per ply. For the stopping threshold of 10^{-6}, the VOI-aware policy is almost always better than UCT, and reaches the winning rate of 64% for 10000 samples per ply.

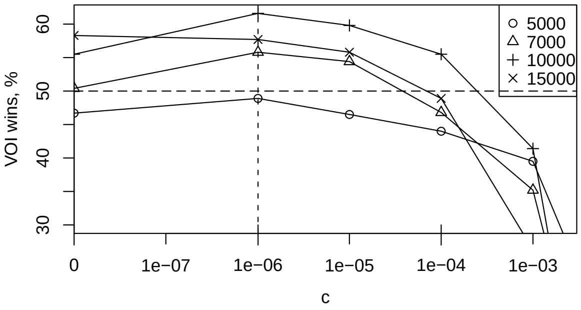

Figure 5.4: Winning rate of the VOI-aware policy in Go as a function of the cost c, for varying numbers of samples per ply.

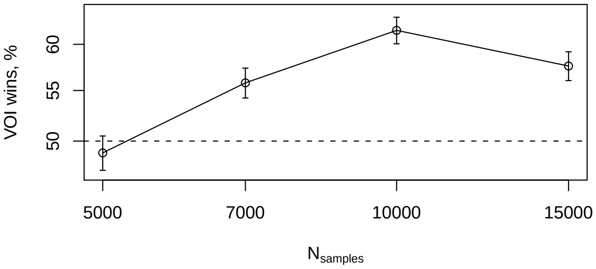

Figure 5.5: Winning rate of the VOI-aware policy in Go as a function of the number of samples, fixing cost $c = 10^{-6}$.

Figure 5.5 shows the winning rate of VOI against UCT $c = 10^{-6}$. In agreement with the intuition (Figure 5.3.4), VOI-based stopping and sample redistribution is most influential for intermediate numbers of samples per ply. When the maximum number of samples is too low, early stopping would result in poorly selected moves. On the other hand, when the maximum number of samples is sufficiently high, the VOI of increasing the maximum number of samples in a future state is low.

Note that if we disallowed reuse of samples in both Pachi and in our VOI-based scheme, the VOI based-scheme win rate is even higher than shown in Figure 5.5. This is as expected, as this setting (which is somewhat unfair to Pachi) is closer to meeting the assumptions underlying the selection MDP.

5.3.5 Conclusion and Further Research

This work suggested a Monte-Carlo sampling policy in which sample selection is based on upper bounds on the value of information. Empirical evaluation showed that this policy outperforms heuristic algorithms for pure exploration in MAB, as well as for MCTS.

MCTS still remains a largely unexplored field of application of VOI-aware algorithms. More elaborate VOI estimates, taking into consideration re-use of samples in future search states should be considered. The policy introduced here differs from the UCT algorithm only at the first step, where the VOI-aware decisions are made. Consistent application of principles of rational metareasoning at all steps of a rollout may further improve the sampling.

5.4 Towards Rational Deployment of Multiple Heuristics in A*

5.4.1 Introduction

This study examines improvements to the A^* algorithm when we have several available admissible heuristics. Clearly, we can evaluate all these heuristics, and use their *maximum* as an admissible heuristic, a scheme we call A^*_{MAX}. The problem with naive maximization is that all the heuristics are computed for all the generated nodes. In order to reduce the time spent on heuristic computations, Lazy A^* (or LA^*, for short) evaluates the heuristics one at a time, lazily. When a node n is generated, LA^* only computes one heuristic, $h_1(n)$, and adds n to Open . Only when n re-emerges as the top of Open is another heuristic, $h_2(n)$, evaluated; if this results in an increased heuristic estimate, n is re-inserted into Open . LA^* is as informative as A^*_{MAX}, but can significantly reduce search time, as we will not need to compute h_2 for many nodes.

LA^* reduces the search time, while maintaining the informativeness of A^*_{MAX}. However, if the goal is reducing search time, it is sometimes better to compute a fast heuristic on several nodes, rather than to compute a slow but informative heuristic on only one node. We combine the ideas of lazy heuristic evaluation and of trading off more node expansions for less heuristic computation time, into a *new* variant of LA^* called *rational lazy* A^* (RLA^*). RLA^* is based on rational metareasoning, and uses a myopic *value-of-information* criterion to decide whether to compute $h_2(n)$ or to bypass the computation of h_2 and expand n immediately when n re-emerges from Open . RLA^* aims to reduce search time, even at the expense of more node expansions than A^*_{MAX}.

5.4.2 Background and Related Work

The A^* algorithm [HNR68] is a best-first heuristic search algorithm guided by the cost function $f(n) = g(n) + h(n)$. If the heuristic $h(n)$ is admissible (never overestimates the real cost to the goal) then the set of nodes expanded by A^* is both necessary and sufficient to find the optimal path to the goal [DP85].

The idea behind the LA^* algorithm was briefly mentioned by [ZB12] in the context of the

MAXSAT heuristic for planning domains. [**?**] described *deferred evaluation* of heuristics in the context of the Fast Downward Planning System. In deferred evaluation, the successors of node n are placed in the open list with the heuristic estimate of n and heuristically evaluated only when expanded. The technique bears a similarity to a special case of LA^* where the first heuristic always returns 0.

Based on the idea of of trading off more node expansions for less heuristic computation time, [DKM12] formulated *selective max* (Sel-MAX), an online learning scheme which chooses one heuristic to compute at each state. Sel-MAX chooses to compute the more expensive heuristic h_2 for node n when its classifier predicts that $h_2(n) - h_1(n)$ is greater than some threshold, which is a function of heuristic computation times and the average branching factor.

5.4.3 Lazy A^*

Throughout this study we assume for clarity that we have two available admissible heuristics, h_1 and h_2. Extension to multiple heuristics is straightforward, at least for LA^*. Unless stated otherwise, we assume that h_1 is faster to compute than h_2 but that h_2 is *weakly more informed*, i.e., $h_1(n) \leq h_2(n)$ for the majority of the nodes n, although counter cases where $h_1(n) > h_2(n)$ are possible. We say that h_2 *dominates* h_1, if such counter cases do not exist and $h_2(n) \geq h_1(n)$ for *all* nodes n. We use $f_1(n)$ to denote $g(n) + h_1(n)$. Likewise, $f_2(n)$ denotes $g(n) + h_2(n)$, and $f_{max}(n)$ denotes $g(n) + \max(h_1(n), h_2(n))$. We denote the cost of the optimal solution by C^*. Additionally, we denote the computation time of h_1 and of h_2 by t_1 and t_2, respectively and denote the overhead of an *insert/pop* operation in OPEN by t_o. Unless stated otherwise we assume that t_2 is much greater than $t_1 + t_o$. LA^* thus mainly aims to reduce computations of h_2.

The pseudo-code for LA^* is depicted as Algorithm 8, and is very similar to A^*. In fact, without lines $7 - 10$, LA^* would be identical to A^* using the h_1 heuristic. When a node n is generated we only compute $h_1(n)$ and n is added to OPEN (Lines $11 - 13$), without computing $h_2(n)$ yet. When n is first removed from OPEN (Lines $7 - 10$), we compute $h_2(n)$ and reinsert it into OPEN , this time with $f_{max}(n)$.

It is easy to see that LA^* is as informative as A^*_{MAX}, in the sense that both A^*_{MAX} and LA^*expand a node n only if $f_{max}(n)$ is the best f-value in OPEN . Therefore, LA^* and A^*_{MAX} generate and expand and the same set of nodes, up to differences caused by tie-breaking.

Algorithm 8 Lazy A^*

```
procedure LAZY-$A^*$
    Apply all heuristics to Start
    Insert Start into Open
    while Open  not empty do
        $n \leftarrow$ best node from Open
        if Goal(n) then return Trace(n)
        end if
        if $h_2$ was not applied to $n$ then
            Apply $h_2$ to $n$ State Insert $n$ into Open
            continue                                        ▷ next node in OPEN
        end if
        for all child $c$ of $n$ do
            Apply $h_1$ to $c$
            Insert $c$ into Open
        end for
        Insert $n$ into Closed
    end whilereturn FAILURE
end procedure
```

In its general form A^* generates many nodes that it does not expand. These nodes, called *surplus* nodes [FGS+12], are in Open when we expand the goal node with $f = C^*$. All nodes in Open with $f > C^*$ are surely surplus but some nodes with $f = C^*$ may also be surplus. The number of surplus nodes in OPEN can grow exponentially in the size of the domain, resulting in significant costs.

LA^* avoids h_2 computations for many of these surplus nodes. Consider a node n that is generated with $f_1(n) > C^*$. This node is inserted into Open but will never reach the top of Open , as the goal node will be found with $f = C^*$. In fact, if Open breaks ties in favor of small h-values, the goal node with $f = C^*$ will be expanded as soon as it is generated and such savings of h_2 will be obtained for some nodes with $f_1 = C^*$ too. We refer to such nodes where we saved the computation of h_2 as *good* nodes. Other nodes, those with $f_1(n) < C^*$ (and some with $f_1(n) = C^*$) are called *regular nodes* as we apply both heuristics for them.

5.4.4 Rational Lazy A*

Using the principles of rational metareasoning (Section 2.1), theoretically every computational operator (heuristic function evaluation, node expansion, open list operation) should be treated as an action in a sequential decision-making meta-level problem, and actions should be chosen so as to achieve the minimal expected search time. However, the appropriate general metareasoning problem is extremely hard to define precisely, and even harder to solve optimally.

Therefore, we focus here on just one decision type, to be made in the context of LA^*, when n re-emerges from OPEN (Line 8). We have two options: **(1)** Evaluate the second heuristic $h_2(n)$ and add the node back to OPEN (Lines 8-10) like LA^*, or **2** bypass the computation of $h_2(n)$ and expand n right way (Lines 11 -13), thereby saving timeby not computing h_2, at the risk of additional expansions and evaluations of h_1. An method which attempts to optimally manage this tradeoff, which we call *Rational Lazy A**, is presented next. In order to choose rationally we define a criterion based on value of information (VOI) of evaluating $h_2(n)$ in this context.

The only addition of RLA* to LA* is the option to bypass h_2 computations (Lines 8-10). Suppose that we choose to compute h_2 — this results in one of the following outcomes:

1. n is eventually expanded.
2. n is re-inserted into OPEN , and the goal is found without ever expanding n.

Observe that computing h_2 can be beneficial only in outcome 2, where potential time savings are due to pruning a search subtree at the expense of the time $t_2(n)$. However, whether outcome 2 takes place after a given state is not known to the algorithm until the goal is found, and the algorithm must decide whether to evaluate h_2 according to what it *believes to be* the probability of each of the outcomes. We derive a *rational policy* for when to evaluate h_2, under the myopic assumption that the algorithm continues to behave like LA^* afterwards (i.e., it will never again consider bypassing the computation of h_2).

The time wasted by being sub-optimal in deciding whether to evaluate h_2 is called the *regret* of the decision. If $h_2(n)$ is not helpful and we decide to compute it, the effort for evaluating $h_2(n)$ turns out to be wasted. On the other hand, if $h_2(n)$ is helpful but we decide to bypass it, we needlessly expand n. Due to the myopic assumption, RLA^* would evaluate both h_1 and h_2 for all successors of n.

	Compute h_2	Bypass h_2
h_2 helpful	0	$t_e + (b(n) - 1)t_d$
h_2 not helpful	t_d	0

Table 5.3: Time losses in Rational Lazy A*

Table 5.3 summarizes the regret of each possible decision, for each possible future outcome;

each column in the table represents a decision, while each row represents a future outcome. In the table, t_d is the to time compute h_2 and re-insert n into OPEN thus delaying the expansion of n, t_e is the time to remove n from OPEN , expand n, evaluate h_1 on each of the $b(n)$ ("local branching factor") children $\{n'\}$ of n, and insert $\{n'\}$ into the open list. Computing h_2 needless "wastes" t_d time. Bypassing h_2 computation when h_2 would have been helpful "wastes" $t_e + b(n)t_d$ time, but because computing h_2 would have cost t_d, the regret is $t_e + (b(n) - 1)t_d$.

Let us denote the probability that h_2 is helpful by p_h. The expected regret of computing h_2 is thus $(1 - p_h)t_d$. On other hand, the expected regret of bypassing h_2 is $p_h(t_e + (b(n) - 1)t_d)$. As we wish to minimize the expected regret, we should thus evaluate h_2 just when:

$$(1 - p_h)t_d < p_h(t_e + (b(n) - 1)t_d) \tag{5.29}$$

or equivalently:

$$(1 - b(n)p_h)t_d < p_h t_e \tag{5.30}$$

If $p_h b(n) \geq 1$, then the expected regret is minimized by always evaluating h_2, regardless of the values of t_d and t_e. In these cases, RLA* cannot be expected to do better than LA^*. For example, in the 15-puzzle and its variants, the effective branching factor is ≈ 2. Therefore, if h_2 is expected to be helpful for more than half of the nodes n on which LA^* evaluates $h_2(n)$, then one should simple use LA^*.

For $p_h b(n) < 1$, the decision of whether to evaluate h_2 depends on the values of t_d and t_e:

$$\textbf{evaluate } h_2 \textbf{ if } t_d < \frac{p_h}{1 - p_h b(n)} t_e \tag{5.31}$$

Let us analyze t_d and t_e. Denote by t_c the time to generate the children of n. Then we have:

$$\begin{aligned} t_d &= t_2 + t_o \\ t_e &= t_o + t_c + b(n)t_1 + b(n)t_o \end{aligned} \tag{5.32}$$

By substituting (5.32) into (5.31), obtain: **evaluate** h_2 **if**:

$$t_2 + t_o < \frac{p_h\left[t_c + b(n)t_1 + (b(n) + 1)t_o\right]}{1 - p_h b(n)} \tag{5.33}$$

The factor $\frac{p_h}{1-p_h b(n)}$ depends on the potentially unknown probability p_h, making it difficult to reach the optimum decision. However, if our goal is just to do better than LA^*, then it is safe to replace p_h by an upper bound on p_h.

We now turn to implementation-specific estimation of the respective runtimes. Open in A^* is frequently implemented as a priority queue, and thus we have, approximately, $t_o = \tau \log N_o$ for some τ, when the size of Open is N_o. Evaluating h_1 is cheap in many domains, as is the case with MD in discrete domains, t_o is the most significant part of t_e. In such cases, rule (5.33) can be approximated as (5.34):

$$\textbf{evaluate } h_2 \textbf{ if } t_2 < \frac{\tau p_h}{1 - p_h b(n)}(b(n)+1)\log N_o \tag{5.34}$$

Rule (5.34) recommends to evaluate h_2 mostly at late stages of the search, when the open list is large, and in nodes with a higher branching factor.

In other domains, such as planning, both t_1 and t_2 are significantly greater than both t_o and t_c, and terms not involving t_1 or t_2 can be dropped from (5.33), resulting in Rule (5.35):

$$\textbf{evaluate } h_2 \textbf{ if } \frac{t_2}{t_1} < \frac{p_h b(n)}{1 - p_h b(n)} \tag{5.35}$$

The right side of (5.35) grows with $b(n)$, and here it is beneficial to evaluate h_2 only for nodes with a sufficiently large branching factor. On rearranging equation 5.35, we get the criterion which we actually use for planning domains, which is to evaluate h_2 only when:

$$b(n) > \frac{t_2}{t_1 p_h \left(1 + \frac{t_2}{t_1}\right)} \tag{5.36}$$

5.4.5 Empirical Evaluation

In the uniform-cost 15 puzzle, the open list contains only a few different f-costs, and is frequently implemented using buckets, violating the assumption of logarithmic time for which RLA* is beneficial. In order to better evaluate the latter, we therefore use the weighted 15-puzzle variant, where the cost of moving each tile is equal to the number on the tile. For consistency of comparison, we used a subset of 36 problem instances out of the set of 100 15-puzzle instances by [Kor85], keeping the problems which could be solved with 2Gb of RAM and 15 minutes

	A^*		LA^*			RLA^*				
lookahead	generated	time	Good1	N_2	time	generated	Good1	Good2	N_2	time
6	889,930	**0.601**	257,598	632,332	0.462	944,750	299,479	239,320	405,951	0.446
8	740,513	0.700	197,107	543,406	**0.431**	892,216	233,370	303,655	260,823	0.402
10	612,010	0.929	145,687	466,327	0.474	859,220	278,431	445,846	134,943	**0.378**
12	454,171	1.128	95,118	359,053	0.621	807,846	277,783	428,686	101,377	0.465

Table 5.4: Weighted 15 puzzle: comparison of $A^*_{\max}$, Lazy A^*, and Rational Lazy A^*

timeout using the Weighted Manhattan heuristic (WMD) for h_1. As the second, expensive and informative, h_2 heuristic for LA^* and RLA^*, we a heuristic based on lookaheads [SKFH10]. Given a bound d we applied a bounded depth-first search from a node n and backtracked when we reached leaf nodes l for which $g(l) + WMD(l) > g(n) + WMD(n) + d$. f-values from leaves were propagated to n.

Table 5.4 presents the results averaged on all instances solved. The running times are reported relative to the time of $A^*$with WMD (with no lookahead), which generated 2012643 nodes (not reported in the table). The first 3 columns of Table 5.4 shows the results for A^* with the lookahead heuristic for different lookahead depths. The best time is achieved for lookahead 6, (0.601 compared to A^* with WMD). The fact that the time is not continuing to decrease with deeper lookaheads is clearly due to the fact that although the resulting heuristic improves as a function of lookahead depth (expanding and generating fewer nodes), the increasing overheads of computing the heuristic eventually outweights the computation time it saved by expanding fewer nodes.

The next 3 columns show the results for LA^* with WMD as h_1, lookahead as h_2, for different lookahead depths (LA^* generates the same number of nodes as A^*). The *Good1* column presents the number of nodes where LA^*saved the computation of h_2 while the N_2 column presents the number of nodes where h_2 was computed. $\approx 28\%$ of nodes were *Good1* and since t_2 was the most dominant time cost, most of this saving is reflected in the timing results. The best results are achieved for lookahead 8, with a runtime of 0.431 compared to A^* with WMD.

The final columns show the results of RLA* with different lookaheads for empirically best values of τ. The *Good2* column counts the number of times that RLA* decided to bypass the h_2 computation and expand the node right away.Observe that RLA* outperforms LA^*, which in turn outperforms standard A*, for all lookahead depths. The lowest time with RLA* (0.378 of $A^*$with WMD) and empirically best τ was obtained for lookahead 10. That is achieved because the more expensive h_2 heuristic is computed less often, reducing its effective computational overhead, with little or no adverse effect in the number of expanded nodes. Although LA* expanded fewer nodes, RLA* performed much fewer h_2 computations as can be seen in the table, resulting in decreased overall runtimes for all lookahead depths.

[TBS$^+$13] presents results for comparing $RLA^*$and other algorithms on 42 planning domains. In these domains, time spent evaluating heuristics constitutes approximately 90% of the total search time, hence applying rule (5.36) is justified. RLA^* was compared to LA^* and to A^* using each of the heuristics individually, as well as to the max-based combination of the heuristics, and to the combination using selective-max (Sel-MAX) [DKM12]. In the evaluation, RLA^* solved most problem instances in the shortest total search time.

The case of both t_1 and t_2 being significantly larger than t_o and t_c can also be modeled using the 15-puzzle domain by considering only the time spent evaluating the heuristics. Table 5.5 presents the results of solving the Weighted 15-puzzle obtained on the the set of 100 15-puzzle instances by [Kor85], using the combination of weighted Manhattan distance and weigthed linear conflict [HMY92] heuristics. Rows N_1 and N_2 present the average numbers of evaluations of h_1 and h_2, rows T_1 and T_2 the average total times (per instance) spent evaluating each of the heuristics. A^*_{MAX} evaluates both h_1 and h_2 on both nodes and is obviously the slowest.

	A^*_{MAX}	LA^*	RLA^*
N_1	1,482,271	1,482,271	1,521,163
T_1, sec	0.979	0.979	1.026
N_2	1,482,271	1,117,826	868,182
T_2, sec	4.122	2.425	1.938
$T_1 + T_2, sec$	5.101	3.404	2.964

Table 5.5: Weighted 15 puzzle: comparison of heuristic computation times in $A^*_{\max}$, Lazy A^*, and Rational Lazy A^*

LA^* evaluates h_2 selectively, spending 40% less time evaluating h_2 and 33% less time overall. RLA^*, using Rule (5.36), spends 20% less time than LA^* evaluating h_2 *at the expense* of spending 4% more time evaluating h_1, and takes 12% less time than LA^* and 40% less time than A^*_{MAX} overall.

5.4.6 Conclusion and Further Research

We discussed two schemes for decreasing heuristic evaluation times. LA^* is very simple to implement and is as informative as A^*_{MAX}. LA^* can significantly speed up the search, especially if t_2 dominates the other time costs, as seen in weighted 15 puzzle and planning domains. Rational LA^* allows additional cuts in h_2 evaluations, at the expense of being less informed than A^*_{MAX}. However, due to a rational tradeoff, this allows for an additional speedup, and Rational LA^* achieves the best overall performance in our domains.

In particular, RLA^* is simpler to implement than its direct competitor, selective max, but its decision can be more informed. When RLA^* has to decide whether to compute h_2 for some node n, it already *knows* that $f_1(n) \leq C^*$. By contrast, although selective max uses a much more complicated decision rule, it chooses which heuristic to compute when n is first generated, and does not know whether h_1 will be informative enough to prune n. Rational LA^* outperforms selective max in our planning experiments.

There are numerous other ways to use rational metareasoning to improve A^*, starting from generalizing RLA^* to handle more than two heuristics, to using the meta-level to control decisions in other variants of A^*. All these potential extensions provide fruitful ground for further research.

Chapter 6

Summary and Contribution

The thesis comprises two major topics:

1. Rational computation of value of information (Chapter 4).

2. Case studies of application of rational metareasoning to selection and application of heuristic computations (Chapter 5).

The first topic addressed cases in which estimating the value of information of a computational action is expensive, as well as cases of too many computational actions, such that although estimating the value of information of a single action is cheap, estimating the value of information of all actions at each step of the algorithm is expensive. The research, published in [TS12b], resulted in an improvement to a widely used class of VOI-based optimization algorithms that allows to decrease the computation time while only slightly affecting the reward. Theoretical analysis of the proposed approach to rational computation of the value of information was supported by empirical evaluation of several combinations of algorithms and search problems.

The second topic considered several search problems and improved some well-known algorithms for solving the problems using the rational metareasoning approach:

- Adaptive deployment of value-ordering heuristics in constraint satisfaction problems (Section 5.2) [TS11];

- Monte Carlo tree search based on simple regret (Section 5.3) [TS12a, HRTS12];

- Decreasing heuristic evaluation time in a variant of A* (Section 5.4) [TBS+13].

In addition to the algorithm improvements, the studies demonstrated a number of common rational metareasoning techniques which can be extended to other problem types. In particular,

- Section 5.3, "VOI-aware Monte-Carlo tree search" provided distribution-independent upper bounds for semi-myopic VOI estimates in Monte-Carlo sampling.
- Section 5.4, "Towards rational deployment of Multiple Heuristics in A*", introduced a novel area of application of rational metareasoning—optimal search in optimization problems.

As a whole, the research advanced the use of rational metareasoning in problem-solving search algorithms. Applications of rational metareasoning in the case studies serve as examples to help researchers employ the methodology in solutions for other problems. Advances in rational computation and estimation of VOI increase performance and applicability of existing and new search algorithms and alleviate dependence of algorithm performance on manual fine-tuning.

Bibliography

[ABF^{+}06] Carlos Ansótegui, Ramón Béjar, César Fernàndez, Carla Gomes, and Carles Mateu. The impact of balancing on problem hardness in a highly structured domain. In *Proc. of 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 06)*, 2006.

[ACBF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the Multiarmed bandit problem. *Mach. Learn.*, 47:235–256, May 2002.

[Ack87] David H. Ackley. *A connectionist machine for genetic hillclimbing.* Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[AM96] John A. Allen and Steven Minton. Selecting the right heuristic algorithm: Runtime performance predictors. In *Advances in Artificial The Eleventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 41–53. Springer-Verlag, 1996.

[BG07] Mustafa Bilgic and Lise Getoor. VOILA: efficient feature-value acquisition for classification. In *AAAI*, pages 1225–1230. AAAI Press, 2007.

[BHL05] Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. Description and representation of the problems selected for the first international constraint satisfaction solver competition. Technical report, Proc. of CPAI'05 workshop, 2005.

[BLG11] Petr Braudiš and Jean Loup Gailly. Pachi: State of the art open source Go program. In *ACG 13*, 2011.

[BLS$^+$08] Vadim Bulitko, Mitja Luštrek, Jonathan Schaeffer, Yngvi Björnsson, and Sverrir Sigmundarson. Dynamic control in real-time heuristic search. *J. Artif. Int. Res.*, 32(1):419–452, 2008.

[BMS11] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in finitely-armed and continuous-armed bandits. *Theor. Comput. Sci.*, 412(19):1832–1852, 2011.

[CBSS08] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-Carlo tree search: A new framework for game AI. In *AIIDE*, 2008.

[DKM10] Carmel Domshlak, Erez Karpas, and Shaul Markovitch. To max or not to max: Online learning for speeding up optimal planning. In *AAAI Conference on Artificial Intelligence*, 2010.

[DKM12] Carmel Domshlak, Erez Karpas, and Shaul Markovitch. Online speedup learning for optimal planning. *JAIR*, 44:709–755, 2012.

[DP85] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

[DP87] Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artif. Intell.*, 34:1–38, December 1987.

[EKH10] Patrick Eyerich, Thomas Keller, and Malte Helmert. High-quality policies for the canadian travelers problem. In *In Proc. AAAI 2010*, pages 51–58, 2010.

[ES08] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing (Natural Computing Series).* Springer, October 2008.

[FGS$^+$12] A. Felner, M. Goldenberg, G. Sharon, R. Stern, T. Beja, N. R. Sturtevant, J. Schaeffer, and Holte R. Partial-expansion a* with selective node generation. In *AAAI*, pages 471–477, 2012.

[Gee92] Pieter Andreas Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proc. 10th European Conf. on AI*, ECAI '92, pages 31–35, New York, NY, USA, 1992. John Wiley & Sons, Inc.

[GS01] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, 126(1–2):43–62, 2001.

[GW06] Sylvain Gelly and Yizao Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. *Computer*, 2006.

[HCL10] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. Technical report, National Taiwan University, Taipei, Taiwan, 2010.

[HH00] Michael C. Horsch and William S. Havens. Probabilistic arc consistency: A connection between constraint reasoning and probabilistic reasoning. In *UAI*, pages 282–290, 2000.

[HHM93] D. Heckerman, E. Horvitz, and B. Middleton. An approximate nonmyopic computation for value of information. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(3):292–298, 1993.

[HK95] Eric J. Horvitz and Adrian Klein. Reasoning, metareasoning, and mathematical truth: Studies of theorem proving under limited resources. In *Proceedings of UAI-95*, pages 306–314. Morgan Kaufmann, 1995.

[HMY92] Othar Hansson, Andrew Mayer, and Moti Yung. Criticizing solutions to relaxed models yields powerful admissible heuristics. *Inf. Sci.*, 63(3):207–227, 1992.

[HNR68] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SCC-4(2):100–107, 1968.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):pp. 13–30, 1963.

[Hor87] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, pages 429–444, 1987.

[HRG$^+$01] Eric J. Horvitz, Yongshao Ruan, Carla Gomes, Henry Kautz, Bart Selman, and Max Chickering. A bayesian approach to tackling hard computational problems. In *Proceedings of UAI-01*, pages 235–244. Morgan Kaufmann, 2001.

[HRTS12] Nicholas Hay, Stuart J. Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: Theory and applications. In *UAI*, pages 346–355, 2012.

[KDG04] Kalev Kask, Rina Dechter, and Vibhav Gogate. Counting-based look-ahead schemes for constraint satisfaction. In *Proc. of 10th Int. Conf. on Constraint Programming (CP04)*, pages 317–331, 2004.

[KG07] Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, pages 1650–1654, 2007.

[KLG$^+$08] Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne VanBriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, November 2008.

[Knu74] Donald E. Knuth. Estimating the efficiency of backtrack programs. Technical report, Stanford University, Stanford, CA, USA, 1974.

[Kor85] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[Kor90] Richard E. Korf. Real-time heuristic search. *Artif. Intell.*, 42(2-3):189–211, 1990.

[KS06] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *ECML*, pages 282–293, 2006.

[MP09] Andreas Maurer and Massimiliano Pontil. Empirical Bernstein bounds and sample-variance penalization. In *COLT*, 2009.

[MS79] Clyde L. Monma and Jeffrey B. Sidney. Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research*, 4(3):215–224, August 1979.

[MSS97] Amnon Meisels, Solomon Eyal Shimony, and Gadi Solotorevsky. Bayes networks for estimating the number of solutions to a CSP. In *Proc. of the 14th National Conference on AI*, pages 179–184, 1997.

[Ref04] Philippe Refalo. Impact-based search strategies for constraint programming. In *CP*, pages 557–571. Springer, 2004.

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 2003.

[RS08] Yan Radovilsky and Solomon Eyal Shimony. Observation subset selection as local compilation of performance profiles. In *UAI*, pages 460–467, 2008.

[RW89] Stuart J. Russell and Eric Wefald. On optimal game-tree search using rational meta-reasoning. In *Proc. of IJCAI*, pages 334–340, 1989.

[RW91] Stuart Russell and Eric Wefald. *Do the right thing: studies in limited rationality.* MIT Press, Cambridge, MA, USA, 1991.

[SF97] Daniel Sabin and Eugene C. Freuder. Understanding and improving the MAC algorithm. In *3rd Int. Conf. on Principles and Practice of Constraint Programming, LNCS 1330*, pages 167–181. Springer, 1997.

[SKFH10] Roni Stern, Tamar Kulberis, Ariel Felner, and Robert Holte. Using lookaheads with optimal best-first search. In *AAAI*, pages 185–190, 2010.

[TBS$^+$13] David Tolpin, Tal Beja, Solomon Eyal Shimony, Ariel Felner, and Erez Karpas. Towards rational deployment of multiple heuristics in a*. In *IJCAI*, 2013. To appear.

[TS11] David Tolpin and Solomon Eyal Shimony. Rational deployment of CSP heuristics. In *IJCAI*, pages 680–686, 2011.

[TS12a] David Tolpin and Solomon Eyal Shimony. MCTS based on simple regret. In *AAAI*, pages 570–576. AAAI Press, 2012.

[TS12b] David Tolpin and Solomon Eyal Shimony. Rational value of information estimation for measurement selection. *Intelligent Decision Technologies*, 6(4):297–304, 2012.

[TS12c] David Tolpin and Solomon Eyal Shimony. Semimyopic measurement selection for optimization under uncertainty. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 42(2):565–579, 2012.

[Tsa93] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London and San Diego, 1993.

[VM05] Joannès Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *ECML*, pages 437–448, 2005.

[WF92] Richard J. Wallace and Eugene C. Freuder. Ordering heuristics for arc consistency algorithms. In *AI/GI/VI 92*, pages 163–169, 1992.

[Wol04] Armin Wolf. Reduce-to-the-opt a specialized search algorithm for contiguous task scheduling. In Krzysztof R. Apt, Franois Fages, Francesca Rossi, Pter Szeredi, and Josef Vncza, editors, *Recent Advances in Constraints*, volume 3010 of *Lecture Notes in Computer Science*, pages 223–232. Springer Berlin / Heidelberg, 2004.

[WP09] Gerhard Wickler and Stephen Potter. Information-gathering: From sensor data to decision support in three simple steps. *Intelligent Decision Technologies*, 3(1):3–17, 2009.

[XL00] Ke Xu and Wei Li. Exact phase transitions in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 12:93–103, 2000.

[ZB12] Lei Zhang and Fahiem Bacchus. Maxsat heuristics for cost optimal planning. In *AAAI*, 2012.

[Zil93] Shlomo Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms.* PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 1993.

[ZRB05] Alice X. Zheng, Irina Rish, and Alina Beygelzimer. Efficient test selection in active diagnosis via entropy approximation. In *UAI*, pages 675–682, 2005.