

# Introduction to AI

## Intro to Intro

Erez Karpas  
Slides by Carmel Domshlak

IE&M — Technion

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# People

## Official

- lecturer: Dr. Erez Karpas
- teaching assistants: Sarah Keren, Alexander Tuisov
- web site: moodle
- grading policy: programming project + exam
- prerequisites: data structures and algorithms + ...

## Teaching materials

- no precise textbook, but “AI: Modern Approach” by Russell & Norvig is very good
- slides are available on the web (the day before the lecture)
- additional resources: **ask us** per topic

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# What is AI?

Answer Here

<http://bit.ly/2eTNbTG>

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# What is AI?

Answer Here

<http://bit.ly/2eTNbTG>

Artificial intelligence is (an historic name for)

- Choosing the right things to do algorithmically
  - Doing so everywhere, with a finite set of algorithms

לפניהם נתקל בדרכם כר בהז'ן; כל אחד יתנו לנו סדרה של

## Introduction to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

### NP-hardness

## AI Agents as Problem Solvers

## Course objectives

# Why AI in IE?

Introduction  
to AI

Artificial intelligence is (an historic name for)

- Choosing the right things to do algorithmically
- Doing so everywhere, with a finite set of algorithms

Ultimate product:

- Autonomous operation for long periods of time  
(low-cost control)
- Reasonable efficiency and correctness
- Operation in human-populated environments
- Low-cost and rapid development
- Operational flexibility (in service and manufacturing)

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# Why AI in IE?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

## Ultimate product:

- Autonomous operation for long periods of time (low-cost control)
- Reasonable efficiency and correctness
- Operation in human-populated environments
- Low-cost and rapid development
- Operational flexibility (in service and manufacturing)



# Why AI in IE?

## Ultimate product:

- Autonomous operation for long periods of time  
(low-cost control)
- Reasonable efficiency and correctness
- Operation in human-populated environments
- **Low-cost and rapid development**
- **Operational flexibility** (in service and manufacturing)

## Think of **customized manufacturing**

- Products = composites of subsets of 100s basic parts
- The list of products is (unbelievably) huge
- Parts are stored and should be collected
- The same robot should ultimately work in different industries

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# Why AI in IE?

## Ultimate product:

- Autonomous operation for long periods of time  
(low-cost control)
- Reasonable efficiency and correctness
- Operation in human-populated environments
- Low-cost and rapid development
- Operational flexibility (in service and manufacturing)

## Think of customized manufacturing

- Products = composites of subsets of 100s basic parts
- The list of products is (unbelievably) huge
- Parts are stored and should be collected
- The same robot should ultimately work in different industries

Do we have it? Not entirely, but we are getting closer

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# Do we have it?

Not entirely, but we are getting closer



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

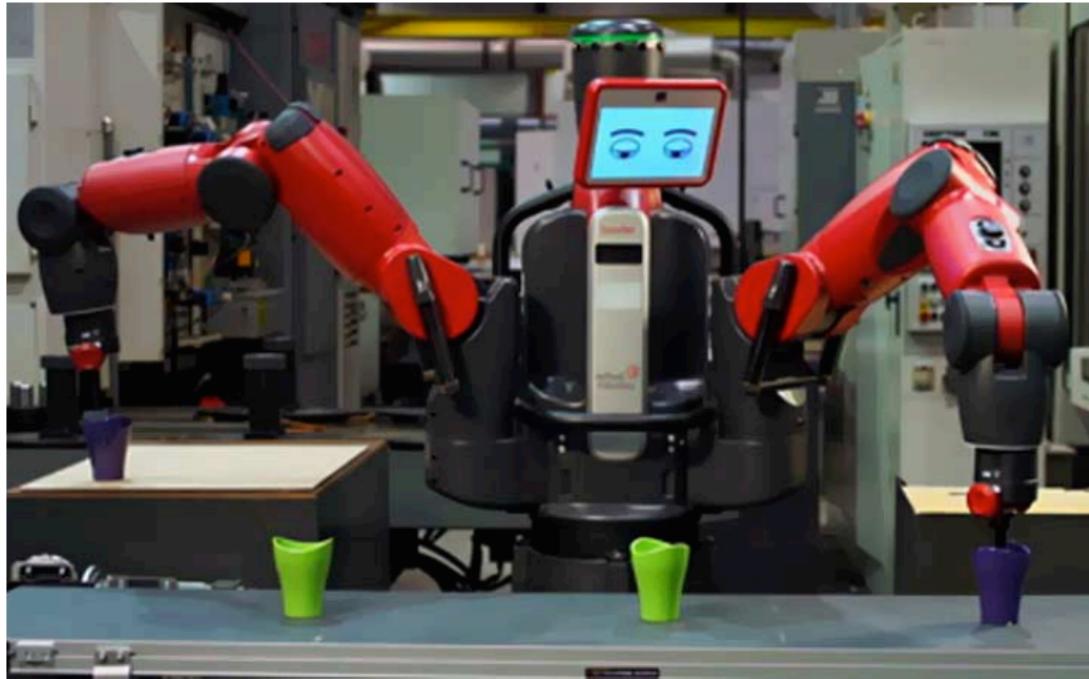
NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

Do we have it?

Not entirely, but we are getting closer



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# AI Technology Components

Note: AI is not neuroscience. (Aeronautics is not ornithology).

What it takes to build a golem? (וילא)

- knowledge representation to store what is known
- planning to decide about a course of action
- automated reasoning to use the stored information to answer questions and to draw new conclusions
- machine learning to adapt to new circumstances and to detect and extrapolate patterns
- natural language processing to communicate in English/Hebrew/...
- vision (understanding) to perceive objects
- robotics to manipulate objects and move about

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# Three Approaches to Problem Solving

- ❶ **programming:** specify control by hand
- ❷ **learning:** learn control from experience
- ❸ **model-oriented solving:**  
specify problem by hand, derive control automatically
- All three have strengths and weaknesses; approaches not exclusive and often complementary.
- All three will be gradually discussed in the course

↙ ראה קב' נאיה נאיה נאיה

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# Model-oriented problem solving

From programming to model-oriented solving

Problem  $\Rightarrow$  Language  $\Rightarrow$  Solver  $\Rightarrow$  Solution

- ➊ models for defining, classifying, and understanding problems

- what is a *problem (type)*
- what is a *solution*, and
- what is a *good/optimal solution*

הה זה נמיין מהויה?

הה זה קניין מהויה?

(בגרת פוליאור, איך כהן?)

מיין נמיין זינדר?

- ➋ languages for representing problems

- ➌ algorithm per language for solving ALL problems expressible in that language

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# Model-oriented problem solving

From programming to model-oriented solving

Problem  $\Rightarrow$  Language  $\Rightarrow$  Solver  $\Rightarrow$  Solution

- ➊ models for defining, classifying, and understanding problems
  - what is a *problem (type)*
  - what is a *solution*, and
  - what is a *good/optimal solution*
- ➋ languages for representing problems
- ➌ algorithm per language for solving ALL problems expressible in that language

## Questions

- ➊ How many models will do?
- ➋ If just a few (?!), what kind of algorithms these will be?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# A VERY brief guide to NP-hardness

Imagine that ...

You just got a new job. Congratulations!

You are asked to develop an **efficient algorithm** for determining whether or not a given set of specifications for a new XXX component can be met, and if so, constructing a design that meets them

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# A VERY brief guide to NP-hardness

Imagine that ...

You just got a new job. Congratulations!

You are asked to develop an **efficient algorithm** for determining whether or not a given set of specifications for a new XXX component can be met, and if so, constructing a design that meets them

What is efficient:  $O(\text{poly}(n))$  vs.  $O(\text{exp}(n))$

	$n = 10$	20	30	40	50
$n$	.00001 sec	.00002 sec	.00003 sec	.00004 sec	.00005 sec
$n^2$	.0001 sec	.0004 sec	.0009 sec	.0016 sec	.0025 sec
$n^3$	.001 sec	.008 sec	.027 sec	.064 sec	.125 sec
$n^5$	.1 sec	3.2 sec	24.3 sec	1.7 min	5.2 min
$2^n$	.001 sec	1.0 sec	17.9 min	12.7 days	35.7 years
$3^n$	.059 sec	58 min	6.5 years	3855 cent	$2 \times 10^8$ cent

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# A VERY brief guide to NP-hardness

Imagine that ...

You just got a new job. Congratulations!

You are asked to develop an **efficient algorithm** for determining whether or not a given set of specifications for a new XXX component can be met, and if so, constructing a design that meets them

Bad news

A year after you still have no algorithm that is substantially more efficient than **searching through all possible designs** ...

What to do?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# You don't want to ...



I can't find an efficient algorithm, I guess I'm just too dumb.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

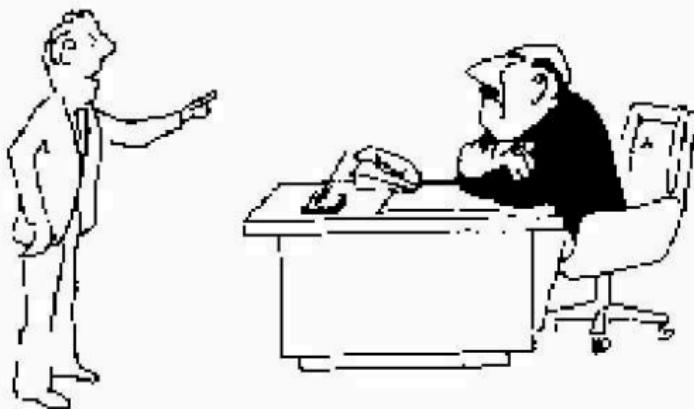
AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# You would like to say, but ...



I can't find an efficient algorithm, because no such algorithm is possible

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# Today you can say



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

I can't find an efficient algorithm, but neither can all these famous people.

# Some important complexity classes

A class of problems is in ...

**P** if any problem in the class can be solved in polynomial time

**NP** if, for any problem in the class, some solutions can be verified in polynomial time

Obviously,  $P \subseteq NP$ , and it is not likely that  $P=NP$ . However, **no proof** so far for  $P \neq NP$ !

מקרה ננתקות נוחות Pic נימן גזע כהן פ. נטול נטול.

## Introduction to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

### NP-hardness

## AI Agents as Problem Solvers

## Course objectives

# Some important complexity classes

A class of problems is in ...

- P** if any problem in the class can be **solved** in polynomial time
- NP** if, for any problem in the class, some solutions can be **verified** in polynomial time
- NPC** if (informally) it is one of the “hardest” problem classes in NP

Obviously,  $P \subseteq NP$ , and it is not likely that  $P=NP$ . However, **no proof** so far for  $P \neq NP$ !

• נספחים נאיהן “הכי קשוחות” נ-NPC •

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# NP-complete problem classes

• נג"ט גוראה: נגיד כ' הטענה היא כז/כז

## Definition: NP-completeness

A decision (yes/no) problem class  $C$  is **NP-complete** if:

- ➊  $C$  is in NP, and
- ➋ Every problem class in NP is reducible to  $C$  in polynomial time.

## Definition: Reducibility

A problem class  $K$  is **reducible** to  $C$  if there is a **polynomial-time deterministic algorithm** (reduction) that transforms any problem  $k \in K$  into a problem  $c \in C$  such that the answer to  $c$  is Yes if and only if the answer to  $k$  is Yes.

( $\text{סוד}$ ) **הוכחה:** מונע מהשאלה  $K$  מתקבלת מ- $C$  בזמן פולינומי.



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# NP-complete problem classes

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS  
NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

## Definition: NP-completeness

A decision (yes/no) problem class  $C$  is **NP-complete** if:

- ①  $C$  is in NP, and
- ② Every problem class in NP is reducible to  $C$  in polynomial time.

- To prove that an NP problem class  $C$  is NP-complete it is sufficient to show that an **already known** NP-complete problem class  $K$  reduces to  $C$ .
- A problem satisfying condition (2) is said to be **NP-hard**, whether or not it satisfies condition (1).
- Bottom line: if we had a polynomial time algorithm for  $C$ , we could solve **all** problems in NP in polynomial time.

# Short Quiz

<http://bit.ly/2fi2vhs>

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# Still, what to do?

The needs to solve a problem won't disappear overnight simply because the problem is known to be **NP-hard**, but knowing the problem is NP-complete does **provide valuable information**

- The search for an efficient exact algorithm should certainly be accorded “low priority”

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# Dealing with NP-hard problems

- Less relevant to our course:
  - approximation algorithms
  - probabilistic algorithms
- More relevant to our course:
  - efficient algorithms for interesting subclasses (special cases) of the general problem
  - relaxing the problem so that a fast algorithm will meet most of the problem's original properties
  - algorithms that do not guarantee to run quickly, but seem likely to do it “most of the time”

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

# Model-oriented problem solving

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

From programming to model-oriented solving

Problem  $\Rightarrow$  Language  $\Rightarrow$  Solver  $\Rightarrow$  Solution

Modeling Time vs. Solution Time and Quality

זמן מודולר יחסית  
זמן פתרון אוניברסלי.  
ההכרה.

- specialized methods are typically more efficient  
(though even that is not necessarily correct),  
but tend to require lots of programming
- goal in AI problem solving is to facilitate modeling and yet provide efficient solutions  
      . פתרון מודולרי קייזר פט
- this involves general languages and thus language-specific algorithms

# Intelligent Agents

- ① Agents and environments
- ② Rationality
- ③ Performance measure,  
Environment, Actuators, Sensors
- ④ Environment types
- ⑤ Agent types

Agent Baxter ↴



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# Agents and environments

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

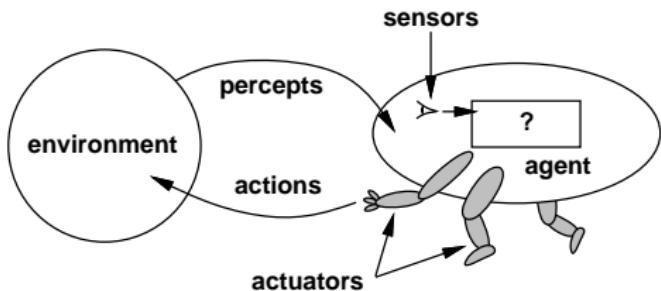
AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

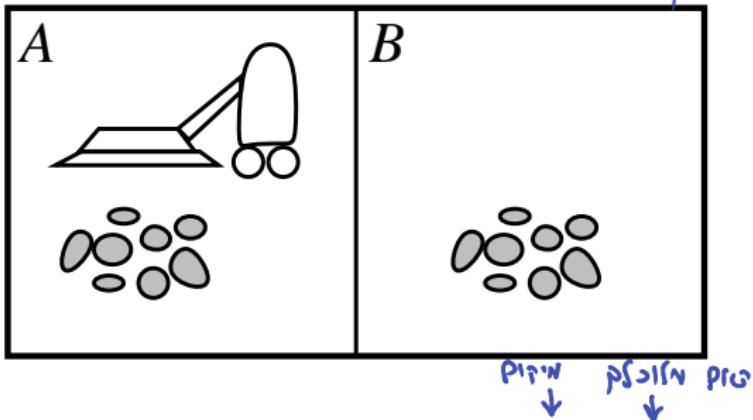
Course  
objectives



- **Agents** include humans, robots, softbots, thermostats, etc.
- The **agent function** maps from percept histories to actions:  
 $f : \mathcal{P}^* \rightarrow \mathcal{A}$   
הAgent מפה ההיסטוריה של ה-sensor לפעולות.  
הפונקציית agent מפה ההיסטוריה של ה-sensor לפעולות.
- The **agent program** runs on the physical **architecture** to produce  $f$ .  
ה-Agent רץ על ארכיטקטורת הגוף物理性的 architecture, ובודק אם מושג המטרה.

# Vacuum-cleaner world

פוך נעל



Percepts: location and contents, e.g., [A, Dirty]

Actions: Left, Right, Suck, NoOp

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# A vacuum-cleaner agent

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
:	:

Procedure **Reflex-Vacuum-Agent**: returns an action

```
def Reflex-Vacuum-Agent(location, status):  
    if status = Dirty then return Suck  
    else if location = A then return Right  
    else if location = B then return Left
```

What is the *right* function?

# Rationality

רין כהן ורין Agent-Function פיק'

- Fixed **performance measure** evaluates the **environment sequence**:
    - one point per square cleaned up in time  $T$ ?
    - one point per clean square per time step, minus one per move?
    - penalize for  $> k$  dirty squares?
  - A **rational agent** chooses whichever action maximizes the **expected** value of the performance measure given the percept sequence to date
  - Rational  $\neq$  knows everything
    - percepts may not supply all relevant information
  - Rational  $\neq$  predicts perfectly
    - action outcomes may not be as expected
- ~ Hence, rational  $\neq$  successful

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# Agent Classification

To design a rational agent, we must specify  
**task environment**

Example: Designing an automated taxi

## Performance measure??

- safety, destination, profits, legality, comfort, ...

## Environment??

- Israel streets/freeways, traffic, pedestrians, weather, ...

## Actuators??

- steering, accelerator, brake, horn, speaker/display, ...

## Sensors??

- video, accelerometers, gauges, engine sensors, keyboard, GPS, ...

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# Internet shopping agent

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

## Performance measure??

- price, quality, appropriateness, efficiency, ...

## Environment??

- current and future WWW sites, vendors, shippers, ...

## Actuators??

- display to user, follow URL, fill in form, ...

## Sensors??

- HTML pages (text, graphics, scripts), ...

# From Problems to Models

## Characterizing Properties of Our Problems

- Observable vs. Partially Observable vs. Unobservable
- Deterministic vs. Non-deterministic vs. Stochastic
- Static vs. Dynamic vs. Semidynamic
- Discrete vs. Continuous
- Single agent vs. Multi-agent

ר' יונתן  
ר' יונה  
ר' יונה  
ר' יונה

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# From Problems to Models

## Characterizing Properties of Our Problems

- Observable vs. Partially Observable vs. Unobservable
- Deterministic vs. Non-deterministic vs. Stochastic
- Static vs. Dynamic vs. Semidynamic
- Discrete vs. Continuous
- Single agent vs. Multi-agent

Try it

<http://bit.ly/2eIHzzo>

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# From Problems to Models

	Solitaire	Backgammon	Internet shopping	Taxi
<u>Observable??</u> <u>Deterministic??</u> <u>Static??</u> <u>Discrete??</u> <u>Single-agent??</u>				

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# From Problems to Models

	Solitaire	Backgammon	Internet shopping	Taxi
<u>Observable??</u>	Yes	Yes	No	No
<u>Deterministic??</u>				
<u>Static??</u>				
<u>Discrete??</u>				
<u>Single-agent??</u>				

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# From Problems to Models

	Solitaire	Backgammon	Internet shopping	Taxi
<u>Observable??</u>	Yes	Yes	No	No
<u>Deterministic??</u>	Yes	No	Partly	No
<u>Static??</u>				
<u>Discrete??</u>				
<u>Single-agent??</u>				

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# From Problems to Models

	Solitaire	Backgammon	Internet shopping	Taxi
<u>Observable??</u>	Yes	Yes	No	No
<u>Deterministic??</u>	Yes	No	Partly	No
<u>Static??</u>				
<u>Discrete??</u>				
<u>Single-agent??</u>				

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# From Problems to Models

	Solitaire	Backgammon	Internet shopping	Taxi
<u>Observable??</u>	Yes	Yes	No	No
<u>Deterministic??</u>	Yes	No	Partly	No
<u>Static??</u>	Yes	Semi (clock)	Semi	No
<u>Discrete??</u>				
<u>Single-agent??</u>				

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# From Problems to Models

	Solitaire	Backgammon	Internet shopping	Taxi
<u>Observable??</u>	Yes	Yes	No	No
<u>Deterministic??</u>	Yes	No	Partly	No
<u>Static??</u>	Yes	Semi (clock)	Semi	No
<u>Discrete??</u>	Yes	Yes	Yes	No
<u>Single-agent??</u>				

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

# From Problems to Models

	Solitaire	Backgammon	Internet shopping	Taxi
<u>Observable??</u>	Yes	Yes	No	No
<u>Deterministic??</u>	Yes	No	Partly	No
<u>Static??</u>	Yes	Semi (clock)	Semi	No
<u>Discrete??</u>	Yes	Yes	Yes	No
<u>Single-agent??</u>	Yes	No	Yes (but auctions)	No

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

From problem to  
problem models

Course  
objectives

The agent design is largely determined by the problem's model

The real world is (of course) partially observable, stochastic,  
sequential, dynamic, continuous, multi-agent

# Course objectives

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

AI as GPS

NP-hardness

AI Agents as  
Problem  
Solves

Course  
objectives

- ➊ Autonomous systems uncovered
- ➋ Familiarity with some fundamental
  - models of decision problems, and
  - algorithms for solving them
- ➌ Fundamentals of logical reasoning

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Introduction to AI

## Deterministic Planning and State-space Search

Erez Karpas  
Slides by Carmel Domshlak

IE&M — Technion

# Problem-solving agents

## Restricted form of general agent

```
def Simple-Problem-Solving-Agent (problem):
```

state, some description of the current world state

*seq*, an action sequence, initially empty

$state \leftarrow \text{UPDATE-STATE}(state, percept) \rightarrow$  *ρ(θ η)N πατ*

**if** *seq* **is empty then**

$seq \leftarrow \text{SEARCHFOR SOLUTION}(problem)$

$action \leftarrow \text{SELECTACTION}(seq, state)$

**return** *action* .  
תְּמִימָה בְּגִזְעָגָג Picnic

Note: offline problem solving; solution executed “eyes closed.”

אנו נתקיימים בראויים מכך כי נאן היבש הנטה.

## Introduction to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

## Richer formalisms

Back to  
deterministic  
systems

## Back to our problems

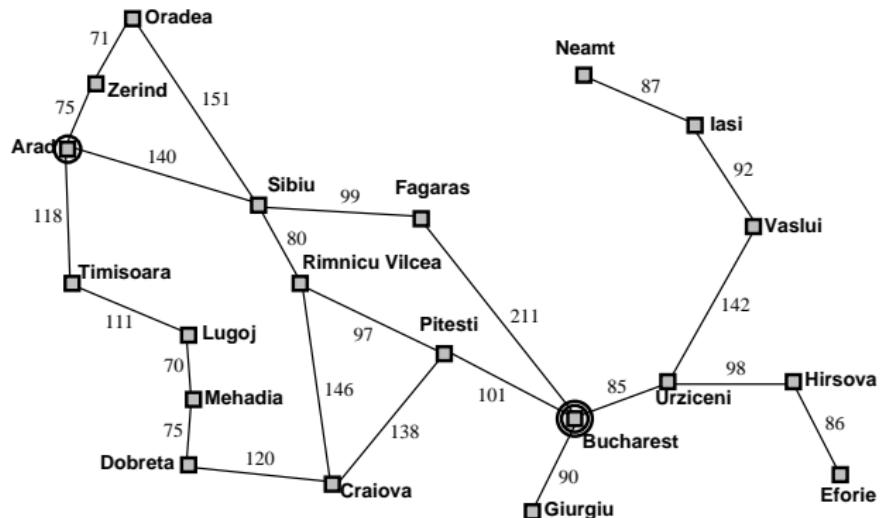
## State-space search

## Uninformed search

# Example: A trip in Romania

## Task

On holiday in Romania; currently in Arad.  
Flight leaves tomorrow from Bucharest



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Example: A trip in Romania

## Task

On holiday in Romania; currently in Arad.

Flight leaves tomorrow from Bucharest

- Formulate goal: be in Bucharest
- Formulate problem:
  - states: various cities
  - actions: drive between cities
- Find solution: sequence of cities,  
e.g., Arad, Sibiu, Fagaras, Bucharest

לען סדרה. (seq.) ↵  
מתקדמת מדרום לצפון ↵  
מתקדמת ממערב למזרח ↵  
Fagaras-Bucharest ↵

You probably know how to solve this one, but ...

. Offline פתרוןOffline ↵. פתרון Offline ↵

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Example: A trip in Romania

## Task

On holiday in Romania; currently in Arad.

Flight leaves tomorrow from Bucharest

- Formulate goal: be in Bucharest
- Formulate problem:
  - states: various cities
  - actions: drive between cities
- Find solution: sequence of cities,  
e.g., Arad, Sibiu, Fagaras, Bucharest

You probably know how to solve this one, **but** ...

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# A sample of problems

- Route selection (from Arad to Bucharest)
- Solving 15-puzzle (or Rubik's cube, or ...)
- Selecting and ordering movements of an elevator or a crane
- Production lines control
- Autonomous robots
- Crisis management
- ...

What is in common?

לכל אחד מהבעיות ישנו מנגנון.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

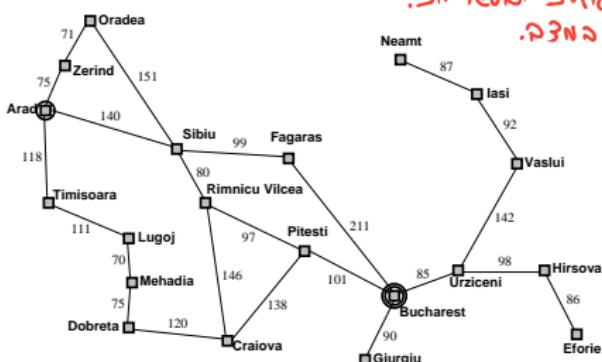
State-space  
search

Uninformed  
search

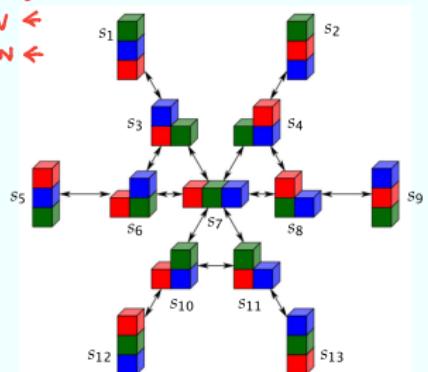
# Planning Problems

What is in common?

- All these problems deal with **action selection or control**
- Some notion of problem state
- (Often) specification of **initial state** and/or **goal state**
- Legal moves or **actions** that allow transition from states to other states



המונחים  
המשמעות  
האפשרויות  
החוקים  
הMOVES



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Planning Problems

For now focus on:

- **Plans** (aka **solutions**) are sequences of moves that transform the initial state into the goal state
- Intuitively, not all solutions are equally desirable

What is our task?

- ➊ Find out whether there is a solution
- ➋ Find any solution
- ➌ Find an optimal solution
- ➍ Find near-optimal solution
- ➎ Fixed amount of time, find best solution possible
- ➏ Find solution that satisfy property  $\Delta$   
(what is  $\Delta$ ? you choose!)

ת'ר'ג'ו  
ג'ו'ג'ו

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Planning Problems

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

## What is our task?

- ➊ Find out whether there is a solution
- ➋ Find any solution
- ➌ Find an optimal solution
- ➍ Find near-optimal solution
- ➎ Fixed amount of time, find best solution possible
- ➏ Find solution that satisfy property  $\aleph$   
(what is  $\aleph$ ? you choose!)

♠ While all these tasks sound related, they are **very different**.  
The best suited techniques are almost disjoint.

# Planning vs. Scheduling

Planning can be seen as **extending** scheduling

## Scheduling

Deciding **when** to perform  
a **given** set of actions

- Time constraints
- Resource constraints
- Global constraints  
(e.g., regulatory issues)
- Objective functions

## Planning

Deciding **what** actions to  
perform (and **when**) to achieve  
a given objective

- same issues

The difference comes in play in solution techniques, and  
actually even in worst-case time/space complexity

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Three Approaches to Problem Solving

- ① **programming**: specify control by hand
- ② **model-oriented solving**:  
specify problem by hand, derive control automatically
- ③ **learning**: learn control from experience

Planning is a form of **model-oriented problem solving**

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# State Model for Deterministic planning

- finite state space  $S$
- an initial state  $s_0 \in S$
- a set  $S_G \subseteq S$  of goal states
- applicable actions  $A(s) \subseteq A$  for  $s \in S$
- a transition function  $s' = f(a, s)$  for  $a \in A(s)$
- a cost function  $c : A^* \rightarrow [0, \infty)$

הינתן אוסף של פעולה  
ונקודת תחילת  $s$   
ונקודת קצה  $s'$  מוגדרת

A **solution** is a sequence of applicable actions that maps  $s_0$  into  $S_G$

An **optimal solution** minimizes  $c$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

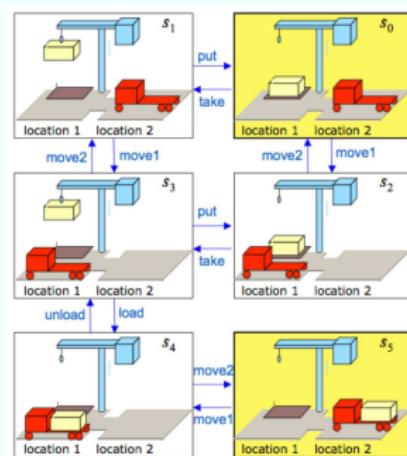
State-space  
search

Uninformed  
search

# State Model for Deterministic planning

- finite state space  $S$
- an initial state  $s_0 \in S$
- a set  $S_G \subseteq S$  of goal states
- applicable actions  
 $A(s) \subseteq A$  for  $s \in S$
- a transition function  
 $s' = f(a, s)$  for  $a \in A(s)$
- a cost function  $c : A^* \rightarrow [0, \infty)$

A **solution** is a sequence of applicable actions that maps  $s_0$  into  $S_G$   
An **optimal solution** minimizes  $c$



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

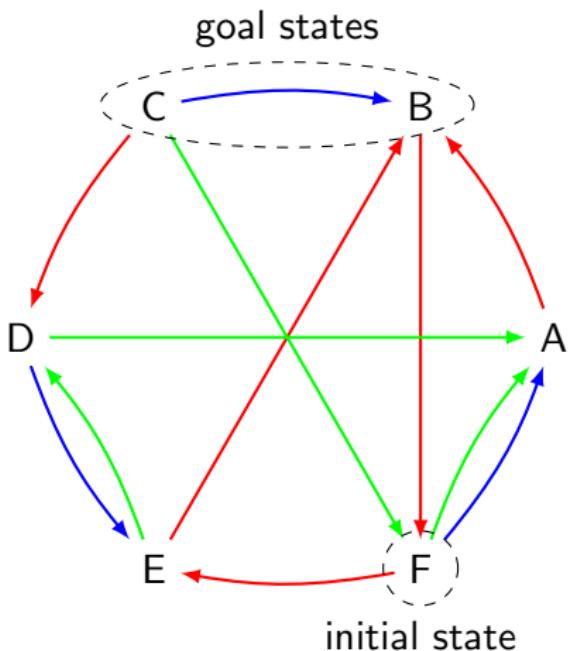
Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Small example



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

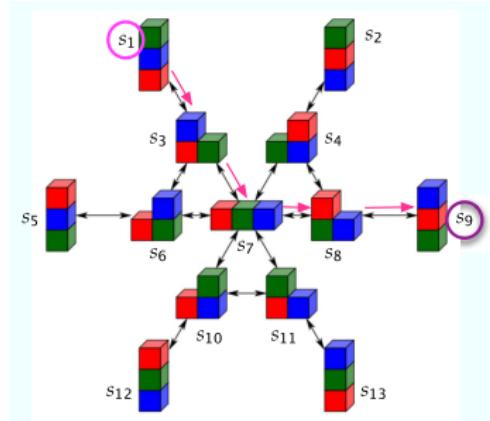
Back to our  
problems

State-space  
search

Uninformed  
search

# Why planning is difficult?

- Solutions to planning problems are **paths** from an initial state to a goal state in the transition graph
- Dijkstra's algorithm solves this problem in  $O(|V| \log (|V|) + |E|)$
- Can we go home??



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

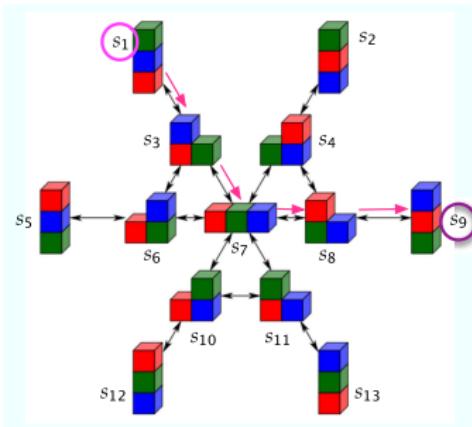
Back to our  
problems

State-space  
search

Uninformed  
search

# Why planning is difficult?

- Solutions to planning problems are **paths** from an initial state to a goal state in the transition graph
- Dijkstra's algorithm solves this problem in  $O(|V| \log (|V|) + |E|)$
- Can we go home??
- ♦ Not exactly  $\Rightarrow |V|$  of our interest is  $10^{10}, 10^{20}, 10^{100}, \dots$
- *But do we need such values of  $|V|$  ?!*



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

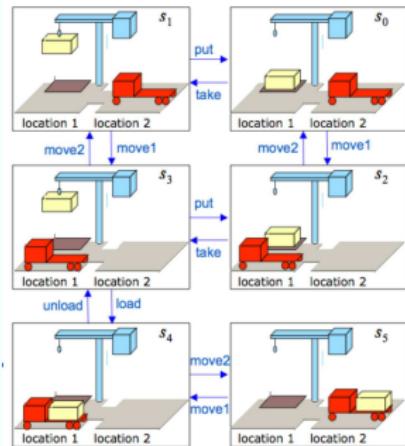
Back to our  
problems

State-space  
search

Uninformed  
search

# Why planning is difficult?

- Generalize the earlier example:
  - Five locations, three robot carts, 100 containers, three piles
  - $|V| \approx 10^{277}$
- The number of atoms in the universe is only about  $10^{87}$ 
  - The state space in our example is more than  $10^{109}$  times as large (upps ...)



Good news:

Very much not hopeless!

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

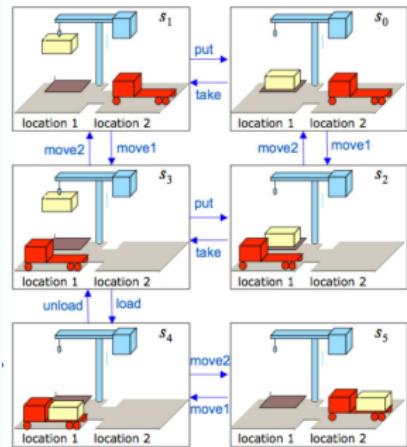
Back to our  
problems

State-space  
search

Uninformed  
search

# Why planning is difficult?

- Generalize the earlier example:
  - Five locations, three robot carts, 100 containers, three piles
  - $|V| \approx 10^{277}$
- The number of atoms in the universe is only about  $10^{87}$ 
  - The state space in our example is more than  $10^{109}$  times as large (upps ...)



Good news:  
Very much not hopeless!

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Different classes of problems

## Properties

- dynamics: deterministic, nondeterministic or probabilistic
- observability: full, partial, or none
- horizon: finite or infinite
- ...

## Side comment ...

- It is not that deterministic problems are easy
- It is not even clear that they are too far from modeling real-world problems well!

השאלה היא מוגן או מוגן ומיותר להציג?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Properties of the world: dynamics

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

## Deterministic dynamics

Action + current state **uniquely** determine successor state.

## Nondeterministic dynamics

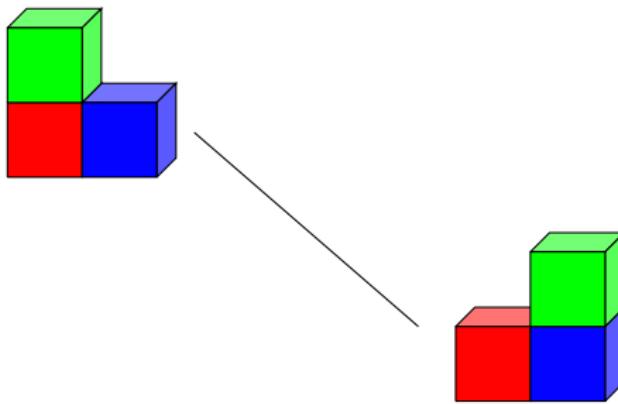
For each action and current state there may be **several possible** successor states.

## Probabilistic dynamics

For each action and current state there is a **probability distribution** over possible successor states.

# Deterministic dynamics example

Moving objects with a robotic hand:  
move the green block onto the blue block.



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

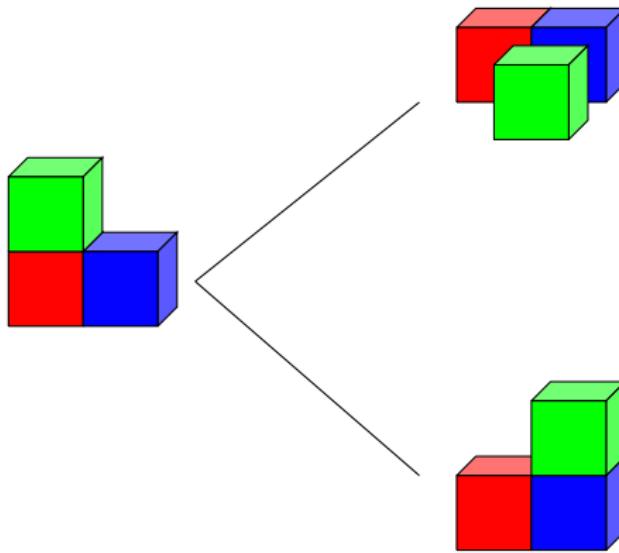
Back to our  
problems

State-space  
search

Uninformed  
search

# Nondeterministic dynamics example

Moving objects with an **unreliable** robotic hand:  
move the green block onto the blue block.



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

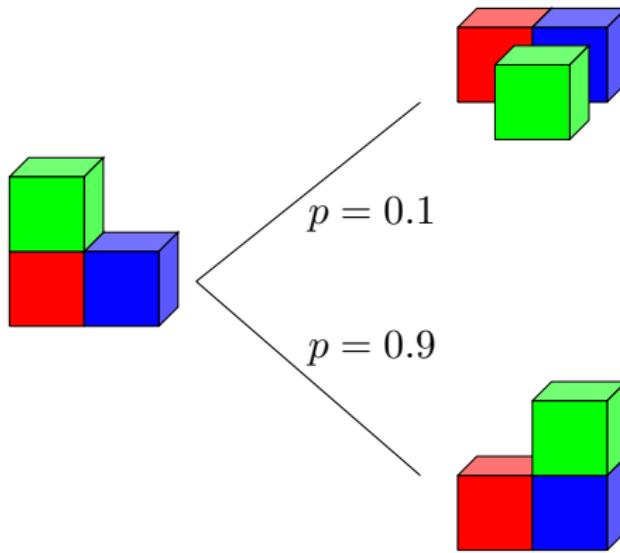
Back to our  
problems

State-space  
search

Uninformed  
search

# Probabilistic dynamics example

Moving objects with an **unreliable** robotic hand:  
move the green block onto the blue block.



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Properties of the world: observability

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

## Full observability

Observations/sensing determine current world state **uniquely**.

## Partial observability

Observations determine current world state **only partially**:  
we only know that current state is one of several possible ones.

## No observability

There are **no observations** to narrow down possible current states. However, can use knowledge of **action dynamics** to deduce which states we might be in.

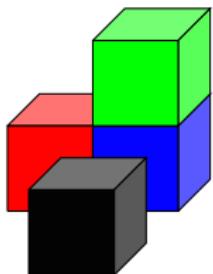
**Consequence:** If observability is not full, must represent the knowledge an agent has.

מזהה ית, מוכנעת ורשות  
. Observability - ה N ית

# What difference does observability make?

Full obs

Camera A

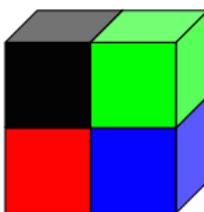


Partial obs

Camera B



Goal



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

## Different objectives

- ① Reach a goal state.
    - Example: Earn 500 NIS.
  - ② Stay in goal states indefinitely (infinite horizon).
    - Example: Never allow the bank account balance to be negative. *账户余额永远不要为负*
  - ③ Maximize the probability of reaching a goal state.
    - Example: To be able to finance buying a house by 2018 study hard and save money.
  - ④ Collect the maximal *expected* rewards/minimal expected costs (infinite horizon).
    - Example: Maximize your future income.
  - ⑤ ...

## Introduction to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

## Dynamics Observability Objectives

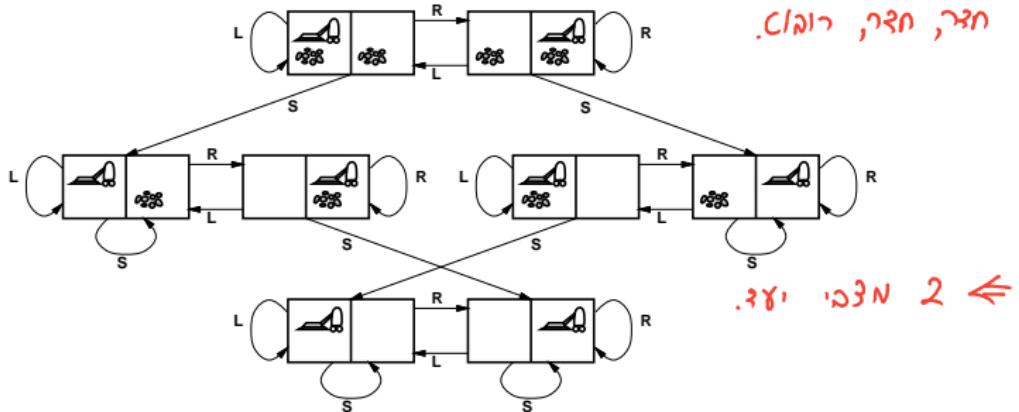
Back to  
deterministic  
systems

## Back to our problems

## State-space search

**Uninformed  
search**

# Example: vacuum world state space graph



states: 0/1 dirt and robot locations (ignore dirt amounts etc.)

actions: *Left, Right, Suck, NoOp*

goal test: no dirt

path cost: 1 per action (0 for *NoOp*)

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

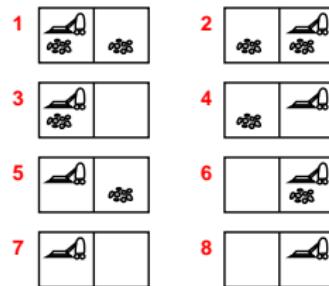
Back to our  
problems

State-space  
search

Uninformed  
search

# Vacuum world: different classes of problems

Single-state, start in #5. Solution?



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

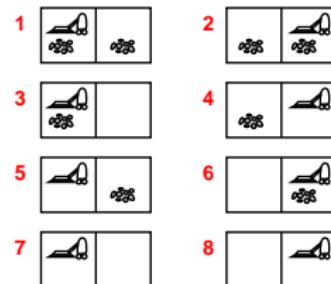
# Vacuum world: different classes of problems

Single-state, start in #5. Solution?

[Right, Suck]

Conformant, start in  $\{1, 2, 3, 4, 5, 6, 7, 8\}$

e.g., Right goes to  $\{2, 4, 6, 8\}$ . Solution?



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Vacuum world: different classes of problems

Single-state, start in #5. Solution?

[Right, Suck]

טַחַת בָּהָר אֶלְגָּה

Conformant, start in  $\{1, 2, 3, 4, 5, 6, 7, 8\}$

e.g., Right goes to  $\{2, 4, 6, 8\}$ . Solution?

[Right, Suck, Left, Suck]

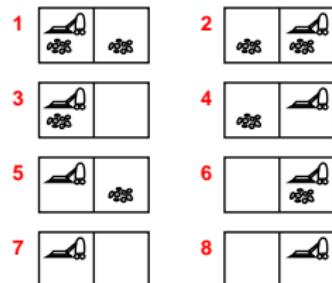
Conditional, start in  $\{5, 7\}$

Murphy's Law: Suck can dirty a clean carpet

Local sensing: dirt, location only.  $\rightarrow$  Partial Obs.

Solution?

[Right, if dirt then Suck]



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Dynamics  
Observability  
Objectives

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Back to deterministic transition systems

A transition system is **deterministic** if there is only **one initial state** and all **actions are deterministic**. Hence all future states of the world are completely predictable.

## Definition (deterministic transition system)

A **deterministic transition system** is  $\langle S, s_0, A, S_G, c \rangle$  where

- finite state space  $S$
- an initial state  $s_0 \in S$
- a set  $S_G \subseteq S$  of goal states
- applicable actions  $A(s) \subseteq A$  for  $s \in S$
- a transition function  $s' = f(a, s)$  for  $a \in A(s)$
- a cost function  $c : A^* \rightarrow [0, \infty)$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Blocks world

The rules of the game

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

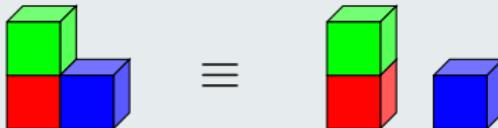
Back to  
deterministic  
systems

Back to our  
problems

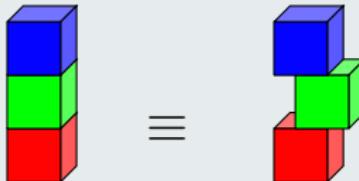
State-space  
search

Uninformed  
search

Location on the table does not matter.



Location on a block does not matter.



# Blocks world

The rules of the game

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

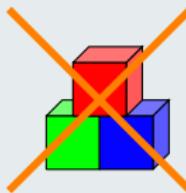
Back to  
deterministic  
systems

Back to our  
problems

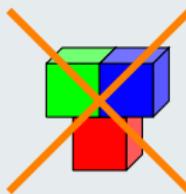
State-space  
search

Uninformed  
search

At most one block may be below a block.

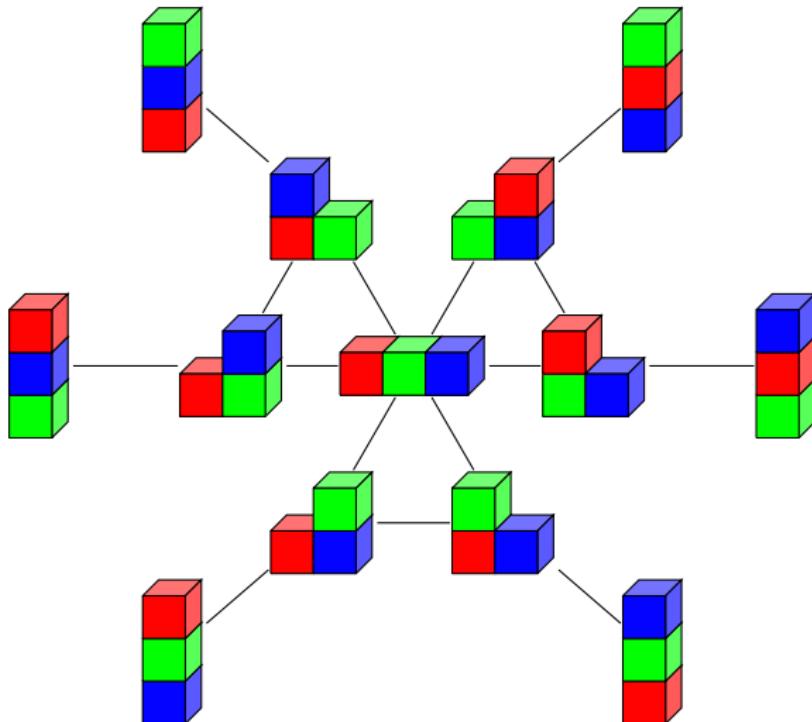


At most one block may be on top of a block.



# Blocks world

The transition graph for three blocks



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Blocks world

## Properties

blocks	states
1	1
2	3
3	13
4	73
5	501
6	4051
7	37633
8	394353
9	4596553
...	
19	13564373693588558173

- ➊ Finding a solution is polynomial time in the number of blocks (move everything onto the table and then construct the goal configuration). *לולא היה לנו גוף אחד*
- ➋ Finding a shortest solution is NP-hard ...

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Planning problems

- Route selection (from Arad to Bucharest)
- Solving 15-puzzle (or Rubik's cube, or ...)
- Selecting and ordering movements of an elevator or a crane
- Production lines control
- Autonomous robots
- Crisis management
- ...

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# State-space search

ח'כ'ל נארה נזק

- State-space search: one of the big success stories of AI
  - Must carefully distinguish two different problems:
    - **satisficing planning:** any solution is OK (although shorter solutions typically preferred)
    - **optimal planning:** plans must have shortest possible length
  - Both are often solved by search, but:
    - details are **very different**
    - almost **no overlap** between good techniques for satisficing planning and good techniques for optimal planning
    - many problems that are trivial for satisficing planners are impossibly hard for optimal planners

פרק י כבשיהו.

(although shorter solutions typically preferred)

כתרון נס ציון

## Introduction to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Back to  
deterministic  
systems

## Back to our problems

## State-space search

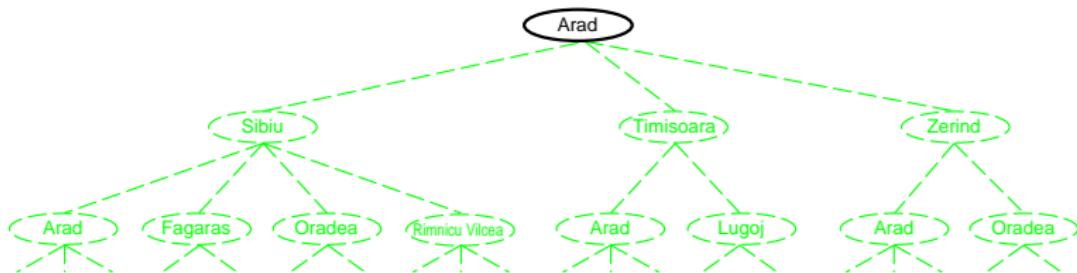
## Uninformed search

# Planning by forward search: progression

**Progression:** Computing the successor state  $f(s, a)$  of a state  $s$  with respect to an action  $a$ .

**Progression planners** find solutions by forward search:

- start from initial state
- iteratively pick a previously generated state and **progress it** through an action, generating a new state
- solution found when a goal state generated



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

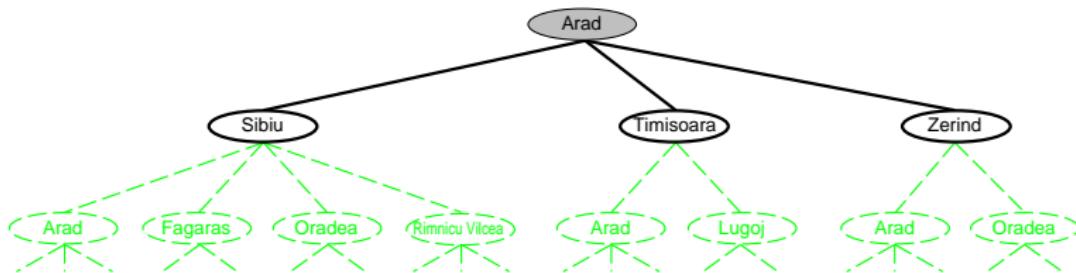
Uninformed  
search

# Planning by forward search: progression

**Progression:** Computing the successor state  $f(s, a)$  of a state  $s$  with respect to an action  $a$ .

**Progression planners** find solutions by forward search:

- start from initial state
- iteratively pick a previously generated state and **progress it** through an action, generating a new state
- solution found when a goal state generated



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

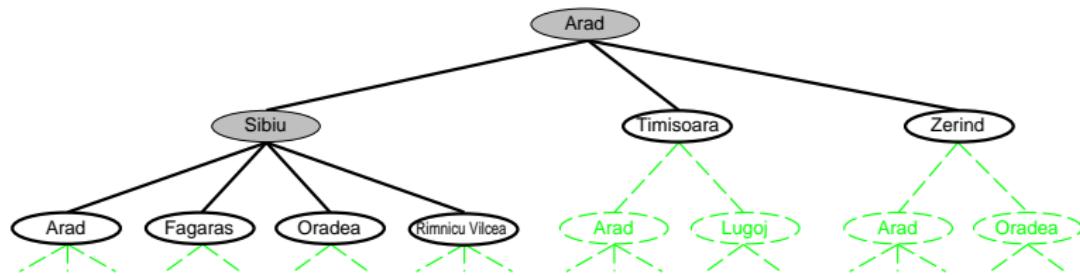
Uninformed  
search

# Planning by forward search: progression

**Progression:** Computing the successor state  $f(s, a)$  of a state  $s$  with respect to an action  $a$ .

**Progression planners** find solutions by forward search:

- start from initial state
- iteratively pick a previously generated state and **progress it** through an action, generating a new state
- solution found when a goal state generated



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Implementation: search states vs. search nodes

In search, one distinguishes:

- **search states  $s$**   $\leadsto$  states (vertices) of the transition system
- **search nodes  $\sigma$**   $\leadsto$  search states plus information on where/when/how they are encountered during search

## What is in a search node?

Different search algorithms store different information in a search node  $\sigma$ , but typical information includes:

- **$state(\sigma)$** : associated search state
- **$parent(\sigma)$** : pointer to search node from which  $\sigma$  is reached
- **$action(\sigma)$** : an action leading from  $state(parent(\sigma))$  to  $state(\sigma)$
- **$g(\sigma)$** : cost of  $\sigma$  (length of path from the root node)

For the root node,  $parent(\sigma)$  and  $action(\sigma)$  are undefined.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

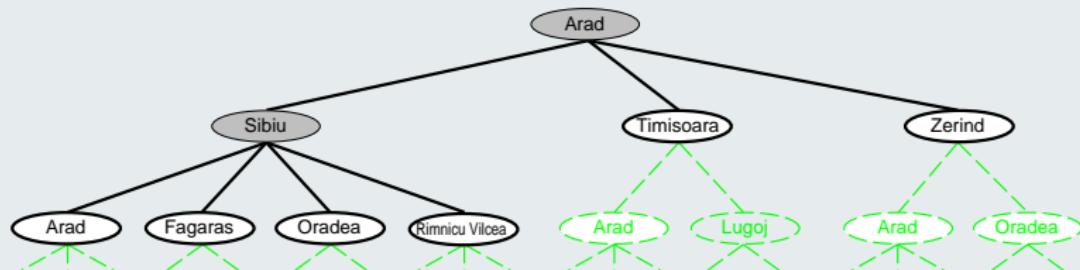
Uninformed  
search

# Implementation: search states vs. search nodes

In search, one distinguishes:

- **search states  $s$**   $\leadsto$  states (vertices) of the transition system
- **search nodes  $\sigma$**   $\leadsto$  search states plus information on where/when/how they are encountered during search

## What is in a search node?



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Required ingredients for search

הנה שלושה נסחים!

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

A general search algorithm can be applied to any transition system for which we can define the following three operations:

- $\text{init}()$ : generate the **initial state** → **האנטיה הראשית**
- $\text{is-goal}(s)$ : test if a given state is a **goal state** → **האנטיה המטרה**
- $\text{succ}(s)$ : generate the set of **successor states** of state  $s$ , along with the **actions** through which they are reached (represented as pairs  $\langle o, s' \rangle$  of actions and states) → **הפעולות**

Together, these three functions form a **search space** (a very similar notion to a transition system).

הו מושג דואלה:  
 $(\text{האנטיה}, \text{ה פעולה})$

# Classification of search algorithms

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

uninformed search vs. heuristic search:

- **uninformed search algorithms** only use the basic ingredients for general search algorithms
- **heuristic search algorithms** additionally use **heuristic functions** which estimate how close a node is to the goal

הירא גודל נס

systematic search vs. local search:

- **systematic algorithms** consider a large number of search nodes simultaneously
- **local search algorithms** work with one (or a few) candidate solutions (search nodes) at a time
- not a black-and-white distinction; there are **crossbreeds** (e.g., enforced hill-climbing)

ר.ג.ן נס נס ר.ג.ן נס

ר.ג.ן נס נס ר.ג.ן נס

# Common procedures for search algorithms

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

כלי בדיקת כבויים

Before we describe the different search algorithms, we introduce three procedures used by all of them:

- **make-root-node:** Create a search node without parent.
- **make-node:** Create a search node for a state generated as the successor of another state.
- **extract-solution:** Extract a solution from a search node representing a goal state.

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Procedure make-root-node

**make-root-node:** Create a search node without parent.

## Procedure make-root-node

```
def make-root-node(s):
    σ := new node
    state(σ) := s
    parent(σ) := undefined
    action(σ) := undefined
    g(σ) := 0
    return σ
```

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Procedure make-node

**make-node:** Create a search node for a state generated as the successor of another state.

## Procedure make-node

```
def make-node( $\sigma$ ,  $o$ ,  $s$ ):  
     $\sigma' := \text{new node}$   
     $\text{state}(\sigma') := s$           (  $\text{State}(\sigma) = S_0$  )  
     $\text{parent}(\sigma') := \sigma$   
     $\text{action}(\sigma') := o$   
     $g(\sigma') := g(\sigma) + 1$   
    return  $\sigma'$ 
```

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Procedure extract-solution

**extract-solution:** Extract a solution from a search node representing a goal state.

## Procedure extract-solution

```
def extract-solution( $\sigma$ ):  
    solution := new list  
    while parent( $\sigma$ ) is defined:  
        solution.push-front(action( $\sigma$ ))  
         $\sigma$  := parent( $\sigma$ )  
    return solution
```

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

# Uninformed search algorithms

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

BFS  
DFS

Popular uninformed systematic search algorithms:

- **breadth-first search**
- depth-first search
- iterated depth-first search

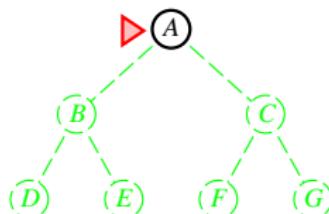
Popular uninformed local search algorithms:

- **random walk**

# Breadth-first search

## Breadth-first search

```
queue := new fifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-back(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

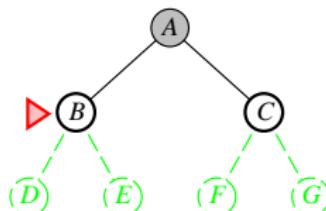
Uninformed  
search

BFS  
DFS

# Breadth-first search

## Breadth-first search

```
queue := new fifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-back(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

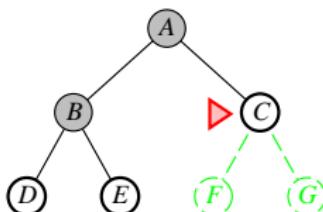
Uninformed  
search

BFS  
DFS

# Breadth-first search

## Breadth-first search

```
queue := new fifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-back(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

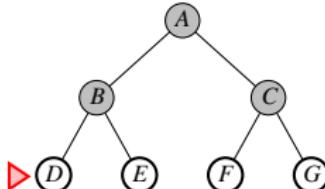
BFS

DFS

# Breadth-first search

## Breadth-first search

```
queue := new fifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-back(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

BFS  
DFS

# Is it any good?

## Dimensions for evaluation

- **completeness**: always find a solution if one exists?
- **time complexity**: number of nodes generated/expanded
- **space complexity**: number of nodes in memory
- **optimality**: does it always find a least-cost solution?

## Time/space complexity measured in terms of

- $b$  maximum branching factor of the search tree ↗ *ט'ה הנקודות בדרכן*
- $d$  depth of the least-cost solution ← *העומק של הפתרון הנכון*
- $m$  maximum depth of the state space (may be  $\infty$ ) ← *העומק של חלל המצבים*

Quiz: <http://bit.ly/2fw14fj>

העומק של חלל המצבים  
... (בשנת 2013)

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

BFS  
DFS

# Properties of breadth-first search

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

BFS  
DFS

**Complete** Yes (if  $b$  is finite)

השאלה מוגבלת פנו



**Time**  $1 + b + b^2 + b^3 + \dots + \cancel{b} + b(b^d - 1) = O(b^{d+1})$ ,  
i.e.,  $\exp(d)$

**Space**  $O(b^{d+1})$  (why?) ← .  
השאלה מוגבלת פנו

**Optimal** Yes (if cost = 1 per step); can be generalized  
(how?)

. BFS פועל בפונקציית-cost

**Space** is the big problem; can easily generate nodes at  
100MB/sec so 24hrs = 8640GB.

# Breadth-first search

## Breadth-first search

```
queue := new fifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-back(σ')
return unsolvable
```

new KB : tree search  
new IP : Graph Search  
. (new FIND) . for BFS

(P)

(S)

- Possible improvement: **duplicate detection** (see next slide).
- Another possible improvement: test if  $\sigma'$  is a goal node; if so, terminate immediately. (We don't do this because it obscures the similarity to some of the later algorithms.)

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

BFS

DFS

# Breadth-first search with duplicate detection

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

BFS

DFS

## Breadth-first search with duplicate detection

```
queue := new fifo-queue
queue.push-back(make-root-node(init()))
closed := []
while not queue.empty():
    σ = queue.pop-front()
    if state(σ) ∉ closed:
        closed := closed ∪ {state(σ)} ←
        if is-goal(state(σ)):
            return extract-solution(σ)
        for each ⟨o, s⟩ ∈ succ(state(σ)):
            σ' := make-node(σ, o, s)
            queue.push-back(σ')
return unsolvable
```

מיצג בדיקת דups  
לניהם נסזם עכבר  
קיים קורן, רקורסיבי  
בהתוצאות הצעה.

# Duplicate detection: Worth the effort?

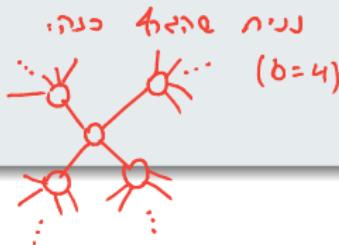
## Worst-case complexity

**Time**  $1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$

Same! 😊 (why?)

**Space**  $O(b^{d+1})$

Same! 😊 (why?)



<http://bit.ly/2f6UAC9>

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

BFS  
DFS

# Duplicate detection: Worth the effort?

## Worst-case complexity

**Time**  $1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$

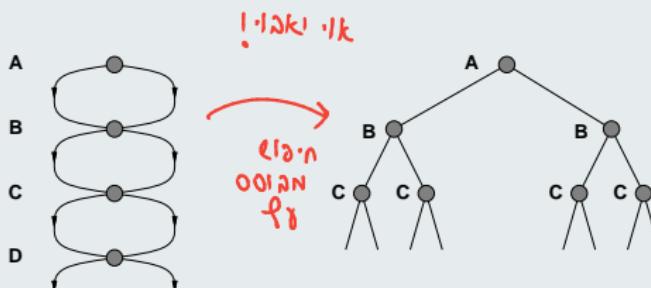
Same! ☺ (why?)

**Space**  $O(b^{d+1})$

Same! ☺ (why?)

## Empirical reality

Failure to detect repeated states can turn a linear problem into an exponential one!



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

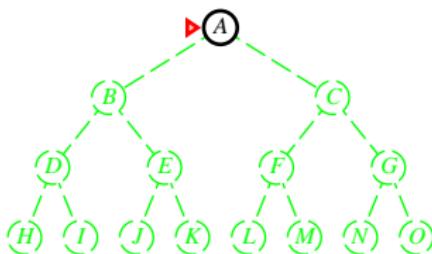
Uninformed  
search

BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

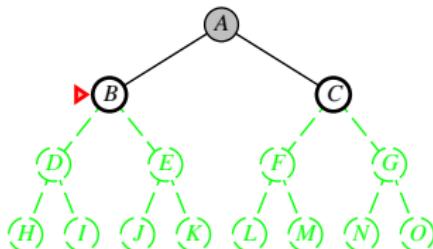
State-space  
search

Uninformed  
search  
BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

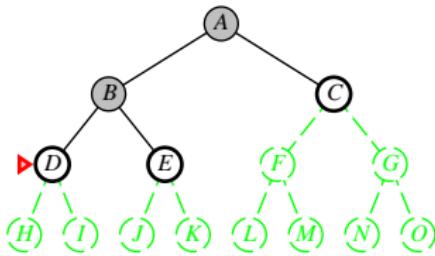
State-space  
search

Uninformed  
search  
BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search  
BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

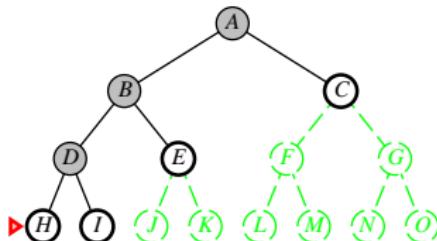
Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

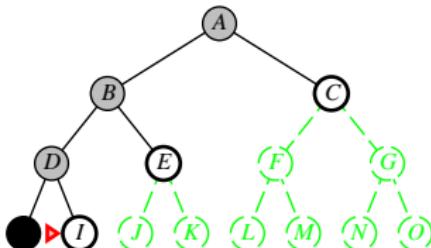
Uninformed  
search  
BFS  
DFS



# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

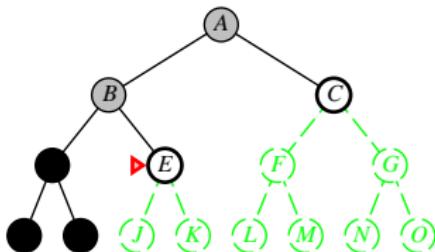
State-space  
search

Uninformed  
search  
BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

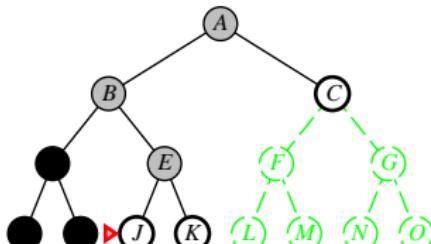
State-space  
search

Uninformed  
search  
BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

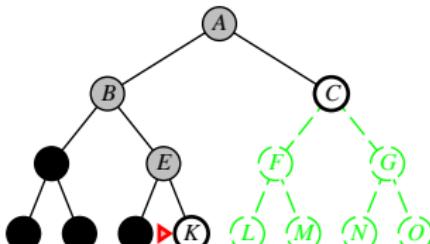
State-space  
search

Uninformed  
search  
BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

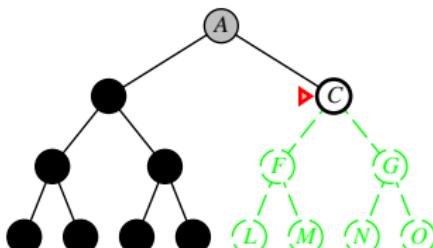
State-space  
search

Uninformed  
search  
BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

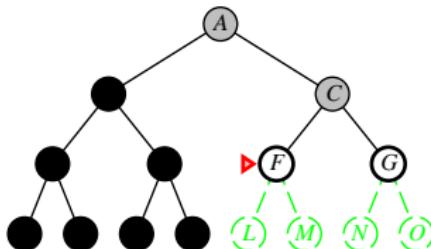
State-space  
search

Uninformed  
search  
BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

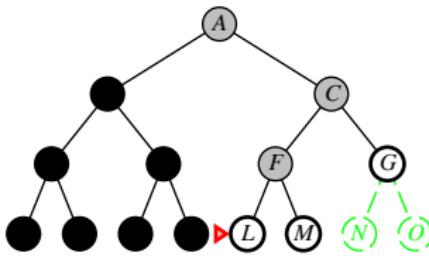
State-space  
search

Uninformed  
search  
BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

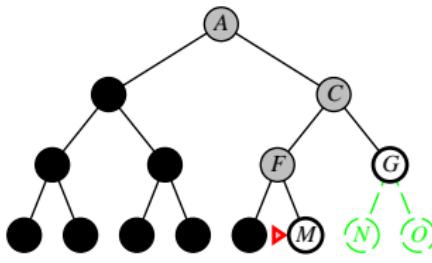
State-space  
search

Uninformed  
search  
BFS  
DFS

# Depth-first search

## Depth-first search

```
queue := new lifo-queue
queue.push-back(make-root-node(init()))
while not queue.empty():
    σ = queue.pop-front()
    if is-goal(state(σ)):
        return extract-solution(σ)
    for each ⟨o, s⟩ ∈ succ(state(σ)):
        σ' := make-node(σ, o, s)
        queue.push-front(σ')
return unsolvable
```



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search  
BFS  
DFS

# Is it any good?

## Dimensions for evaluation

- **completeness**: always find a solution if one exists?
- **time complexity**: number of nodes generated/expanded
- **space complexity**: number of nodes in memory
- **optimality**: does it always find a least-cost solution?

## Time/space complexity measured in terms of

- $b$  maximum branching factor of the search tree
- $d$  depth of the least-cost solution
- $m$  maximum depth of the state space (may be  $\infty$ )

Quiz: <http://bit.ly/2dUHzfZ>

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

BFS  
DFS

# Properties of depth-first search

**Complete** No: fails in spaces with  $b = \infty$  or  $m = \infty$ , spaces with loops

- Possible improvement:  
duplicate detection along path  
 $\leadsto$  complete in finite spaces

**Time**  $O(b^m)$

- terrible if  $m$  is much larger than  $d$ , but
- if solutions are dense, may be much faster than BFS

**Space**  $O(bm) \sim$  linear space!

**Optimal** No (*right?*)

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

BFS  
DFS

# Random walk

## Random walk

```
 $\sigma := \text{make-root-node}(\text{init}())$ 
```

**forever:**

**if** `is-goal(state( $\sigma$ ))`:

**return** `extract-solution( $\sigma$ )`

    Choose a random element  $\langle o, s \rangle$  from `succ(state( $\sigma$ ))`.

$\sigma := \text{make-node}(\sigma, o, s)$

- The algorithm usually does not find any solutions, unless almost every sequence of actions is a plan.
- Often, it runs indefinitely without making progress.
- It can also fail by reaching a **dead end**, a state with no successors. This is a weakness of many local search approaches.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

From  
problems to  
models

Deterministic  
planning

Richer  
formalisms

Back to  
deterministic  
systems

Back to our  
problems

State-space  
search

Uninformed  
search

BFS

DFS

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

# Introduction to AI

## Informed State-space Search

Erez Karpas  
Slides by Carmel Domshlak

IE&M — Technion

# Heuristic search algorithms: systematic

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- Heuristic search algorithms are the most common and overall most successful algorithms for deterministic planning.

Popular systematic heuristic search algorithms:

- greedy best-first search
- A\*
- weighted A\*
- IDA\*
- depth-first branch-and-bound search
- breadth-first heuristic search
- ...

# Heuristic search algorithms: local

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- Heuristic search algorithms are the most common and overall most successful algorithms for deterministic planning.

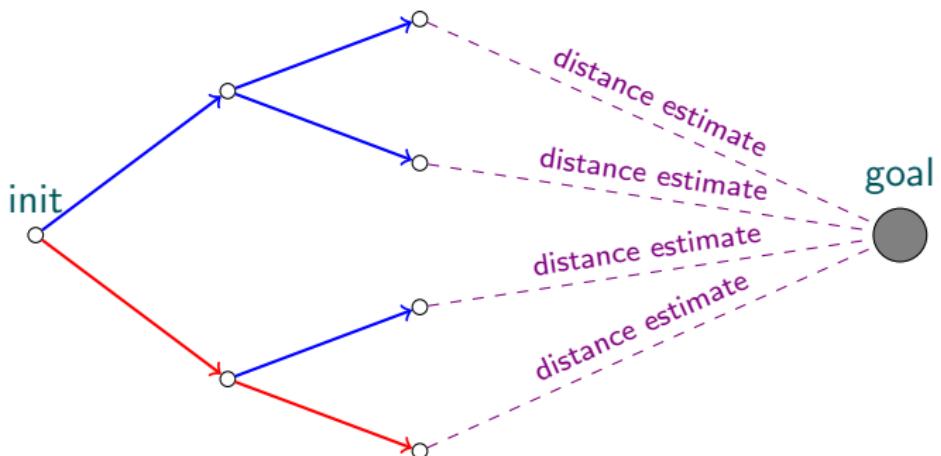
Popular heuristic local search algorithms:

- hill-climbing
- enforced hill-climbing
- beam search
- tabu search
- genetic algorithms
- simulated annealing
- ...

# Heuristic search: idea

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak



# Required ingredients for heuristic search

A **heuristic search algorithm** requires one more operation in addition to the definition of a search space.

## Definition (heuristic function)

Let  $\Sigma$  be the set of nodes of a given search space.

A **heuristic function** or **heuristic** (for that search space) is a function  $h : \Sigma \rightarrow \mathbb{N}_0 \cup \{\infty\}$ .

- The value  $h(\sigma)$  supposed to estimate the distance from node  $\sigma$  to the nearest goal node.
- Typically:  $h(\sigma) \stackrel{\text{def}}{=} h(state(\sigma))$

# What exactly is a heuristic estimate?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

What does it mean that  $h$  “estimates the goal distance”?

- For most heuristic search algorithms,  $h$  does not need to have any strong properties for the algorithm to work
- However, the **efficiency** of the algorithm closely relates to how accurately  $h$  reflects the actual goal distance.
- For some algorithms, like A\*, we can prove strong formal relationships between properties of  $h$  and properties of the algorithm (optimality, dominance, run-time for bounded error, ...)
- For other search algorithms, “it works well in practice” is often as good an analysis as one gets.

# Perfect heuristic

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Let  $\Sigma$  be the set of nodes of a given search space.

## Definition (optimal/perfect heuristic)

The **optimal** or **perfect heuristic** of a search space is the heuristic  $h^*$  which maps each search node  $\sigma$  to the length of a shortest path from  $state(\sigma)$  to any goal state.

**Note:**  $h^*(\sigma) = \infty$  iff no goal state is reachable from  $\sigma$ .

# Properties of heuristics

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

A heuristic  $h$  is called

- **safe** if for all  $\sigma \in \Sigma$   $h(\sigma) = \infty$  implies  $h^*(\sigma) = \infty$
- **goal-aware** if  $h(\sigma) = 0$  for all goal nodes  $\sigma \in \Sigma$
- **admissible** if  $h(\sigma) \leq h^*(\sigma)$  for all nodes  $\sigma \in \Sigma$
- **consistent** if  $h(\sigma) \leq h(\sigma') + cost(\sigma, \sigma')$   
for all nodes  $\sigma, \sigma' \in \Sigma$  such that  $\sigma'$  is a successor of  $\sigma$

Relationships?

# Greedy best-first search

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

## Greedy best-first search (with duplicate detection)

*open* := new min-heap ordered by ( $\sigma \mapsto h(\sigma)$ )

*open.insert*(make-root-node(*init*()))

*closed* :=  $\emptyset$

**while** not *open.empty*():

$\sigma = \text{open.pop-min}()$

**if** *state*( $\sigma$ )  $\notin$  *closed*:

*closed* := *closed*  $\cup$  {*state*( $\sigma$ )}

**if** *is-goal*(*state*( $\sigma$ )):

**return** extract-solution( $\sigma$ )

**for each**  $\langle o, s \rangle \in \text{succ}(\text{state}(\sigma))$ :

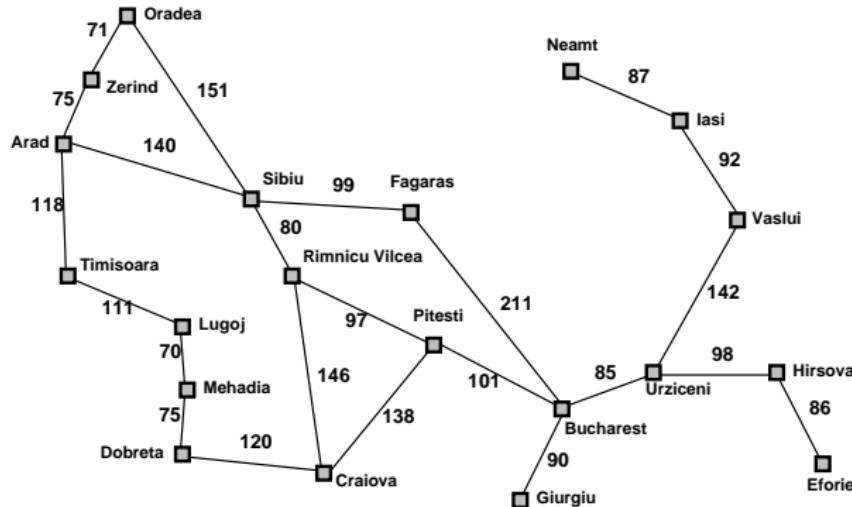
$\sigma' := \text{make-node}(\sigma, o, s)$

**if**  $h(\sigma') < \infty$ :

*open.insert*( $\sigma'$ )

**return** unsolvable

# Planning trip to Bucharest with SLD heuristic



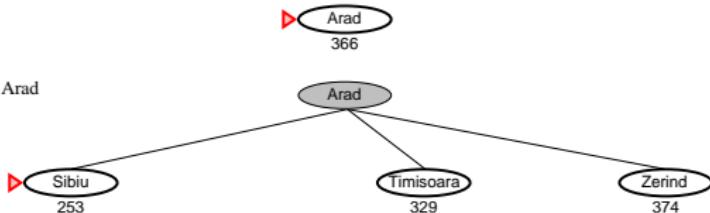
	Straight-line distance to Bucharest
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

# GBFS by example

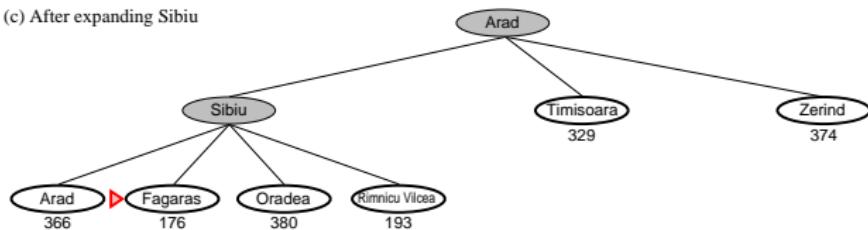
(a) The initial state



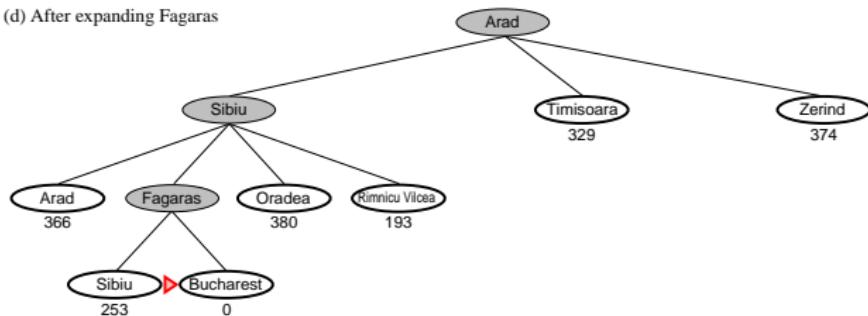
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



# Properties of greedy best-first search

- one of the three most commonly used algorithms for satisficing planning
- **complete** for safe heuristics (due to duplicate detection)
- **suboptimal** unless  $h$  satisfies some very strong assumptions (similar to being perfect)
- invariant under all strictly monotonic transformations of  $h$  (e.g., scaling with a positive constant or adding a constant)

## A\* (with duplicate detection and reopening)

```
open := new min-heap ordered by ( $\sigma \mapsto f(\sigma) = g(\sigma) + h(\sigma)$ )
open.insert(make-root-node(init()))
closed :=  $\emptyset$ 
distance :=  $\emptyset$ 
while not open.empty():
     $\sigma = open.pop-min()$ 
    if state( $\sigma$ )  $\notin$  closed or  $g(\sigma) < distance(state(\sigma))$ :
        closed := closed  $\cup$  {state( $\sigma$ )}
        distance(state( $\sigma$ )) :=  $g(\sigma)$ 
        if is-goal(state( $\sigma$ )):
            return extract-solution( $\sigma$ )
        for each  $\langle o, s \rangle \in succ(state(\sigma))$ :
             $\sigma' :=$  make-node( $\sigma, o, s$ )
            if  $h(\sigma') < \infty$ :
                open.insert( $\sigma'$ )
return unsolvable
```

# A\*(one node maintained per state)

## A\* (with duplicate detection and reopening)

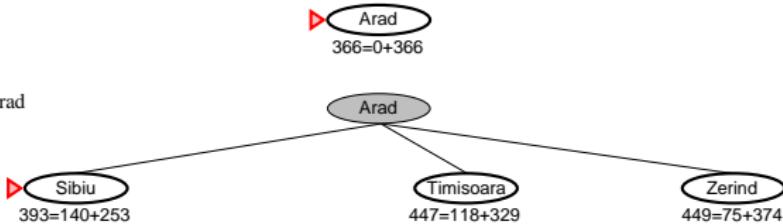
```
open := new min-set-heap ordered by ( $\sigma \mapsto f(\sigma) = g(\sigma) + h(\sigma)$ )
open.insert(make-root-node(init()))
closed :=  $\emptyset$ 
distance :=  $\emptyset$ 
while not open.empty():
     $\sigma = open.pop-min()$ 
    if state( $\sigma$ )  $\notin$  closed or  $g(\sigma) < distance(state(\sigma))$ :
        closed := closed  $\cup$  {state( $\sigma$ )}
        distance(state( $\sigma$ )) :=  $g(\sigma)$ 
        if is-goal(state( $\sigma$ )):
            return extract-solution( $\sigma$ )
        for each  $\langle o, s \rangle \in succ(state(\sigma))$ :
             $\sigma' :=$  make-node( $\sigma, o, s$ )
            if  $h(\sigma') < \infty$ :
                open.insert( $\sigma'$ )
return unsolvable
```

# A\* by example

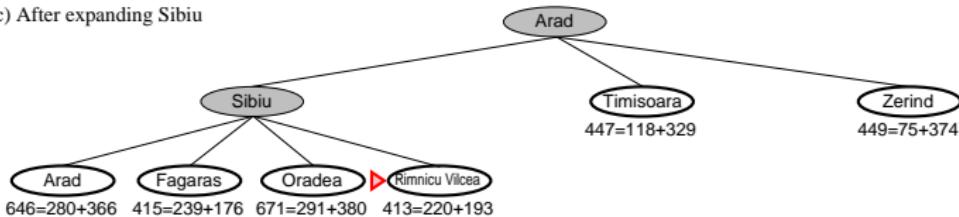
(a) The initial state



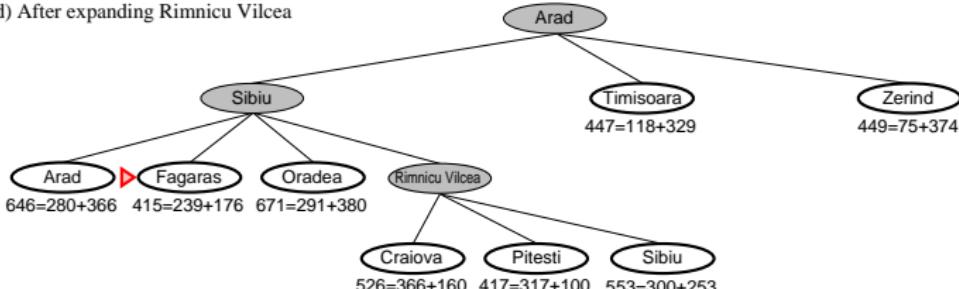
(b) After expanding Arad



(c) After expanding Sibiu



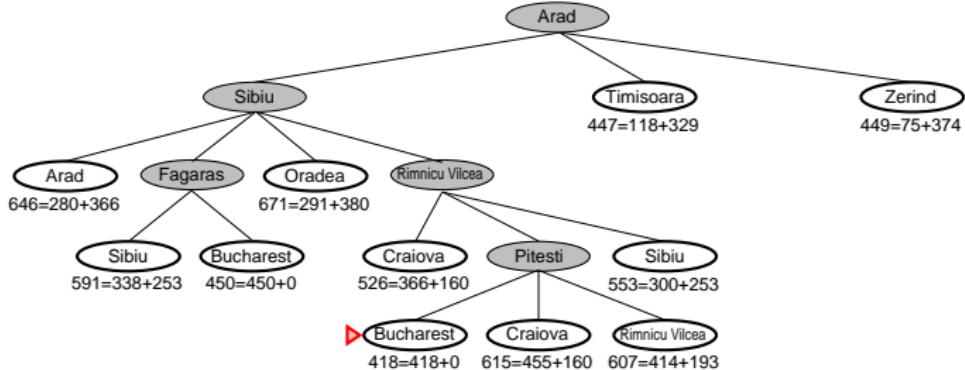
(d) After expanding Rimnicu Vilcea



# A\* by example

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak



# Properties of A\*

- the most commonly used algorithm for optimal planning
- rarely used for satisficing planning
- **complete** for safe heuristics  
(even without duplicate detection)
- **optimal** for admissible heuristics

# Optimality of Tree-Search A\* with Admissible $h$

Standard proof

Suppose to the contrary that the algorithm returns a suboptimal plan via  $\text{extract-solution}(\sigma)$  for some node  $\sigma$  with  $\text{state}(\sigma) \in G$ .

- If so, then no optimal goal node  $\sigma^*$  was in the open list (right?)
- If so, then open list contains some unexpanded node  $\sigma'$  on the (optimal) path from root node to  $\sigma^*$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

$$\begin{aligned} f(\sigma) &= g(\sigma) \quad \text{since } h(\sigma) = 0 \\ &> g(\sigma^*) \quad \text{since } \sigma \text{ is suboptimal} \\ &\geq f(\sigma') \quad \text{since } h \text{ is admissible} \end{aligned}$$

$f(\sigma) > f(\sigma')$   $\rightsquigarrow$  contradiction with “ $\sigma$  is dequeued before  $\sigma'$ ”

# Optimality of Tree-Search A\* with Admissible $h$

Standard proof

Suppose to the contrary that the algorithm returns a suboptimal plan via  $\text{extract-solution}(\sigma)$  for some node  $\sigma$  with  $\text{state}(\sigma) \in G$ .

- If so, then no optimal goal node  $\sigma^*$  was in the open list (right?)
- If so, then open list contains some unexpanded node  $\sigma'$  on the (optimal) path from root node to  $\sigma^*$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

$$\begin{aligned} f(\sigma) &= g(\sigma) \quad \text{since } h(\sigma) = 0 \\ &> g(\sigma^*) \quad \text{since } \sigma \text{ is suboptimal} \\ &\geq f(\sigma') \quad \text{since } h \text{ is admissible} \end{aligned}$$

$f(\sigma) > f(\sigma')$   $\rightsquigarrow$  contradiction with “ $\sigma$  is dequeued before  $\sigma'$ ”

# Weighted A\*

## Weighted A\* (with duplicate detection and reopening)

```
open := new min-heap ordered by ( $\sigma \mapsto g(\sigma) + W \cdot h(\sigma)$ )
open.insert(make-root-node(init()))
closed :=  $\emptyset$ 
distance :=  $\emptyset$ 
while not open.empty():
     $\sigma = open.pop-min()$ 
    if state( $\sigma$ )  $\notin$  closed or  $g(\sigma) < distance(state(\sigma))$ :
        closed := closed  $\cup \{state(\sigma)\}$ 
        distance( $\sigma$ ) :=  $g(\sigma)$ 
        if is-goal(state( $\sigma$ )):
            return extract-solution( $\sigma$ )
        for each  $\langle o, s \rangle \in succ(state(\sigma))$ :
             $\sigma' :=$  make-node( $\sigma, o, s$ )
            if  $h(\sigma') < \infty$ :
                open.insert( $\sigma'$ )
return unsolvable
```

## Properties of weighted A\*

## Introduction to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

The **weight**  $W \in \mathbb{R}_0^+$  is a parameter of the algorithm.

- for  $W = 0$ , behaves like breadth-first search
  - for  $W = 1$ , behaves like A\*
  - for  $W \rightarrow \infty$ , behaves like greedy best-first search

## Properties:

- one of the three most commonly used algorithms for satisficing planning
  - for  $W > 1$ , can prove similar properties to A\*, replacing **optimal** with **bounded suboptimal**: generated solutions are at most a factor  $W$  as long as optimal ones

מונר,  $\varphi A^*$  מיש יתגלה כתבו שילם כ. 2 ( $\varphi N$ ).  
ויר ניכרין ( $\varphi \Omega(\gamma, N)$ ).

## Hill-climbing

.PIG n.CAN Kd

## Hill-climbing

$\sigma := \text{make-root-node}(\text{init}())$

**forever:**

**if** `is-goal(state( $\sigma$ )):`

כָּבֵד נִתְמַלֵּט

**return** extract-solution( $\sigma$ )

$$\Sigma' := \{ \text{make-node}(\sigma, o, s) \mid \langle o, s \rangle \in \text{succ}(\text{state}(\sigma)) \}$$

$\sigma :=$  an element of  $\Sigma'$  minimizing  $h$  (random tie breaking)

בְּנֵתֶךָ נִזְמַן זֶה אֲלֵיכֶם הַנִּזְמָן.

- can easily get stuck in **local minima** where immediate improvements of  $h(\sigma)$  are not possible
  - many variations: tie-breaking strategies, restarts

# Enforced hill-climbing

היל קלימבינג-ס BFS ינ' נ' אל'ג'

Enforced hill-climbing: procedure improve

```
def improve( $\sigma_0$ ):  
    queue := new fifo-queue  
    queue.push-back( $\sigma_0$ )  
    closed :=  $\emptyset$   
    while not queue.empty():  
         $\sigma$  = queue.pop-front()  
        if state( $\sigma$ )  $\notin$  closed:  
            closed := closed  $\cup$  {state( $\sigma$ )}  
            if  $h(\sigma) < h(\sigma_0)$ :  
                return  $\sigma$   
            for each  $\langle o, s \rangle \in \text{succ}(\text{state}(\sigma))$ :  
                 $\sigma' := \text{make-node}(\sigma, o, s)$   
                queue.push-back( $\sigma'$ )  
    fail
```

היל קלימבינג-ס BFS ינ' נ' אל'ג'  
היל קלימבינג-ס BFS ינ' נ' אל'ג'  
היל קלימבינג-ס BFS ינ' נ' אל'ג'

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

~ breadth-first search for more promising node than  $\sigma_0$

# Enforced hill-climbing (ctd.)

## Enforced hill-climbing

```
 $\sigma := \text{make-root-node}(\text{init}())$ 
```

```
while not is-goal(state( $\sigma$ )):
```

```
     $\sigma := \text{improve}(\sigma)$ 
```

```
return extract-solution( $\sigma$ )
```

- one of the three most commonly used algorithms for satisficing planning
- can fail if procedure *improve* fails (when the goal is unreachable from  $\sigma_0$ )
- complete for **undirected** search spaces (where the successor relation is symmetric) if  $h(\sigma) = 0$  for all goal nodes and only for goal nodes

# Classification: what works where in (deterministic) planning?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

uninformed vs. heuristic search:

- For **satisficing** planning, heuristic search vastly outperforms uninformed algorithms on most domains.
- For **optimal** planning, heuristic search typically outperforms uninformed algorithms, but an efficiently implemented uninformed algorithm is not easy to beat in most domains.

systematic search vs. local search:

- For **satisficing** planning, the most successful algorithms are somewhere between the two extremes.
- For **optimal** planning, systematic algorithms are required.

# Where heuristics come from?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

## General idea

(Admissible) heuristic functions obtained as  
**(optimal) cost functions of relaxed problems**

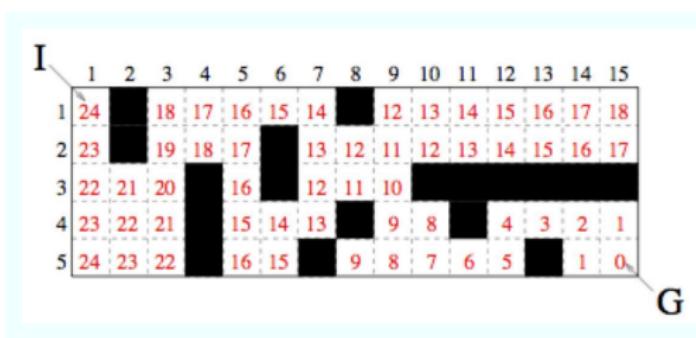
## Examples

- Euclidian distance in Path Finding
- Manhattan distance in N-puzzle
- Spanning Tree in Traveling Salesman Problem
- Shortest Path in Job Shop Scheduling

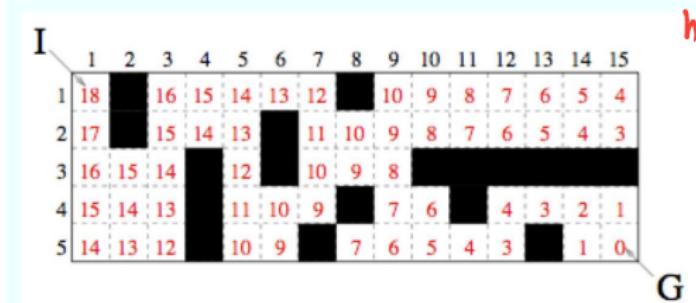
# Example

Grid Search and Manhattan Distance

True distance  $h^*$  for different search states



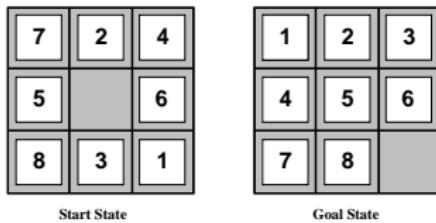
Manhattan distance is based on the relaxation that ...?



$$h = \Delta x + \Delta y$$

# Example

## 8-Puzzle



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

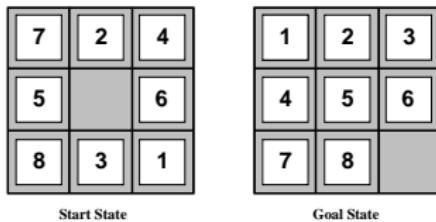
- A tile can move from square A to square B if A is adjacent to B and B is blank  $\leadsto$  solution distance  $h^*$
- A tile can move from square A to square B if A is adjacent to B  $\leadsto$  manhattan distance heuristic  $h^{MD}$ . (אנו ינורא כוונתית) ←
- A tile can move from square A to square B  $\leadsto$  misplaced tiles heuristic  $h^{MT}$ . (טבל כנה קווינטלי) ←

Here:  $h^*(s_0) = ?$ ,  $h^{MD}(s_0) = 14$ ,  $h^{MT}(s_0) = 6$

In general,  $h^* \geq h^{MD} \geq h^{MT}$ . (Why?)

# Example

## 8-Puzzle



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- A tile can move from square A to square B if A is adjacent to B and B is blank  $\leadsto$  solution distance  $h^*$  המינימום של המרחק
- A tile can move from square A to square B if A is adjacent to B  $\leadsto$  manhattan distance heuristic  $h^{MD}$  המינימום של המרחק
- A tile can move from square A to square B  $\leadsto$  misplaced tiles heuristic  $h^{MT}$  המינימום של המרחק

Here:  $h^*(s_0) = ?$ ,  $h^{MD}(s_0) = 14$ ,  $h^{MT}(s_0) = 6$

In general,  $h^* \geq h^{MD} \geq h^{MT}$ . (Why?) המינימום של המרחק המינימום של המרחק המינימום של המרחק

# Dominance relation between admissible heuristics

## Precision matters

Given two admissible heuristics  $h_1, h_2$ , if  $h_2(\sigma) \geq h_1(\sigma)$  for all search nodes  $\sigma$ , then  $h_2$  **dominates**  $h_1$  and is better for optimizing search

ה<sub>1</sub> מושג בדרכן קיימת צדקה ל<sub>2</sub> בA\* ← ה<sub>2</sub> מושג בדרכן קיימת צדקה ל<sub>1</sub> בA\*

## Typical search costs (unit-cost action)

$$h^*(I) = 14 \quad \text{BFS} \approx 1,700,000 \text{ nodes}$$

ה<sub>1</sub> מושג בדרכן קיימת צדקה ל<sub>2</sub> בA\* ←  
ה<sub>2</sub> מושג בדרכן קיימת צדקה ל<sub>1</sub> בA\*

$$h^*(I) = 24 \quad \text{BFS} \approx 27,000,000,000 \text{ nodes}$$

$$h^*(I) = 24 \quad \text{BFS} \approx 27,000,000,000 \text{ nodes}$$

ה<sub>1</sub> מושג בדרכן קיימת צדקה ל<sub>2</sub> בA\* ←  
ה<sub>2</sub> מושג בדרכן קיימת צדקה ל<sub>1</sub> בA\*

$$A^*(h_1) \approx 560 \text{ nodes}$$
$$A^*(h_2) \approx 115 \text{ nodes}$$

$$A^*(h_1) \approx 40,000 \text{ nodes}$$

ה<sub>1</sub> מושג בדרכן קיימת צדקה ל<sub>2</sub> בA\* ←  
ה<sub>2</sub> מושג בדרכן קיימת צדקה ל<sub>1</sub> בA\*

$$A^*(h_2) \approx 1,650 \text{ nodes}$$

## Dominance relation between admissible heuristics

# Precision matters

Given two admissible heuristics  $h_1, h_2$ , if  $h_2(\sigma) \geq h_1(\sigma)$  for all search nodes  $\sigma$ , then  $h_2$  **dominates**  $h_1$  and is better for optimizing search

## Introduction to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

## Combining admissible heuristics

For any admissible heuristics  $h_1, \dots, h_k$ ,

$$h(\sigma) = \max_{i=1}^k \{h_i(\sigma)\}$$

is also admissible and dominates all individual  $h_i$ .

# Are we solver?

## General idea

(Admissible) heuristic functions obtained as (optimal) cost functions of relaxed problems

- OK, but heuristic is **yet another input** to our agent!
- Satisfactory for general solvers?
- Satisfactory in special purpose solvers?

## Towards domain-independent agents

- How to get heuristics **automatically**?
- Can such automatically derived heuristics **dominate** the domain-specific heuristics crafted by hand?

# Are we solver?

## General idea

(Admissible) heuristic functions obtained as (optimal) cost functions of relaxed problems

- OK, but heuristic is **yet another input** to our agent!
- Satisfactory for general solvers?
- Satisfactory in special purpose solvers?

## Towards domain-independent agents

- How to get heuristics **automatically**?
- Can such automatically derived heuristics **dominate** the domain-specific heuristics crafted by hand?

# Introduction to AI

## Domain-independent planning

Carmel Domshlak

IE&M — Technion

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

# Reminder: Are we solver?

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

## General idea

(Admissible) heuristic functions obtained as (optimal) cost functions of relaxed problems

- OK, but heuristic is **yet another input** to our agent!
- Satisfactory for general solvers?
- Satisfactory in special purpose solvers?

## Towards domain-independent agents

- How to get heuristics **automatically**?
- Can such automatically derived heuristics **dominate** the domain-specific heuristics crafted by hand?

# Model-based Problem Solving

We want planning to take the form of **model-based solving**

Problem  $\Rightarrow$  Language  $\Rightarrow$  **Planner**  $\Rightarrow$  Solution

- ① **models** for defining, classifying, and understanding problems
  - what is a *planning problem*
  - what is a *solution (plan)*, and
  - what is an *optimal solution*
- ② **languages** for representing problems
- ③ **algorithms** for solving them

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification  
State variables  
Tasks  
Action  
Languages

Obtaining  
heuristics

Concrete  
Heuristics

# Succinct representation of transition systems

- More **compact** representation of actions than as relations is often
  - **possible** because of symmetries and other regularities,
  - **unavoidable** because the relations are too big.
- Represent different aspects of the world in terms of different **state variables**.  
  ~ A state is a **valuation of state variables**.
- Represent actions in terms of changes to the state variables.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

State variables

Tasks

Action

Languages

Obtaining  
heuristics

Concrete  
Heuristics

# State variables

- The state of the world is described in terms of a **finite set** of **finite-valued** state variables.

## Example

*hour*:  $\{0, \dots, 23\} = 13$

*minute*:  $\{0, \dots, 59\} = 55$

*location*:  $\{51, 52, 82, 101, 102\} = 101$

*weather*:  $\{\text{sunny}, \text{cloudy}, \text{rainy}\} = \text{cloudy}$

*holiday*:  $\{\text{T}, \text{F}\} = \text{F}$

- Actions change the values of the state variables.

הפעולות מוחזקות במשתנים  
המשתנים מוחזקים במקומות קבועים

Introduction  
to AI

C. Domshlak

Introduction

Problem

specification

State variables

Tasks

Action

Languages

Obtaining  
heuristics

Concrete

Heuristics

# Blocks world with state variables

Introduction  
to AI

C. Domshlak

Introduction

Problem

specification

State variables

Tasks

Action

Languages

Obtaining  
heuristics

Concrete  
Heuristics

State variables:

*location-of-A*: {B, C, table}

*location-of-B*: {A, C, table}

*location-of-C*: {A, B, table}

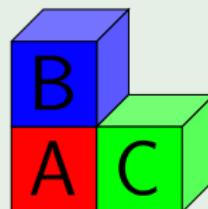
Example

תא ני תרבותה לְפָנֶיךָ אֵין  
.S P3NP ↓

$s(\text{location-of-}A) = \text{table}$

$s(\text{location-of-}B) = A$

$s(\text{location-of-}C) = \text{table}$



Not all valuations correspond to an intended blocks world state, e.g.  $s$  such that  $s(\text{location-of-}A) = B$  and  $s(\text{location-of-}B) = A$ .

# Blocks world with Boolean state variables

## Example

‘עכטן דיאז’

$$s(A\text{-on-}B) = 0$$

$$s(A\text{-on-}C) = 0$$

$$s(A\text{-on-table}) = 1$$

$$s(B\text{-on-}A) = 1$$

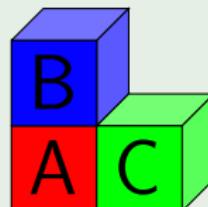
$$s(B\text{-on-}C) = 0$$

$$s(B\text{-on-table}) = 0$$

$$s(C\text{-on-}A) = 0$$

$$s(C\text{-on-}B) = 0$$

$$s(C\text{-on-table}) = 1$$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

State variables

Tasks

Action  
Languages

Obtaining  
heuristics

Concrete  
Heuristics

# Deterministic planning tasks

Introduction  
to AI

C. Domshlak

## Definition (deterministic planning task)

A **deterministic planning task** is a 4-tuple  $\Pi = \langle V, I, A, G \rangle$

- $V$  is a finite set of **state variables**,
- $I$  is an **initial state** over  $V$ ,
- $A$  is a finite set of **actions** over  $V$ , and
- $G$  is a **formula** over  $V$  describing the **goal states**.
  - in what follows: partial assignment to  $V$

"B on C == 1" : few ↘

~ Mapping planning tasks to transition systems

Introduction

Problem  
specification

State variables

Tasks

Action  
Languages

Obtaining  
heuristics

Concrete  
Heuristics

# Planning Languages

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification  
State variables  
Tasks  
Action  
Languages

Obtaining  
heuristics

Concrete  
Heuristics

## Key issue

Models represented **implicitly** in a **declarative language**

## Play two roles

- **specification**: concise model description
- **computation**: reveal useful info about problem's *structure*

# The STRIPS language

Introduction  
to AI

C. Domshlak

Introduction

Problem specification

State variables

Tasks

Action Languages

Obtaining heuristics

Concrete Heuristics

A problem in **STRIPS** is a tuple  $\langle P, A, I, G \rangle$

- $P$  stands for a finite set of **atoms** (boolean vars)
- $I \subseteq P$  stands for **initial situation**
- $G \subseteq P$  stands for **goal situation** → *Goal State*  $\rightarrow$  *Goal Condition*  $\rightarrow$  *Goal Function*
- $A$  is a finite set of **actions**  $a$  specified via  $\text{pre}(a)$ ,  $\text{add}(a)$ , and  $\text{del}(a)$ , all subsets of  $P$
- States are collections of atoms  $\leftarrow$  *Pinch*  $\rightarrow$  *States*
- An action  $a$  is applicable in a state  $s$  iff  $\text{pre}(a) \subseteq s$
- Applying an applicable action  $a$  at  $s$  results in  $s' = (s \setminus \text{del}(a)) \cup \text{add}(a)$

.*T Pre condition of goal*  $\rightarrow$  *Goal Condition*  $\rightarrow$  *Goal Function*

# Why STRIPS is interesting

Pre(a) - מושג יסוד בSTRIPS  
הנורמליזציה של S-ים  
מגדיר מושג הנקרא Pre(a). כלומר, Pre(a) מגדיר מושג הנקרא Pre(a).

add(a) - מושג יסוד בSTRIPS  
הנורמליזציה של actions

del(a) - מושג יסוד בSTRIPS

- STRIPS actions are **particularly simple**, yet expressive enough to capture general planning problems.
- In particular, STRIPS planning is **no easier** than general planning problems.
- Many algorithms in the planning literature are **easier to present in terms of STRIPS**.

.STRIPS - מושג יסוד בSTRIPS

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

State variables

Tasks

Action  
Languages

Obtaining  
heuristics

Concrete  
Heuristics

# A simple heuristic for deterministic planning

STRIPS סטריפס

STRIPS (Fikes & Nilsson, 1971) used the number of state variables that differ in current state  $s$  and a STRIPS goal  
 $G = \{g_1, \dots, g_k\}$ :

$$h(s) := |G \setminus s|.$$

Intuition: more true goal literals  $\rightsquigarrow$  closer to the goal

$\rightsquigarrow$  STRIPS heuristic (properties?)

- goal aware ✓

- admissible X

הנ"ט מבחן צפויותי י"

הנ"ט 2 כפאות נור

הנ"ט 1 אהכטן.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Concrete  
Heuristics

# Criticism of the STRIPS heuristic

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics  
The relaxation  
lemma  
Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

## What is wrong with the STRIPS heuristic?

- quite **uninformative**:  
the range of heuristic values in a given task is small;  
typically, most successors have the same estimate
  - very sensitive to **reformulation**:  
can easily transform any planning task into an equivalent  
one where  $h(s) = 1$  for all non-goal states (how?)
  - ignores almost all **problem structure**:  
heuristic value does not depend on the set of actions!
- ~ need a better, principled way of coming up with heuristics

now 4.015  
as per your  
KS the gen 23N

# Coming up with heuristics in a principled way

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristics  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

## General procedure for obtaining a heuristic

Solve an easier version of the problem.

Two common methods:

- **relaxation:** consider **less constrained** version of the problem
- **abstraction:** consider **smaller** version of real problem

Both have been very successfully applied in planning  
(separately and *together*).

We consider **relaxation**.

# Relaxing a problem

How do we relax a problem?

## Example (Route planning for a road network)

The road network is formalized as a weighted graph over points in the Euclidean plane. The weight of an edge is the **road distance** between two locations.

A relaxation **drops constraints** of the original problem.

## Example (Relaxation for route planning)

Use the **Euclidean distance**  $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$  as a heuristic for the road distance between  $(x_1, x_2)$  and  $(y_1, y_2)$ . This is a **lower bound** on the road distance ( $\leadsto$  admissible).

$\leadsto$  We drop the constraint of having to travel on roads.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Relaxations for planning

- Relaxation is a general technique for heuristic design:
  - **Straight-line heuristic** (route planning): Ignore the fact that one must stay on roads.
  - **Manhattan heuristic** (15-puzzle): Ignore the fact that one cannot move through occupied tiles.
- We want to apply the idea of relaxations to planning.
- Informally, we want to ignore **bad side effects** of applying actions.

## Example (8-puzzle)

If we move a tile from  $x$  to  $y$ , then the **good effect** is (in particular) that  $x$  is now free.

The **bad effect** is that  $y$  is not free anymore, preventing us from moving tiles through it.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Relaxed planning tasks: idea

In STRIPS, good and bad effects are easy to distinguish:

- Effects that make atoms true are good  
(**add effects**).
- Effects that make atoms false are bad  
(**delete effects**).

Idea for the heuristic: **Ignore all delete effects.**

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Concrete  
Heuristics

# Relaxed planning tasks: idea

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Concrete  
Heuristics

In STRIPS, good and bad effects are easy to distinguish:

- Effects that make atoms true are good  
(**add effects**).  $\text{add}(a)$
- Effects that make atoms false are bad  
(**delete effects**).  $\text{del}(a)$

Idea for the heuristic: **Ignore all delete effects.**

# Relaxed planning tasks

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

## Definition (relaxation of actions)

The **relaxation**  $a^+$  of a STRIPS action

$a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$  is the action

$a^+ = \langle \text{pre}(a), \text{add}(a), \emptyset \rangle$ .

## Definition (relaxation of planning tasks)

The **relaxation**  $\Pi^+$  of a STRIPS planning task  $\Pi = \langle P, A, I, G \rangle$

is the planning task  $\Pi^+ := \langle P, \{a^+ \mid a \in A\}, I, G \rangle$ .

## Definition (relaxation of action sequences)

The **relaxation** of an action sequence  $\rho = a_1 \dots a_n$  is the action

sequence  $\rho^+ := a_1^+ \dots a_n^+$ .

# Relaxed planning tasks: terminology

- STRIPS planning tasks without delete effects are called **relaxed planning tasks**.
- Plans for relaxed planning tasks are called **relaxed plans**.
- If  $\Pi$  is a STRIPS planning task and  $\rho^+$  is a plan for  $\Pi^+$ , then  $\rho^+$  is called a **relaxed plan for  $\Pi$** .

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristics  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Example: Logistics

: תרגום נבנ



- Initial state  $I$ :  $\{at(A, Left), at(T, Left), at(B, Right)\}$
- $f(I, Drive(Left, Right)) = \otimes$   
 $\{at(A, Left), \textcolor{blue}{at(T, Right)}, at(B, Right)\}$  הזקקה גייתה פראס
- $f(I, Drive(Left, Right)^+) = \leftarrow$   
 $\{at(A, Left), \textcolor{red}{at(T, Left)}, \textcolor{blue}{at(T, Right)}, at(B, Right)\}$  פראס
- $f(I, \langle Drive(Left, Right), Load(A, Left) \rangle)$  is undefined
- $f(I, \langle Drive(Left, Right)^+, Load(A, Left)^+ \rangle) =$   
 $\{at(A, Left), \textcolor{red}{at(T, Left)}, \textcolor{blue}{at(T, Right)}, at(B, Right), \textcolor{blue}{in(A, T)}\}$  מגיעה מכאן פראס, מילויים

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

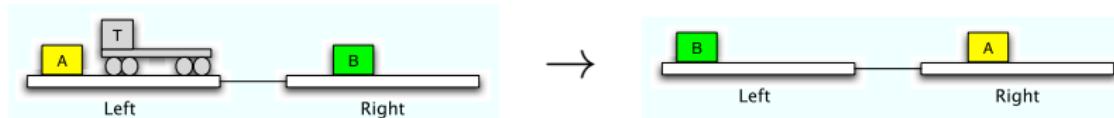
STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

## Example: Logistics



- Optimal plan:

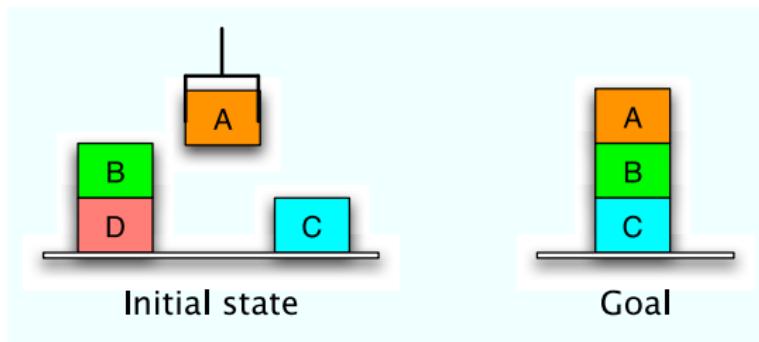
- 1  $load(A, T, Left),$
  - 2  $drive(Left, Right),$
  - 3  $unload(A, T, Right),$
  - 4  $load(B, T, Right),$
  - 5  $drive(Right, Left), \text{ } \leftarrow$
  - 6  $unload(B, T, Left)\}$

הנורווגית

- Optimal relaxed plan: ??? (subsequence of the optimal plan)
  - $h^*(I) = 6, h^+(I) = ???$  ↗

# Always subsequence? (Just curious)

An optimal relaxed plan can *not* always be obtained by skipping actions from the (real) optimal plan.



- Optimal plan:  
 $\langle \text{putdown}(A), \text{unstack}(B, D), \text{stack}(B, C), \text{pickup}(A), \text{stack}(A, B) \rangle$
- Optimal relaxed subsequence: ???  $\text{stack}(A, B), \text{pickup}(B)$   
 $\text{stack}(B, C)$
- Optimal relaxed plan: ???  
הינתן לנו סדרה של אctions. האם היא תת-סדרה?

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

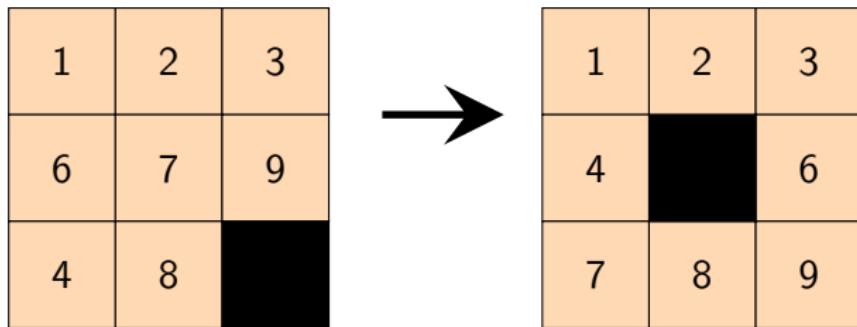
Greedy algorithm

Optimality

Discussion

Concrete  
Heuristics

# Example: 8-Puzzle



- Real problem:
  - A tile can move from square A to square B if A is adjacent to B and B is blank
- Monotonically relaxed problem:
  - A tile can move from square A to square B if A is adjacent to B and B is blank (!!!)
  - In effect ...

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

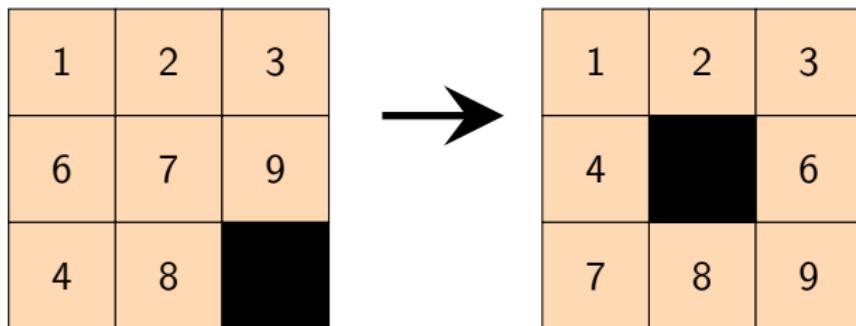
STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Example: 8-Puzzle



- A tile can move from square A to square B if A is adjacent to B and B is blank - solution distance  $h^*$
- A tile can move from square A to square B if A is adjacent to B - manhattan distance heuristic  $h^{MD}$
- A tile can move from square A to square B if A is adjacent to B and B is blank; in effect, the tile is at both A and B, and both A and B are blank -  $h^+$

Here:  $h^*(s_0) = 8$ ,  $h^{MD}(s_0) = 6$ ,  $h^+(s_0) = ???$

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Concrete  
Heuristics

# Example: 8-Puzzle

1	2	3
6	7	9
4	8	



1	2	3
4		6
7	8	9

Optimal MD plan:

- ①  $move(t_9, p_6, p_9)$
- ②  $move(t_7, p_5, p_8)$
- ③  $move(t_6, p_4, p_5)$
- ④  $move(t_6, p_5, p_6)$
- ⑤  $move(t_4, p_7, p_4)$
- ⑥  $move(t_7, p_8, p_7)$

Optimal relaxed plan:

- ①  $move(t_9, p_6, p_9)$
- ②  $move(t_8, p_8, p_9)$
- ③  $move(t_7, p_5, p_8)$
- ④  $move(t_6, p_4, p_5)$
- ⑤  $move(t_6, p_5, p_6)$
- ⑥  $move(t_4, p_7, p_4)$
- ⑦  $move(t_7, p_8, p_7)$

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Example: 8-Puzzle

1	2	3
6	7	9
4	8	



1	2	3
4		6
7	8	9

Optimal MD plan:

- ❶  $move(t_9, p_6, p_9)$
- ❷  $move(t_7, p_5, p_8)$
- ❸  $move(t_6, p_4, p_5)$
- ❹  $move(t_6, p_5, p_6)$
- ❺  $move(t_4, p_7, p_4)$
- ❻  $move(t_7, p_8, p_7)$

Optimal relaxed plan:

- ❶  $move(t_9, p_6, p_9)$
- ❷  $move(t_8, p_8, p_9)$
- ❸  $move(t_7, p_5, p_8)$
- ❹  $move(t_6, p_4, p_5)$
- ❺  $move(t_6, p_5, p_6)$
- ❻  $move(t_4, p_7, p_4)$
- ❼  $move(t_7, p_8, p_7)$

המקרה  
בז' רצינית  
בנ' איה יתג'ר

בנ' יתג'ר  
. זט'!!

So  $h^*(s_0) = 8$ ,  $h^{MD}(s_0) = 6$ ,  $h^+(s_0) = 7 (> h^{MD}!)$

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality

Discussion

Concrete  
Heuristics

# 8-Puzzle: $h^+$ vs. $h^{MD}$

1	2	3
6	7	9
4	8	



1	2	3
4		6
7	8	9

*MP<sub>i</sub> (blank)  $\geq$  blank  $\oplus$  non-blank blanks.  $\Rightarrow$  MP  $\leq$  h<sup>+</sup>*

$h^+$  dominates  $h^{MD}$

- The goal is given as a conjunction of  $at(t_i, p_j)$  atoms
- Achieving each single one of them takes at least as many steps as the respective tile's Manhattan distance
- Each action moves a single tile only

And we have just seen that  $h^+$  strictly dominates  $h^{MD}$

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Concrete  
Heuristics

# Dominating states

תנאי חישוב גזע ב פס. ש-ה פס יוצר סדרה יפה נ-ו ב-  
. ש' פס סדרה יוניק ס-ה

A state  $s'$  **dominates** another state  $s$  iff  $s \subseteq s'$ .

## Lemma (relaxation)

Let  $s$  be a state, let  $s'$  be a state that dominates  $s$ , and let  $\rho$  be an action sequence which is applicable in  $s$ .

Then  $\rho^+$  is applicable in  $s'$  and  $app_{\rho^+}(s')$  dominates  $app_\rho(s)$ . Moreover, if  $\rho$  leads to a goal state from  $s$ , then  $\rho^+$  leads to a goal state from  $s'$ .

## Proof.

The “moreover” part is immediate from  $app_{\rho^+}(s')$  dominating  $app_\rho(s)$ . Prove the rest by induction over the length of  $\rho$ .

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Consequences of the relaxation lemma

לעג'ז

Corollary (relaxation leads to dominance and preserves plans)

Let  $\rho$  be an action sequence which is applicable in state  $s$ .

Then  $\rho^+$  is applicable in  $s$  and  $app_{\rho^+}(s)$  dominates  $app_{\rho}(s)$ .

If  $\rho$  is a plan for  $\Pi$ , then  $\rho^+$  is a plan for  $\Pi^+$ .

Proof.

Apply relaxation lemma with  $s' = s$ . □

- ~ Relaxations of plans are relaxed plans.
- ~ Relaxations are no harder to solve than the original task.
- ~ Optimal relaxed plans are never longer than optimal plans for original tasks.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Concrete  
Heuristics

# Consequences of the relaxation lemma (ctd.)

15<5.3

## Corollary (relaxation preserves dominance)

Let  $s$  be a state, let  $s'$  be a state that dominates  $s$ , and let  $\rho^+$  be a relaxed action sequence applicable in  $s$ . Then  $\rho^+$  is applicable in  $s'$  and  $\text{app}_{\rho^+}(s')$  dominates  $\text{app}_{\rho^+}(s)$ .

## Proof.

Apply relaxation lemma with  $\rho^+$  for  $\rho$ , noting that  $(\rho^+)^+ = \rho^+$ .



- ~ If there is a relaxed plan starting from state  $s$ , the same plan can be used starting from a dominating state  $s'$ .
- ~ Making a transition to a dominating state never hurts in relaxed planning tasks.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Concrete  
Heuristics

# Monotonicity of relaxed planning tasks

15/19

We need one final property before we can provide an algorithm for solving relaxed planning tasks.

## Lemma (monotonicity)

Let  $a^+ = \langle \text{pre}(a), \text{add}(a), \emptyset \rangle$  be a relaxed action and let  $s$  be a state in which  $a^+$  is applicable.

Then  $\text{app}_{a^+}(s)$  dominates  $s$ .

## Proof.

Since relaxed actions only have positive effects, we have

$$s \subseteq s \cup \text{add}(a) = \text{app}_{a^+}(s).$$



~ Together with our previous results, this means that making a transition in a relaxed planning task **never** hurts.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

## Greedy algorithm for relaxed planning tasks

The relaxation and monotonicity lemmas suggest the following algorithm for solving relaxed planning tasks:

Greedy planning algorithm for  $\langle P, A^+, I, G \rangle$

$s_i = I$

$\rho^+ := \epsilon$  ← גורן יון  
forever:

**if**  $G \in s$ :

**return**  $\rho^+$

**else if** there is an action  $a^+ \in A^+$  applicable in  $s$

with  $app_{a+}(s) \neq s$ :

Append such an action  $a^+$  to  $\rho^+$ .

$s := app_{a^+}(s)$

**else:**

**return** unsolvable

$\text{P}_{\text{N}} = \frac{\rho_{\text{N}}}{\rho_{\text{air}}}$

## Introduction to AI

C. Domshlak

## Obtaining heuristics

Relaxation  
heuristics  
The relaxation

## Greedy algorithm Optimality

## Concrete Heuristics

# Correctness of the greedy algorithm

The algorithm is **sound**:

- If it returns a plan, this is indeed a correct solution.
- If it returns “unsolvable”, the task is indeed unsolvable
  - Upon termination, there clearly is no relaxed plan from  $s$ .
  - By iterated application of the monotonicity lemma,  $s$  dominates  $I$ .
  - By the relaxation lemma, there is no solution from  $I$ .

What about **completeness** (termination) and **runtime**?

- Each iteration of the loop adds at least one atom to  $s$ .
- This guarantees termination after at most  $|P|$  iterations.
- Thus, the algorithm can clearly be implemented to run in polynomial time.
  - A good implementation runs in  $O(\|\Pi\|)$ .

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Using the greedy algorithm as a heuristic

מבחן גס נרמולן נון:  $h$

We can apply the greedy algorithm within heuristic search:

- In a search node  $\sigma$ , solve the relaxation of the planning task with  $state(\sigma)$  as the initial state.
- Set  $h(\sigma)$  to the length of the generated relaxed plan.

Is this an **admissible** heuristic?

- Yes if the relaxed plans are **optimal** (due to the plan preservation corollary).
- However, usually they are not, because our greedy planning algorithm is very poor.

(What about safety? Goal-awareness? Consistency?)

טבז לא יס סנו לא

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# The set cover problem

ה问题是怎样的？

To obtain an admissible heuristic, we need to generate optimal relaxed plans. Can we do this efficiently?

This question is related to the following problem:

## Problem (set cover)

**Given:** a finite set  $U$ , a collection of subsets  $C = \{C_1, \dots, C_n\}$  with  $C_i \subseteq U$  for all  $i \in \{1, \dots, n\}$ , and a natural number  $K$ .

**Question:** Does there exist a set cover of size at most  $K$ , i. e., a subcollection  $S = \{S_1, \dots, S_m\} \subseteq C$  with  $S_1 \cup \dots \cup S_m = U$  and  $m \leq K$ ?

The following is a classical result from complexity theory:

## Theorem

The set cover problem is NP-complete.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Hardness of optimal relaxed planning

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

## Theorem (optimal relaxed planning is hard)

*The problem of deciding whether a given relaxed planning task has a plan of length at most  $K$  is NP-complete.*

## Proof.

For membership in NP, guess a plan and verify. It is sufficient to check plans of length at most  $|P|$ , so this can be done in nondeterministic polynomial time.

For hardness, we reduce from the set cover problem.

# Hardness of optimal relaxed planning (ctd.)

## Proof (ctd.)

Given a set cover instance  $\langle U, C, K \rangle$ , we generate the following relaxed planning task  $\Pi^+ = \langle P, I, A^+, G \rangle$ :

- $P = U$
- $I = \emptyset \quad \equiv I = \{p = 0 \mid p \in P\}$
- $A^+ = \{\langle \emptyset, \bigcup_{p \in C_i} \{p\}, \emptyset \rangle \mid C_i \in C\}$
- $G = U$

If  $S$  is a set cover, the corresponding actions form a plan.

Conversely, each plan induces a set cover by taking the subsets corresponding to the actions. Clearly, there exists a plan of length at most  $K$  iff there exists a set cover of size  $K$ .

Moreover,  $\Pi^+$  can be generated from the set cover instance in polynomial time, so this is a polynomial reduction. □

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS

heuristic

Relaxation

heuristics

The relaxation

lemma

Greedy algorithm

Optimality

Discussion

Concrete

Heuristics

# Using relaxations in practice

How can we use relaxations for heuristic planning in practice?

Different possibilities:

- Implement an **optimal planner** for relaxed planning tasks and use its solution lengths as an estimate, even though it is NP-hard.  
~~~  $h^+$  heuristic (*not that realistic. why?*) ... condition p'noob
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.  
~~~  $h_{\max}$  heuristic,  $h_{\text{add}}$  heuristic
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but “reasonable”.  
~~~  $h_{\text{FF}}$  heuristic

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Using relaxations in practice

How can we use relaxations for heuristic planning in practice?

Different possibilities:

- Implement an **optimal planner** for relaxed planning tasks and use its solution lengths as an estimate, even though it is NP-hard.  
~~~  $h^+$  **heuristic** (*not that realistic. why?*)
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.  
~~~  $h_{\max}$  **heuristic**,  $h_{\text{add}}$  **heuristic**
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but “reasonable”.  
~~~  $h_{\text{FF}}$  **heuristic**

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Using relaxations in practice

How can we use relaxations for heuristic planning in practice?

Different possibilities:

- Implement an **optimal planner** for relaxed planning tasks and use its solution lengths as an estimate, even though it is NP-hard.  
  ~  $h^+$  **heuristic** (*not that realistic. why?*)
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.  
  ~  $h_{\max}$  **heuristic**,  $h_{\text{add}}$  **heuristic**
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but “reasonable”.  
  ~  $h_{\text{FF}}$  **heuristic**

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Reminder: Greedy algorithm for relaxed planning tasks

## Greedy planning algorithm for $\langle P, A^+, I, G \rangle$

$s := I$

$\rho^+ := \epsilon$

**forever:**

**if**  $G \subseteq s$ :

**return**  $\rho^+$

**else if** there is an action  $a^+ \in A^+$  applicable in  $s$

        with  $app_{a^+}(s) \neq s$ :

            Append such an action  $a^+$  to  $\rho^+$ .

$s := app_{a^+}(s)$

**else:**

**return** unsolvable

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

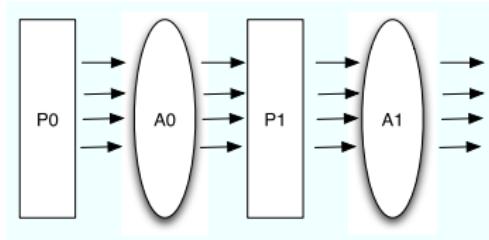
# Graphical “interpretation”: Relaxed planning graphs

- Build a layered **reachability graph**  $P_0, A_0, P_1, A_1, \dots$

$$P_0 = \{p \in I\}$$

$$A_i = \{a \in A \mid \text{pre}(a) \subseteq P_i\}$$

$$P_{i+1} = P_i \cup \{p \in \text{add}(a) \mid a \in A_i\}$$



- Terminate when  $G \subseteq P_i$

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h^{\text{max}}$   
 $h^{\text{add}}$   
 $h^{\text{FF}}$   
Comparison &  
practice

# Running example

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

$$P = \{a, b, c, d, e, f, g, h\}$$

$$I = \{a\}$$

$$G = \{c, d, e, f, g\}$$

$$a_1 = \langle \{a\}, \{b, c\}, \emptyset \rangle$$

$a_1$  *not yet closed*

$$a_2 = \langle \{a, c\}, \{d\}, \emptyset \rangle$$

*not yet closed*

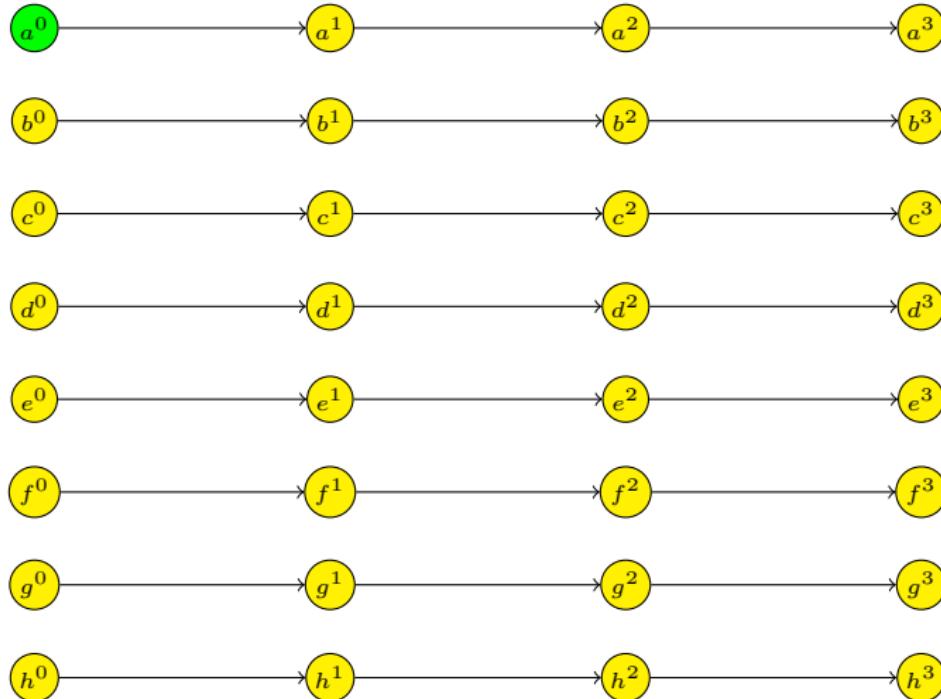
$$a_3 = \langle \{b, c\}, \{e\}, \emptyset \rangle$$

$$a_4 = \langle \{b\}, \{f\}, \emptyset \rangle$$

$$a_5 = \langle \{d\}, \{e, f\}, \emptyset \rangle$$

$$a_6 = \langle \{d\}, \{g\}, \emptyset \rangle$$

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

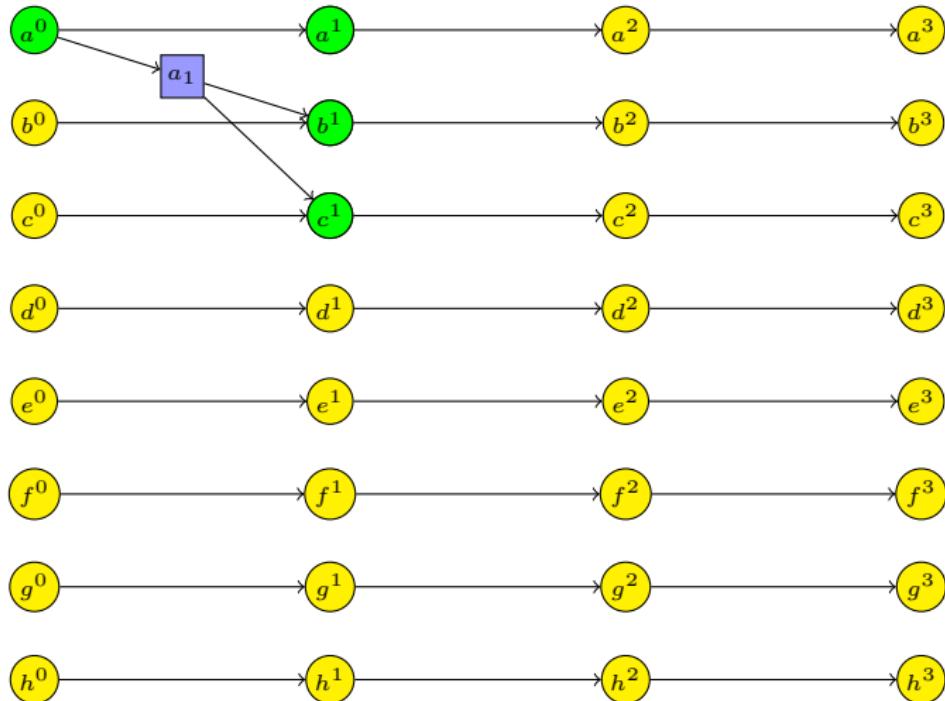
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

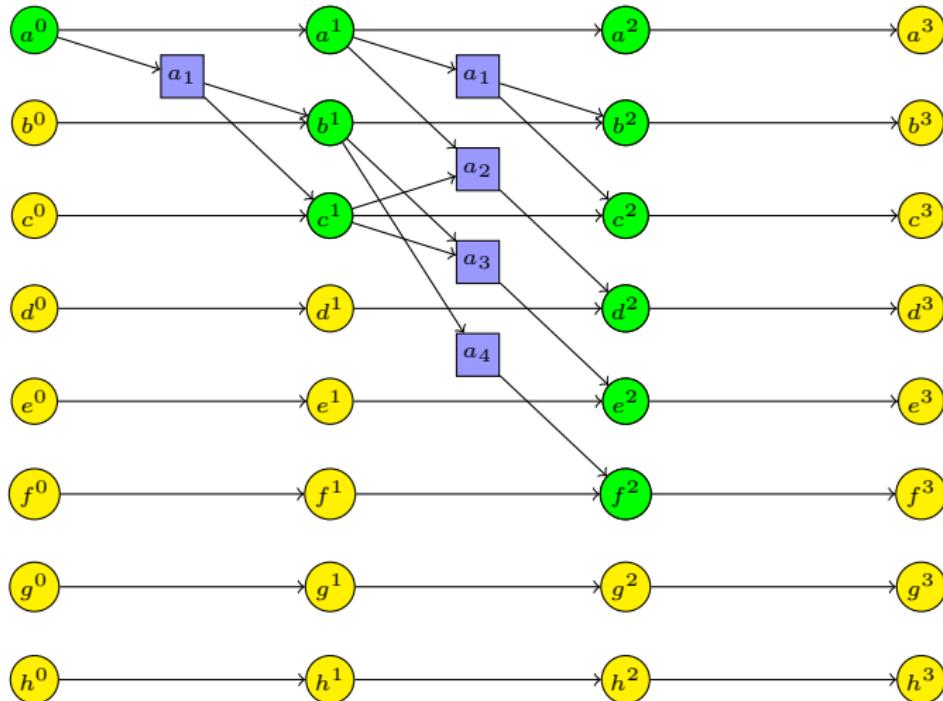
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

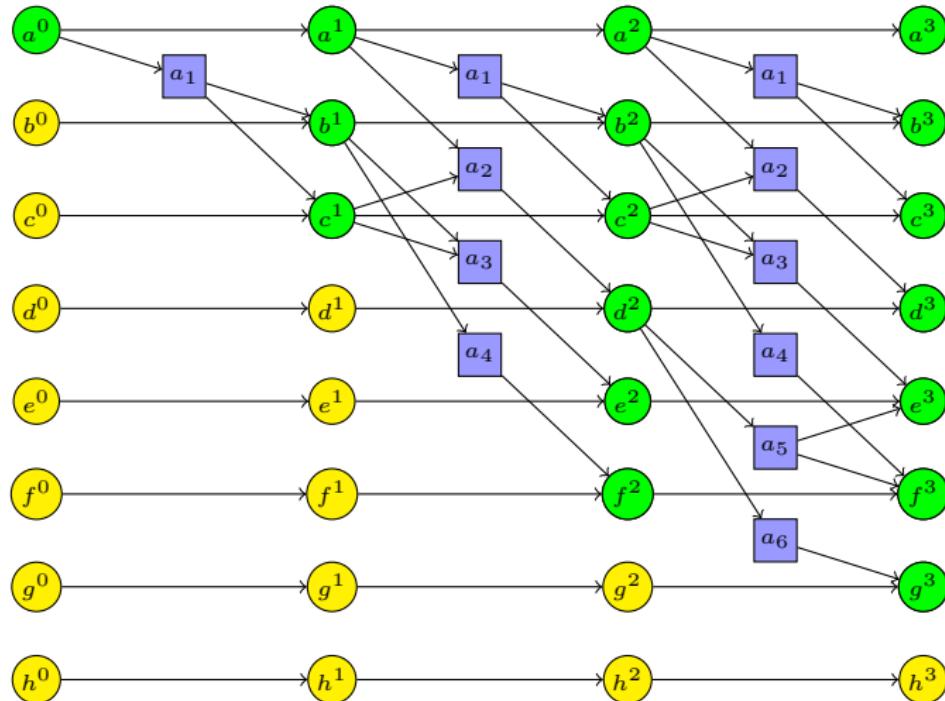
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

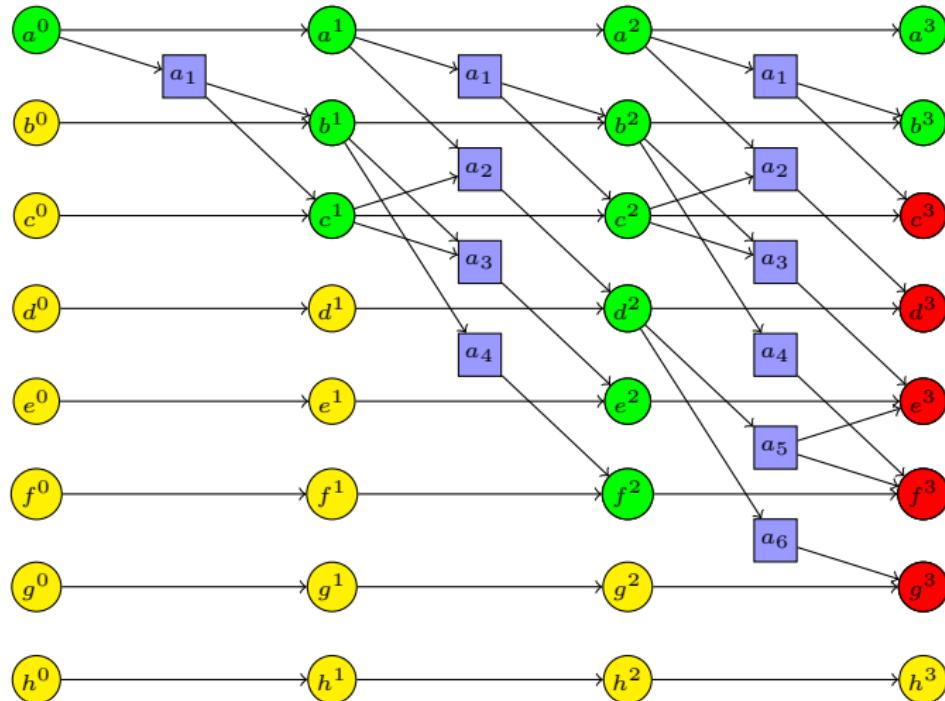
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

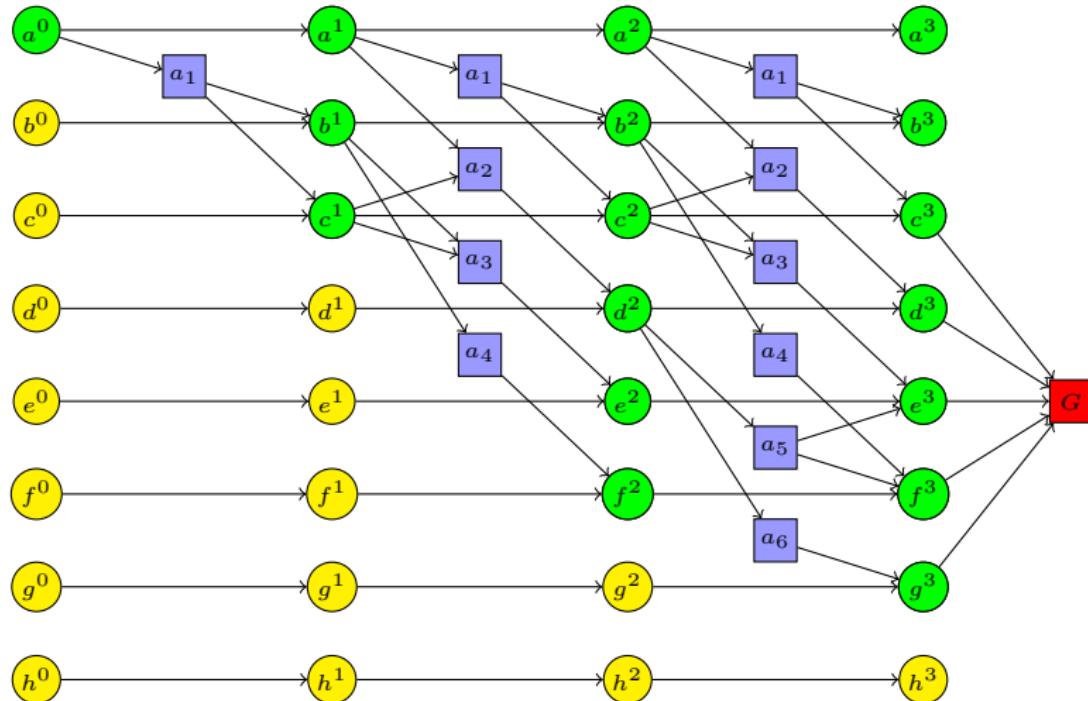
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Generic relaxed planning graph heuristics

## Computing heuristics from relaxed planning graphs

```
def generic-rpg-heuristic(<P, I, O, G>, s):
     $\Pi^+ := \langle P, s, O^+, G \rangle$ 
    for k in {0, 1, 2, ...}:
        rpg := RPGk( $\Pi^+$ )
        if  $G \subseteq P_k$ :
            Annotate nodes of rpg.
            if termination criterion is true:
                return heuristic value from annotations
        else if k = |P|:
            return  $\infty$ 
```

- ~ generic template for heuristic functions
- ~ to get concrete heuristic: fill in highlighted parts

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Concrete examples for the generic heuristic

Many planning heuristics fit the generic template:

- max heuristic  $h_{\max}$
- additive heuristic  $h_{\text{add}}$
- FF heuristic  $h_{\text{FF}}$
- ...

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Forward cost heuristics

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

- The simplest relaxed planning graph heuristics are **forward cost heuristics**.
- Examples:  $h_{\max}$ ,  $h_{\text{add}}$
- Here, node annotations are **cost values** (natural numbers).
- The cost of a node estimates how expensive (in terms of required operators) it is to make this node true.

↙ ייקי יזנוגה : כנה פלאה יזרויה זיק נאוי .  
↙ ייקי יזנוגה : כנה פלאה יזרויה זיק נאוי .

# Forward cost heuristics: fitting the template

## Forward cost heuristics

### Computing annotations:

- Propagate cost values bottom-up using a combination rule for action nodes and a combination rule for proposition nodes.
- At **action nodes**, add 1 after applying combination rule.

↗ f<sub>00</sub>  
↓

↑  
P'NICIC

### Termination criterion:

- **stability:** terminate if  $P_k = P_{k-1}$  and cost for each proposition node  $p^k \in P_k$  equals cost for  $p^{k-1} \in P_{k-1}$

### Heuristic value:

- The heuristic value is the cost of the auxiliary goal node.
- Different forward cost heuristics only differ in their choice of combination rules.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h^{\max}$

$h^{\text{add}}$

$h^{\text{FF}}$

Comparison &  
practice

# The max heuristic $h_{\max}$ (again)

Forward cost heuristics: max heuristic  $h_{\max}$

Combination rule for action nodes:

- $\text{cost}(u) = \max(\{\text{cost}(v_1), \dots, \text{cost}(v_k)\})$   
(with  $\max(\emptyset) := 0$ )

Combination rule for proposition nodes:

- $\text{cost}(u) = \min(\{\text{cost}(v_1), \dots, \text{cost}(v_k)\})$

In both cases,  $\{v_1, \dots, v_k\}$  is the set of immediate predecessors of  $u$ .

Intuition:

- **Action rule:** If we have to achieve several preconditions, estimate this by the **most expensive** cost. → !נורא מושך
- **Proposition rule:** If we have a choice how to achieve a proposition, pick the **cheapest** possibility.

Introduction  
to AI

C. Domshlak

Introduction

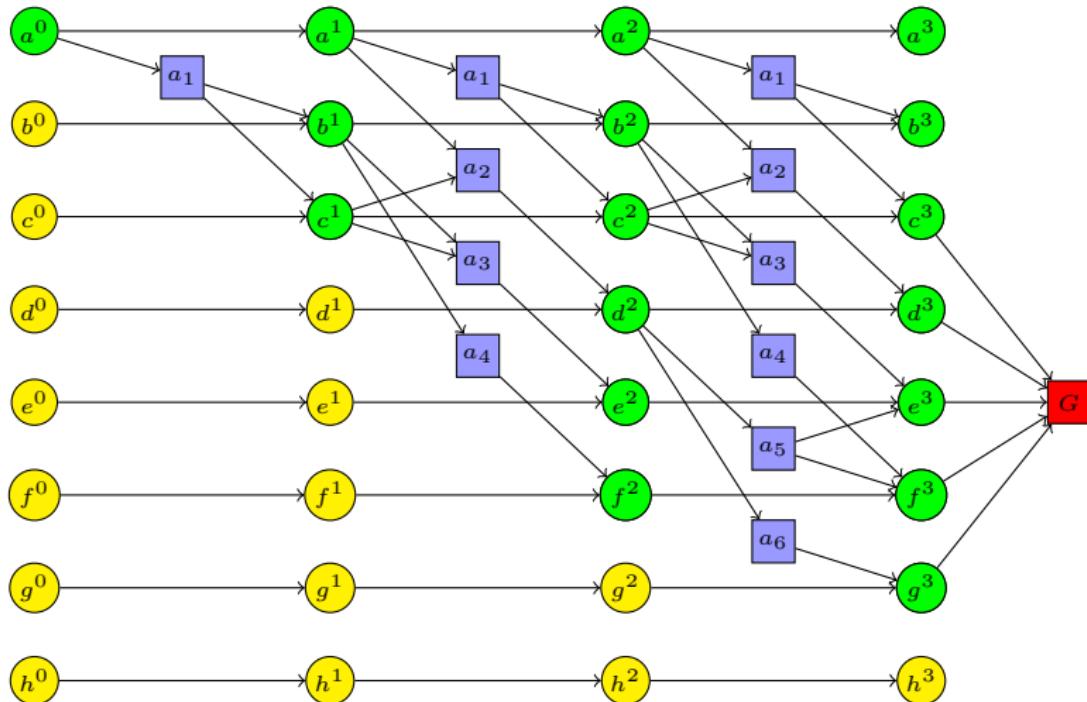
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

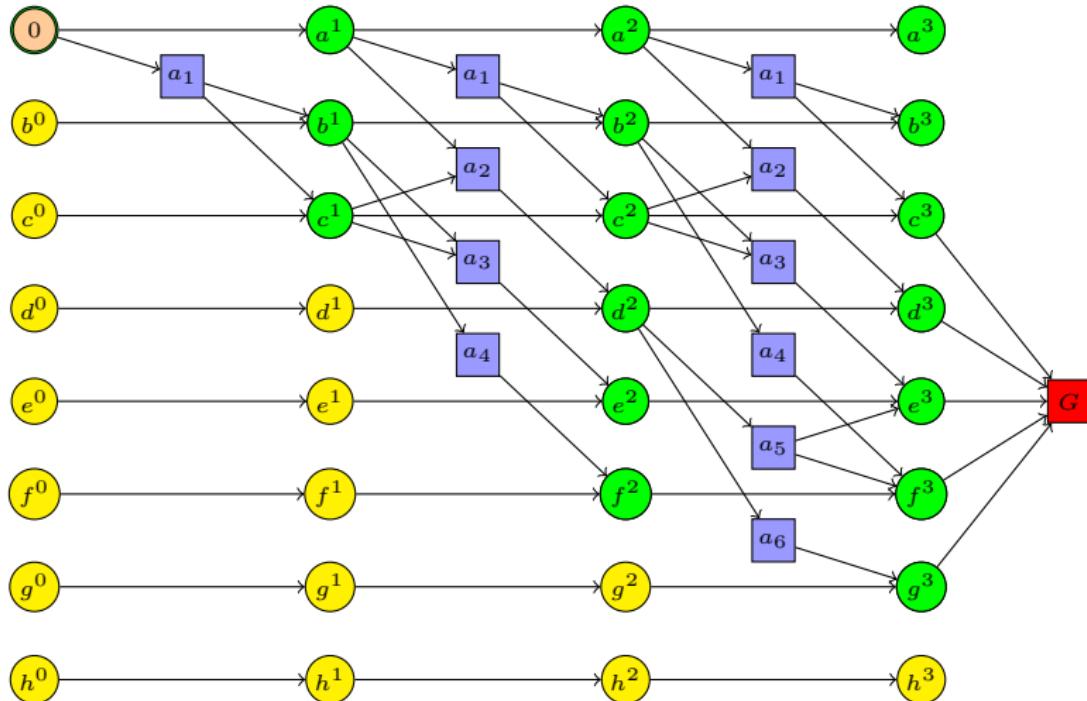
$h_{\max}$

$h_{\text{add}}$

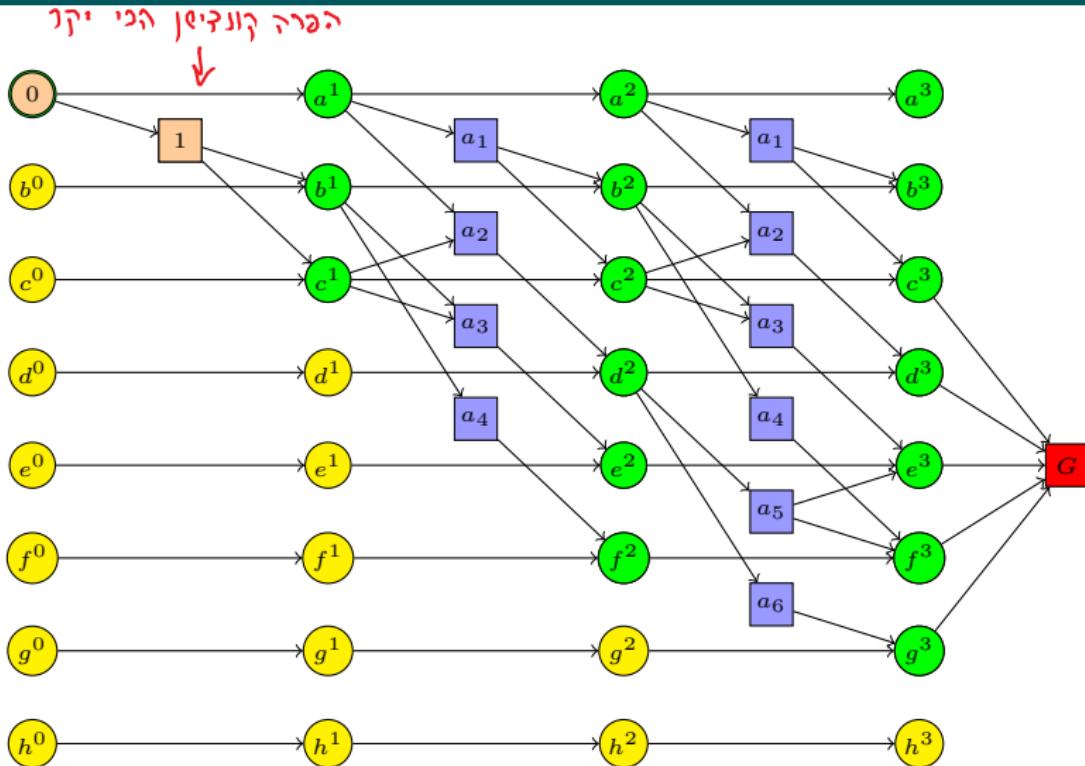
$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

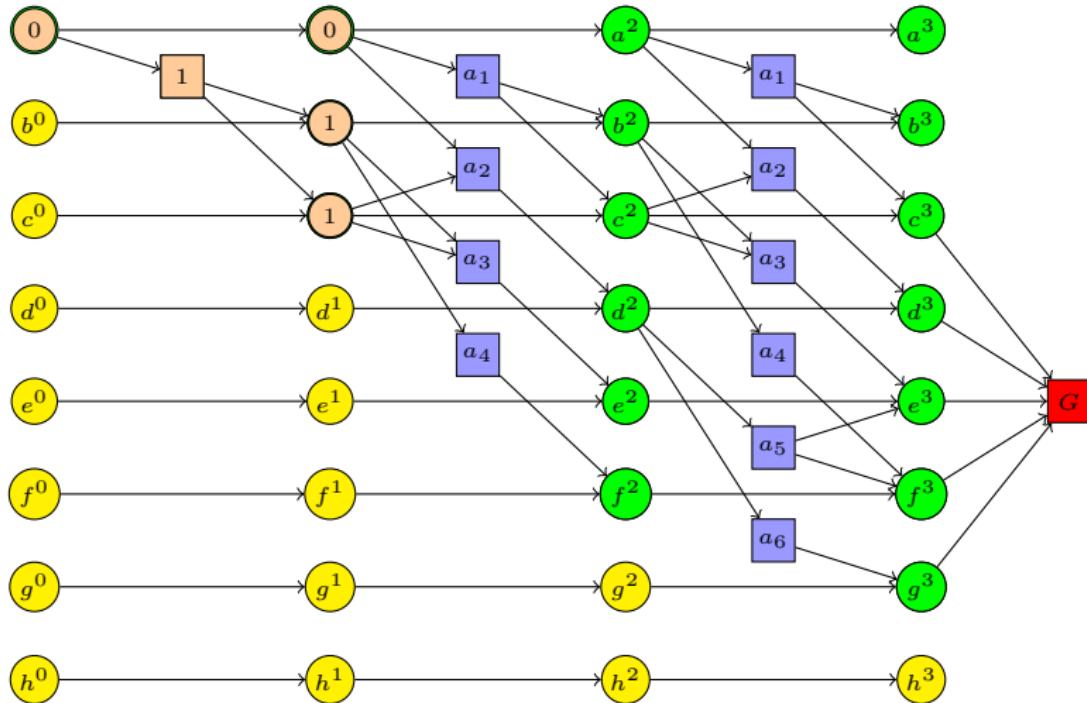
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

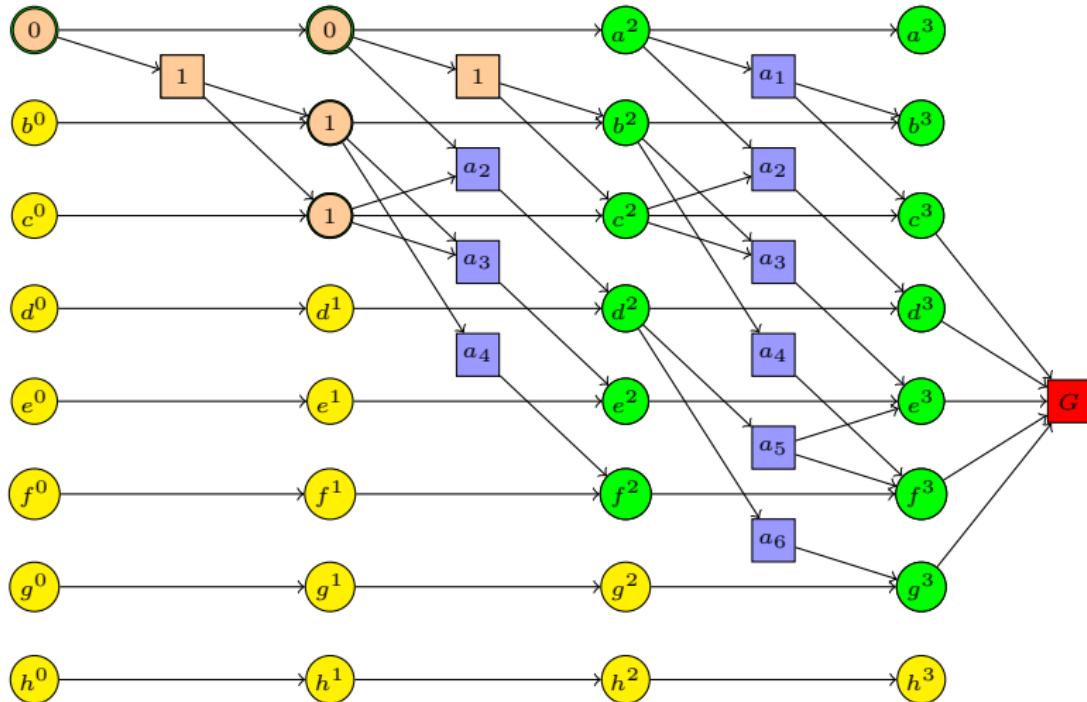
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

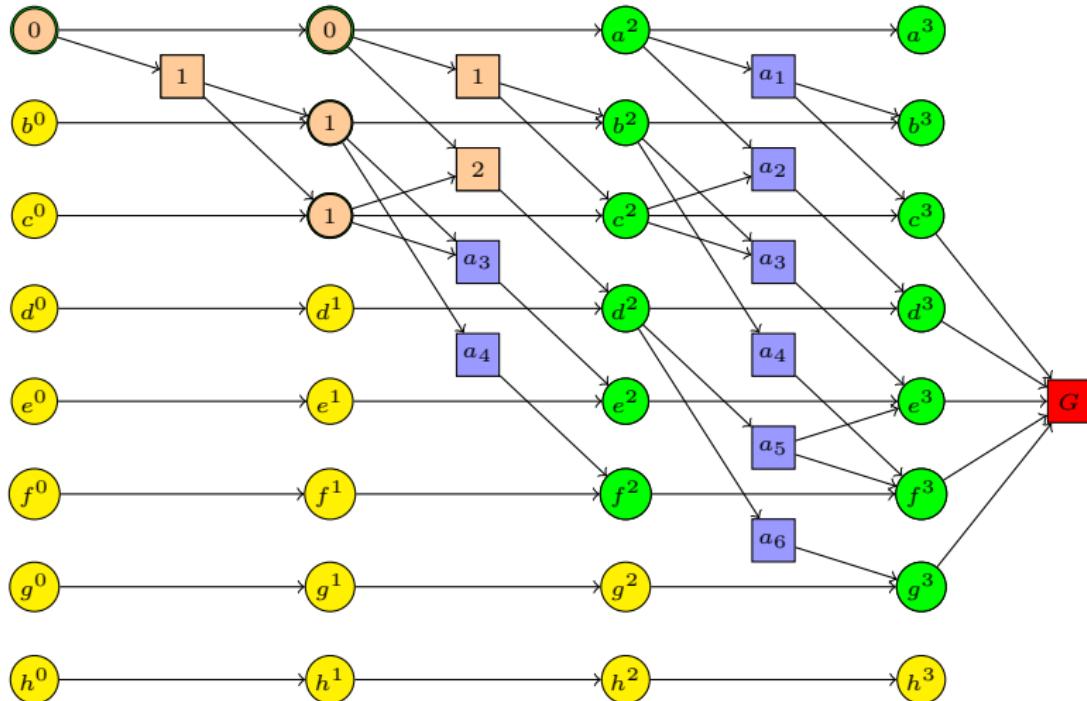
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

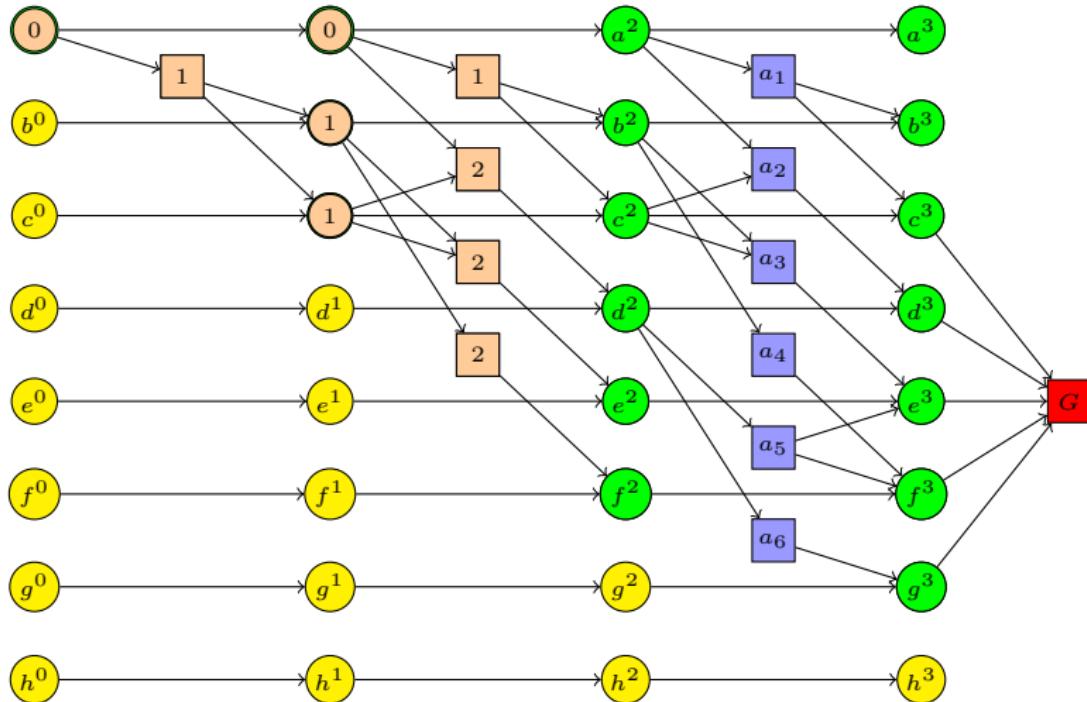
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

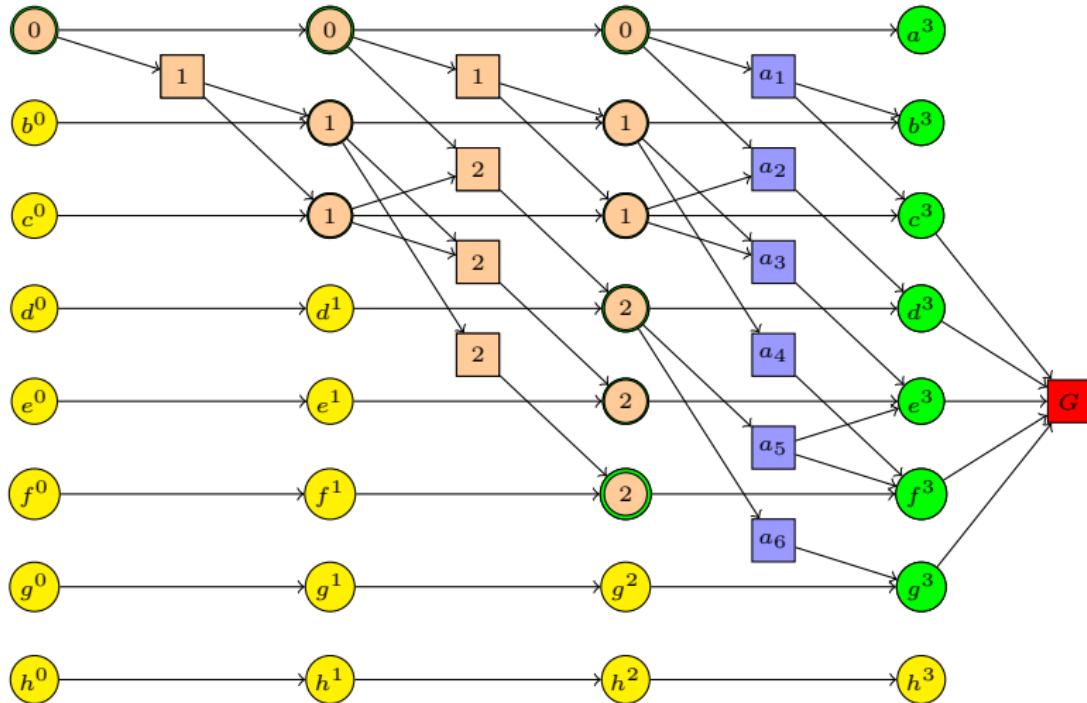
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

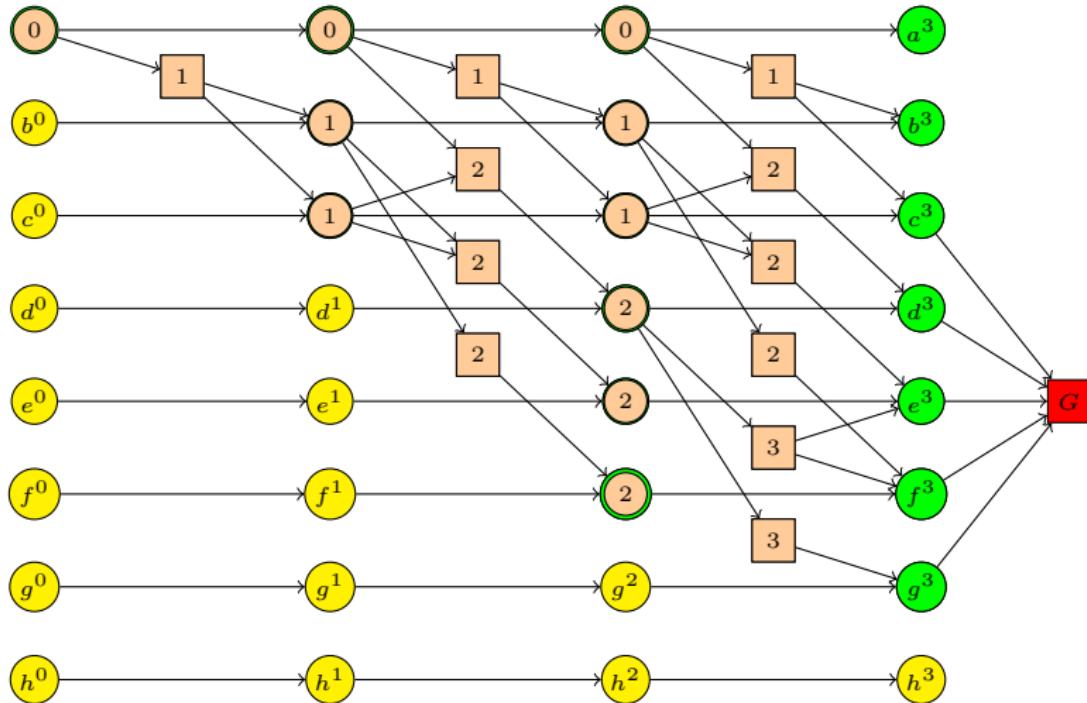
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

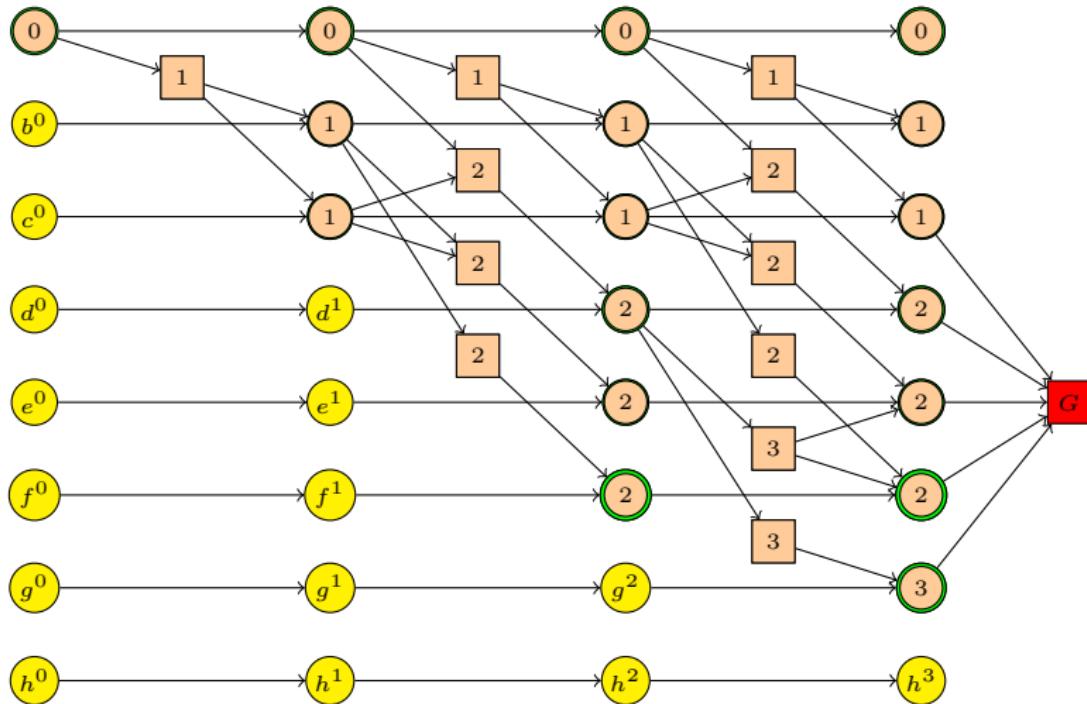
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

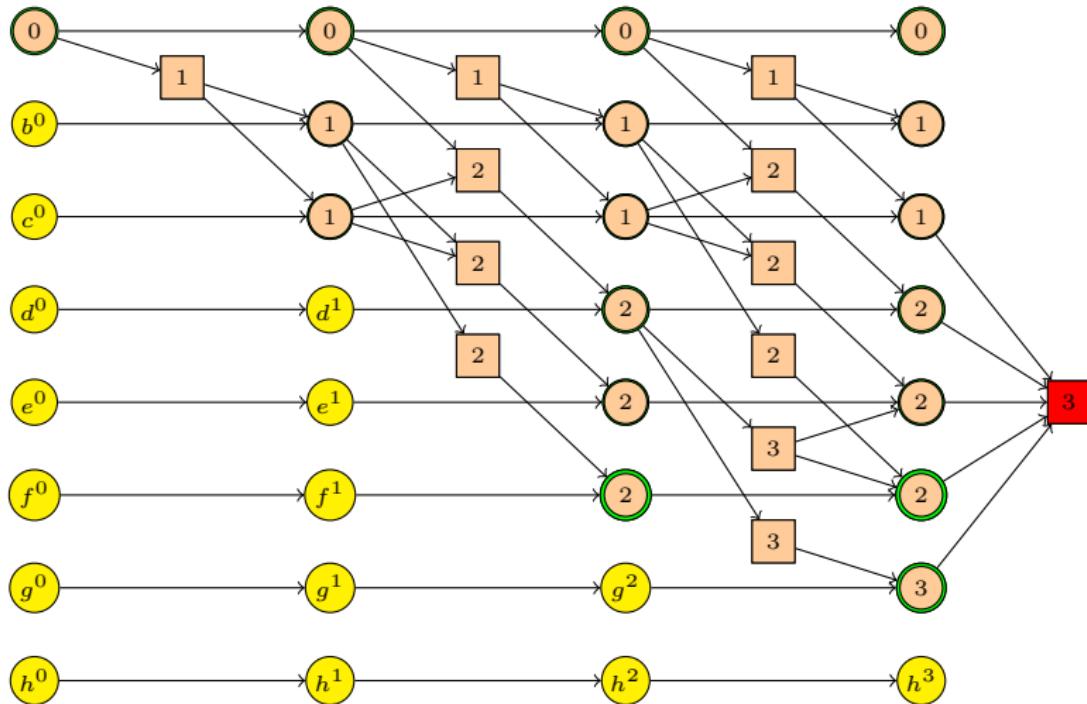
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# The additive heuristic

Forward cost heuristics: additive heuristic  $h_{\text{add}}$

Combination rule for action nodes:

- $\text{cost}(u) = \text{cost}(v_1) + \dots + \text{cost}(v_k)$   
(with  $\sum(\emptyset) := 0$ )

Combination rule for proposition nodes:

- $\text{cost}(u) = \min(\{\text{cost}(v_1), \dots, \text{cost}(v_k)\})$

In both cases,  $\{v_1, \dots, v_k\}$  is the set of immediate predecessors of  $u$ .

Intuition:

- **Action rule:** If we have to achieve several preconditions, estimate this by the cost of achieving **each in isolation**. !שניהם נטול ←
- **Proposition rule:** If we have a choice how to achieve a proposition, pick the **cheapest** possibility.

Introduction  
to AI

C. Domshlak

Introduction

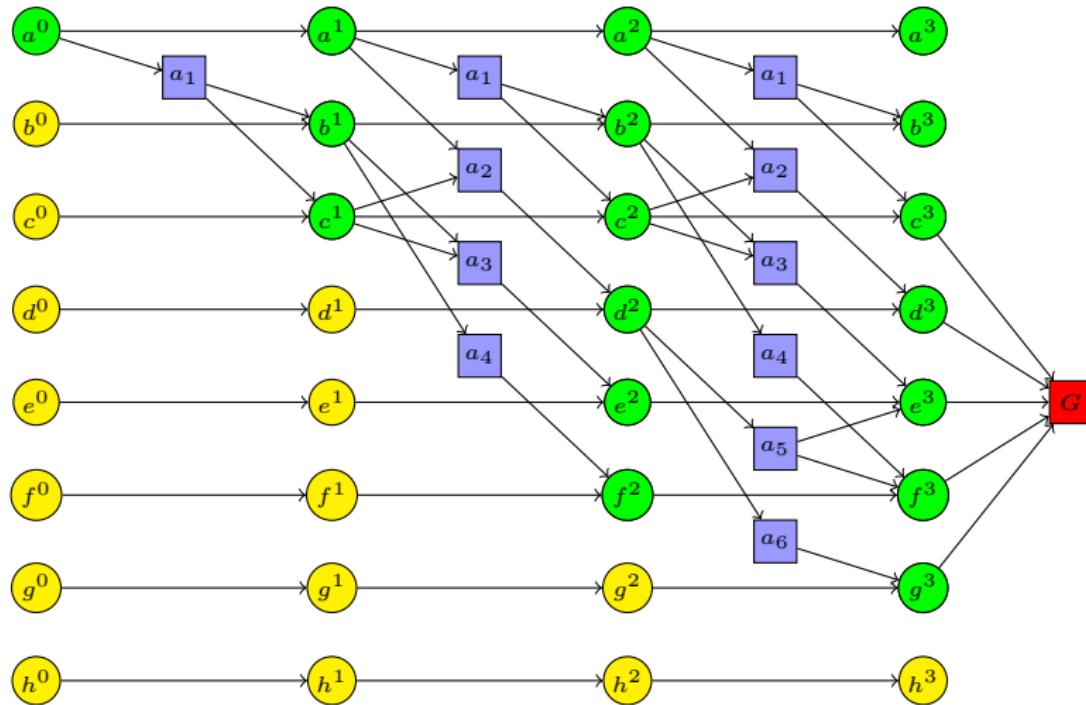
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

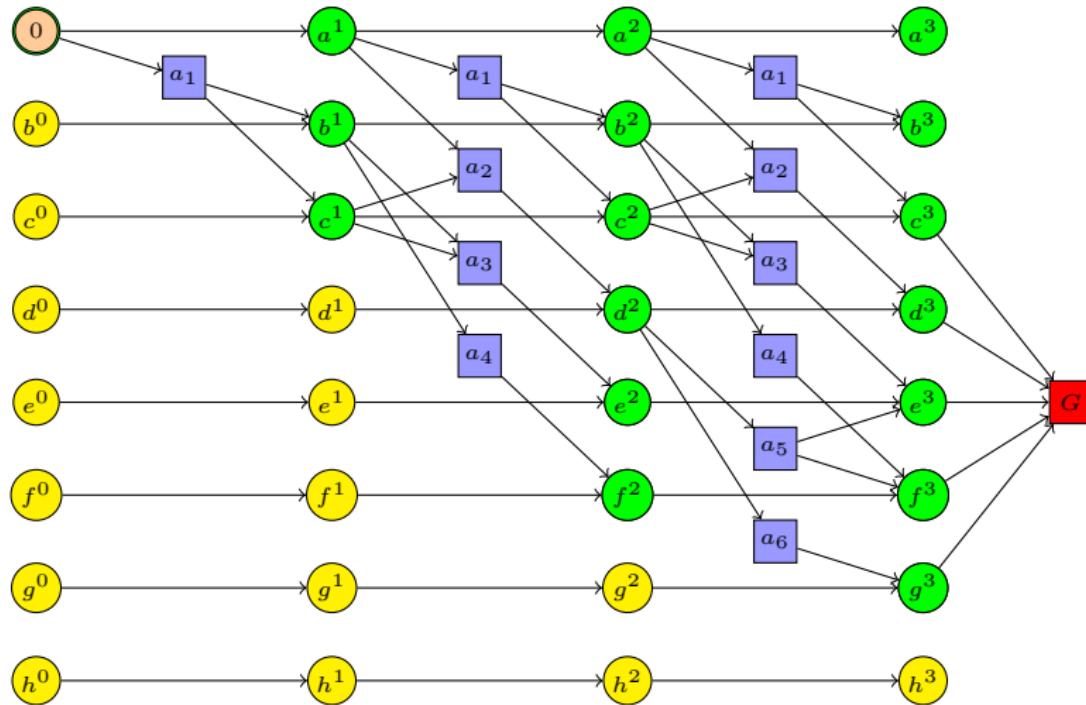
$h^{\max}$

$h_{\text{add}}$

$h^{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

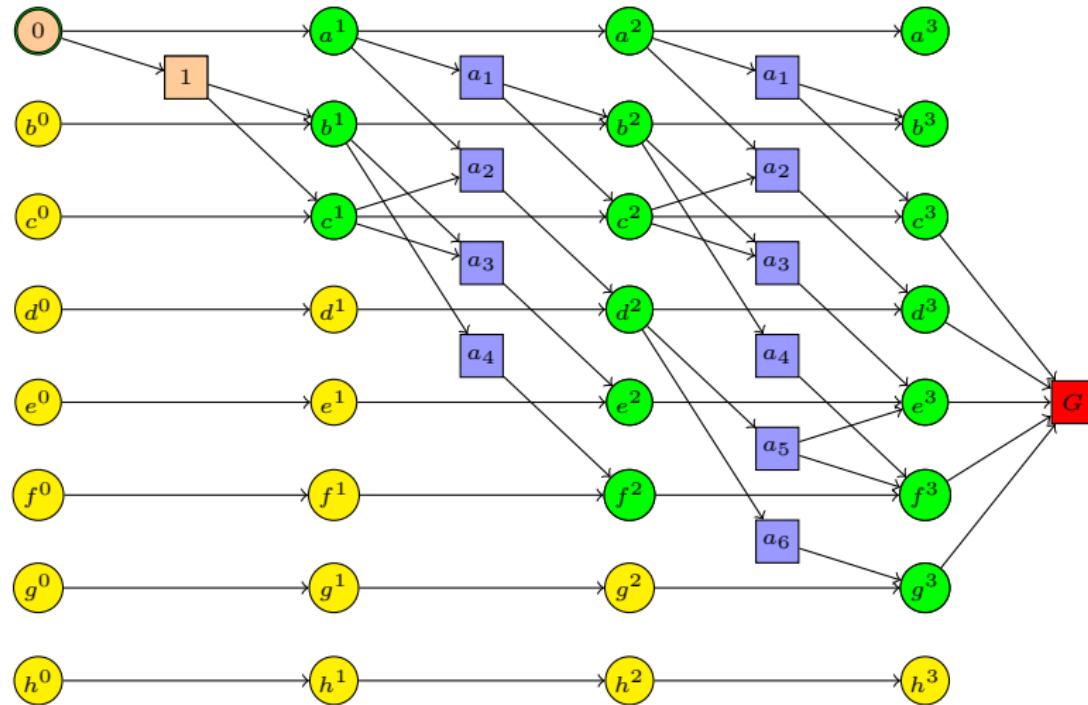
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

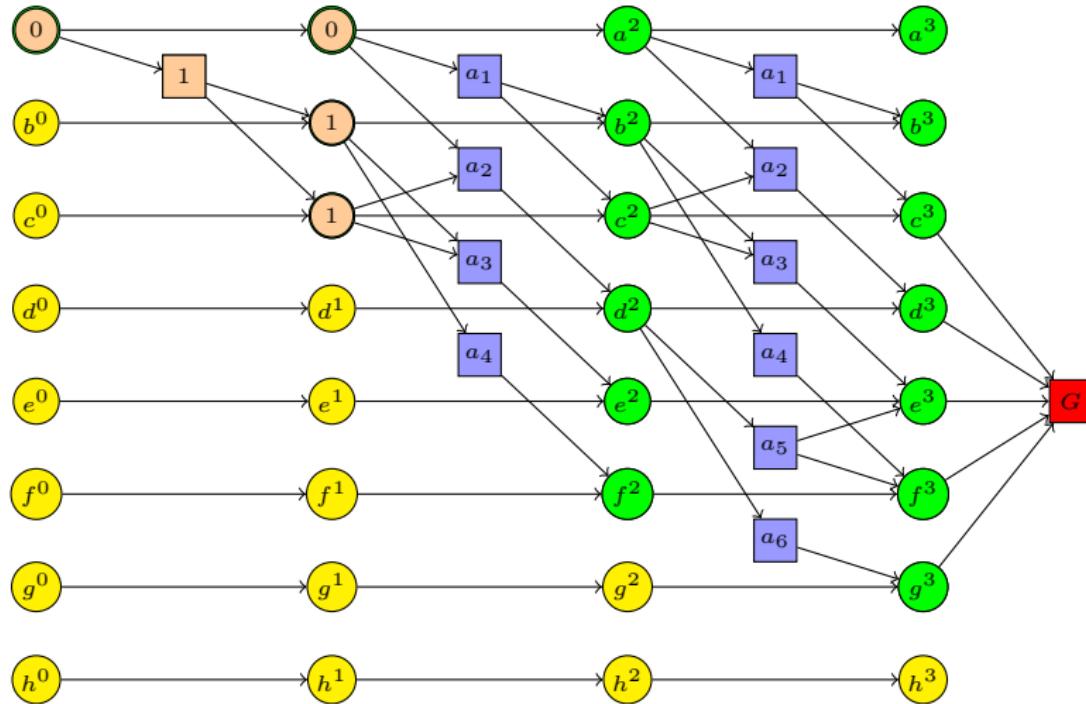
$h^{\max}$

$h_{\text{add}}$

$h^{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

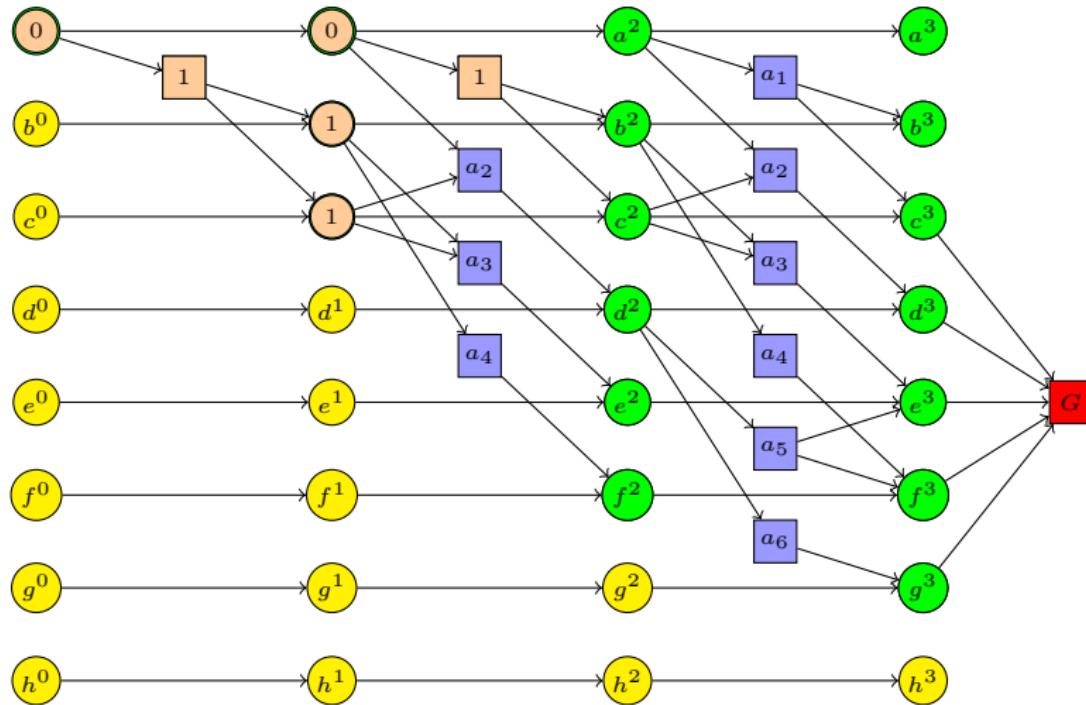
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

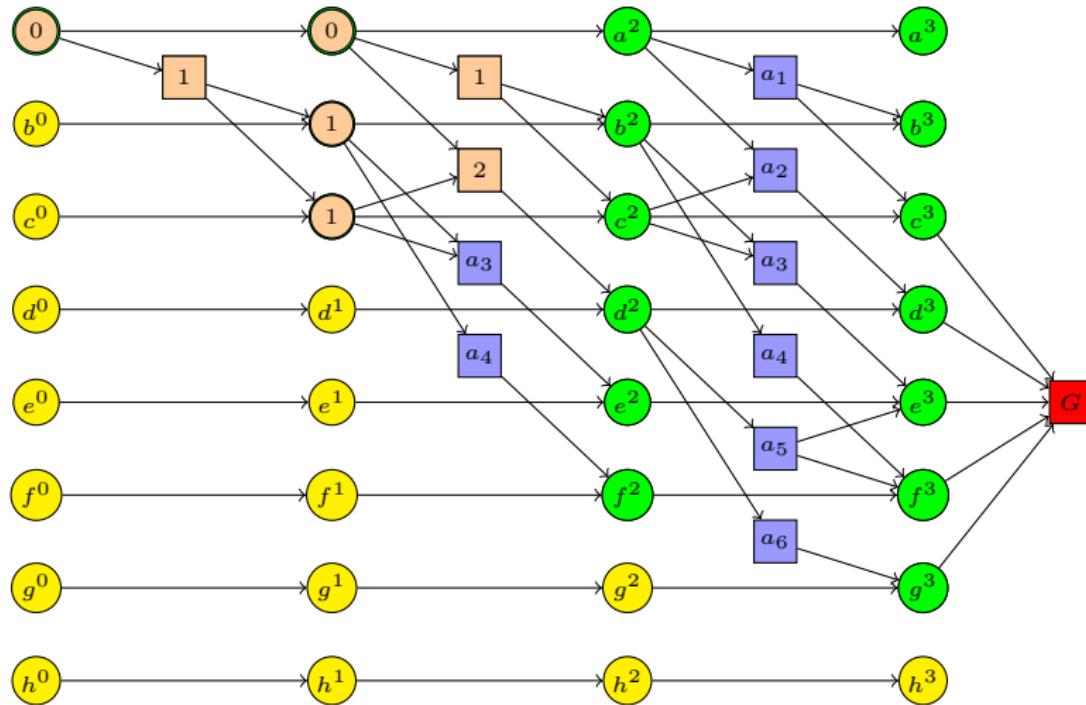
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h^{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

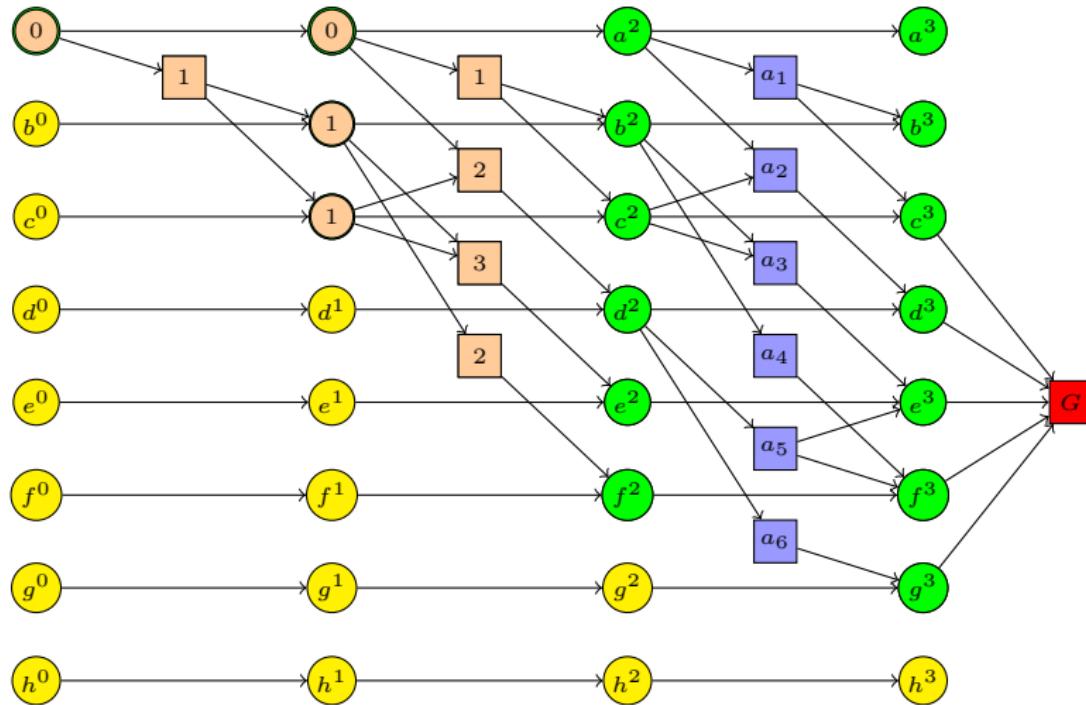
$h^{\max}$

$h_{\text{add}}$

$h^{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

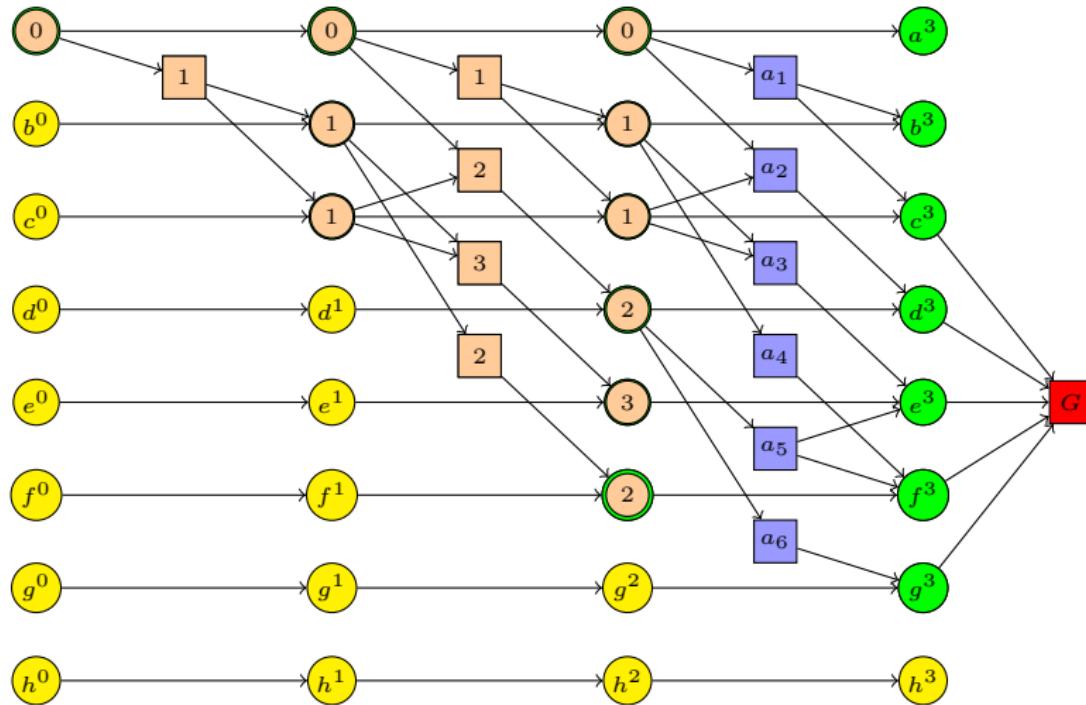
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h^{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

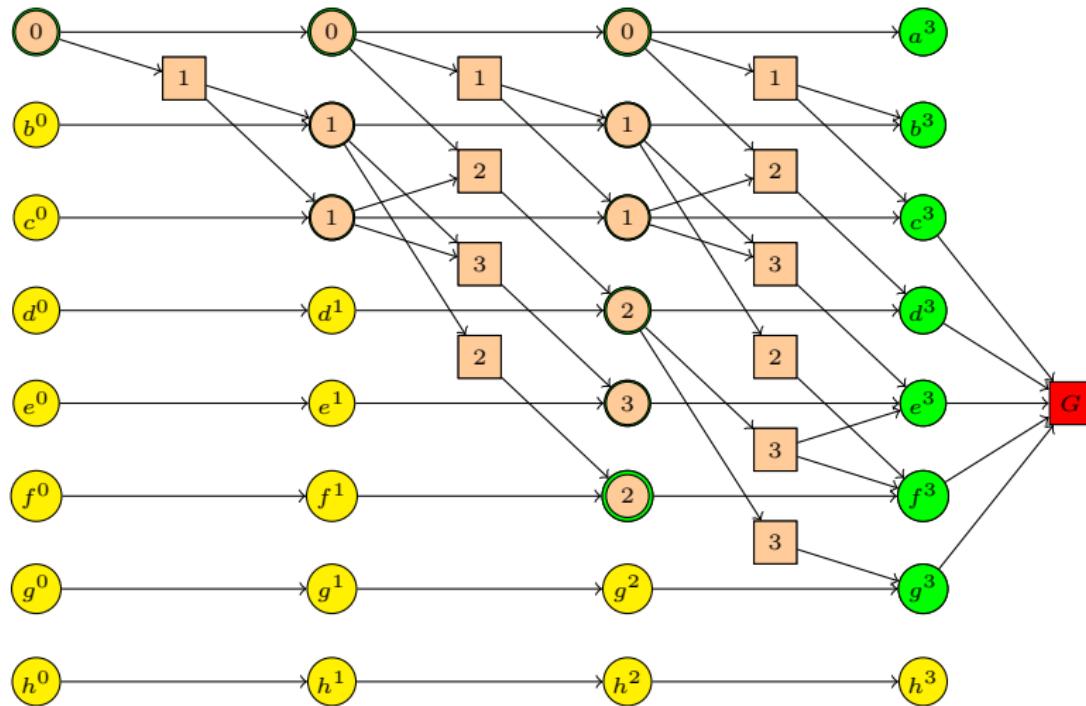
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

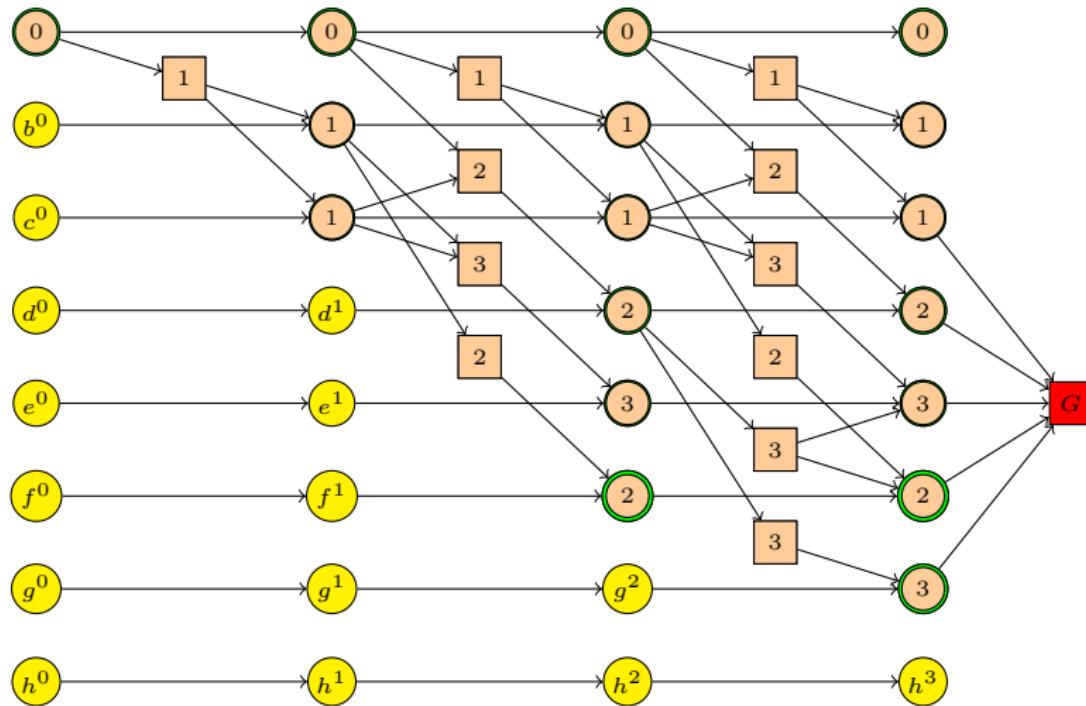
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

## Running example: $h_{\text{add}}$



## Introduction to AI

C. Domshlak

## Obtaining heuristics

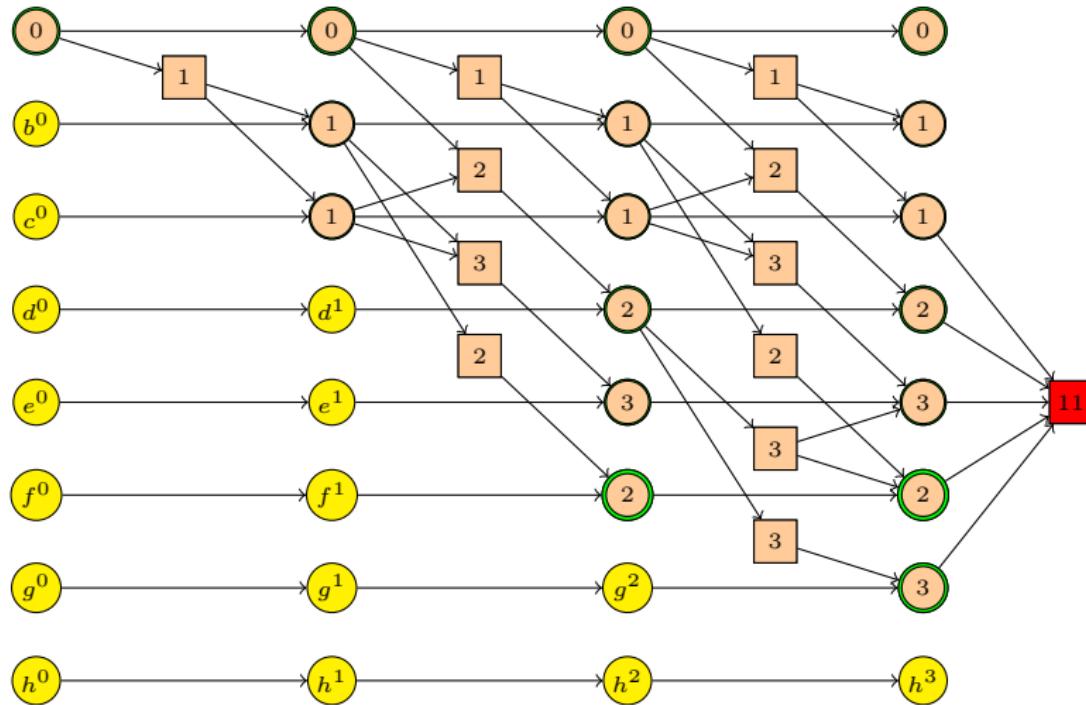
## Concrete Heuristics

$h_{\text{add}}$

hFF

Comp

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Remarks on $h_{\text{add}}$

- $h_{\text{add}}$  is **safe** and **goal-aware**.
- Unlike  $h_{\text{max}}$ ,  $h_{\text{add}}$  is a **very informative** heuristic in many planning domains.
- Q: Intuitively, when it will be informative? ↪
- The price for this is that it is **not admissible** (and hence also **not consistent**), so not suitable for optimal planning.
- In fact, it **almost always** overestimates the  $h^+$  value because it does not take **positive interactions** into account.

הערך המינימלי של  $h_{\text{add}}$  יהיה שווה לערך  $h_{\text{max}}$ .  
ולוק גיילן, נאר טרניר מוכיח מכך.  
 $h_{\text{add}}$  מזמין נזירות הגדלת הערך  $h_{\text{add}}$ .

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\text{max}}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# FF heuristic: fitting the template

fast forward

## The FF heuristic $h_{FF}$

### Computing annotations:

- Annotations are **Boolean values**, computed top-down.

A node is **marked** when its annotation is set to 1 and **unmarked** if it is set to 0. Initially, the goal node is marked, and all other nodes are unmarked.

We say that an action node is **justified** if all its true immediate predecessors are marked, and that a proposition node is **justified** if at least one of its immediate predecessors is marked.

...

רוויאיון ב-  
הACTION ו-  
הPROPOSITION  
.goal ->

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice

# FF heuristic: fitting the template (ctd.)

## The FF heuristic $h_{\text{FF}}$ (ctd.)

Computing annotations:

- ...

Apply these rules until **all marked nodes are justified**:

- ① Mark all immediate predecessors of a marked unjustified ACTION node.
- ② Mark the immediate predecessor of a marked unjustified PROP node with only one immediate predecessor.
- ③ Mark an immediate predecessor of a marked unjustified PROP node connected via an idle arc.
- ④ Mark any immediate predecessor of a marked unjustified PROP node.

The rules are given in priority order: earlier rules are preferred if applicable.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h^{\max}$   
 $h^{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# FF heuristic: fitting the template (ctd.)

## The FF heuristic $h_{\text{FF}}$ (ctd.)

Termination criterion:

- Always terminate at first layer where goal node is true.

Heuristic value:

- The heuristic value is the number of marked action nodes.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

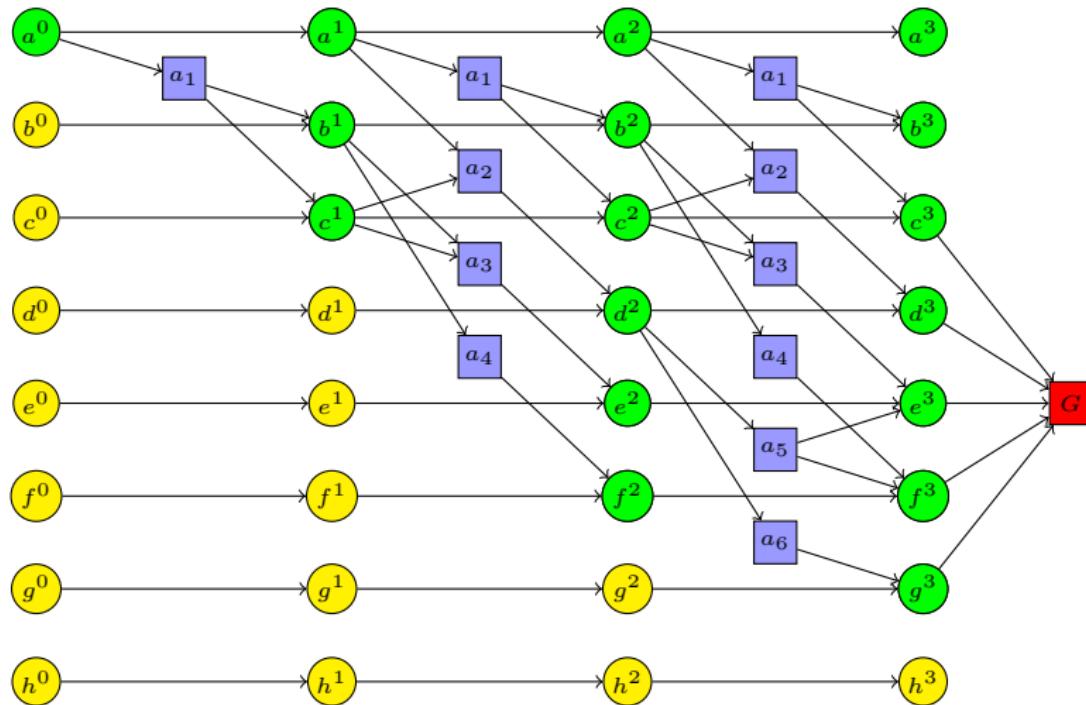
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h^{\max}$   
 $h^{\text{add}}$   
 $h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

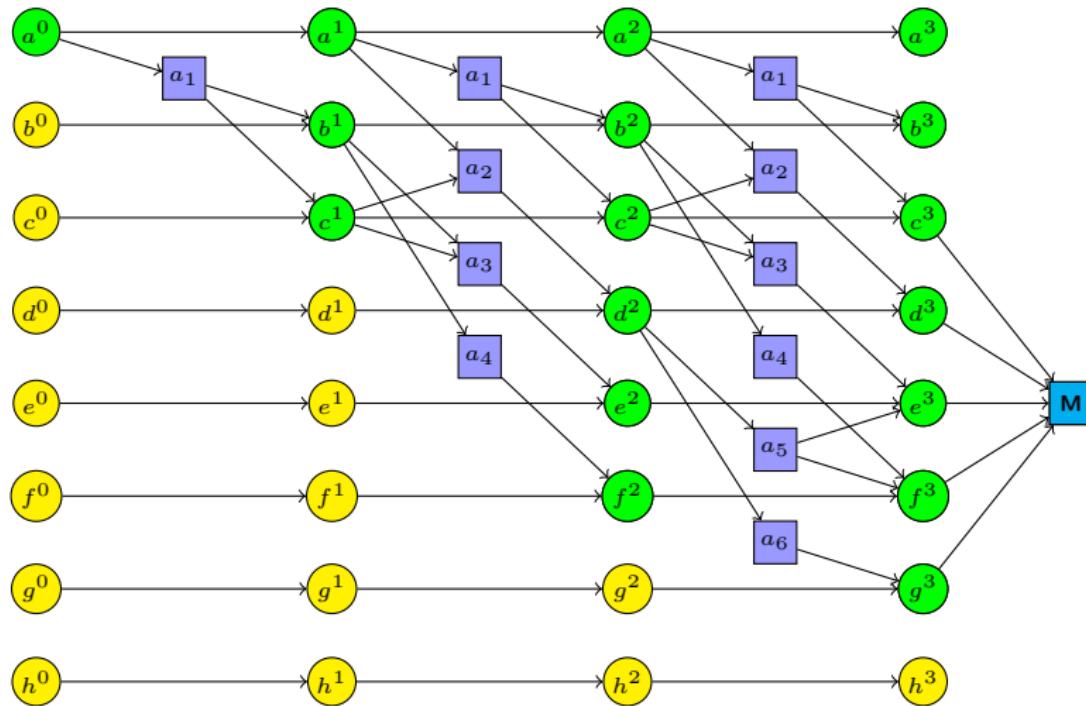
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$   
Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

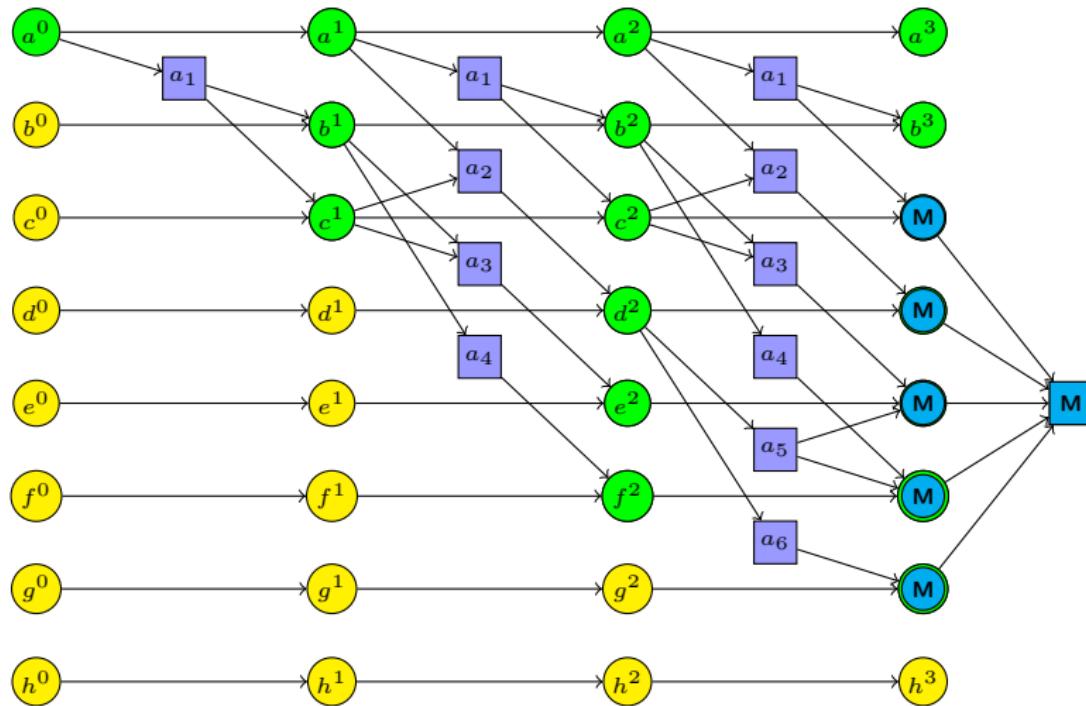
$h_{\max}$

$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

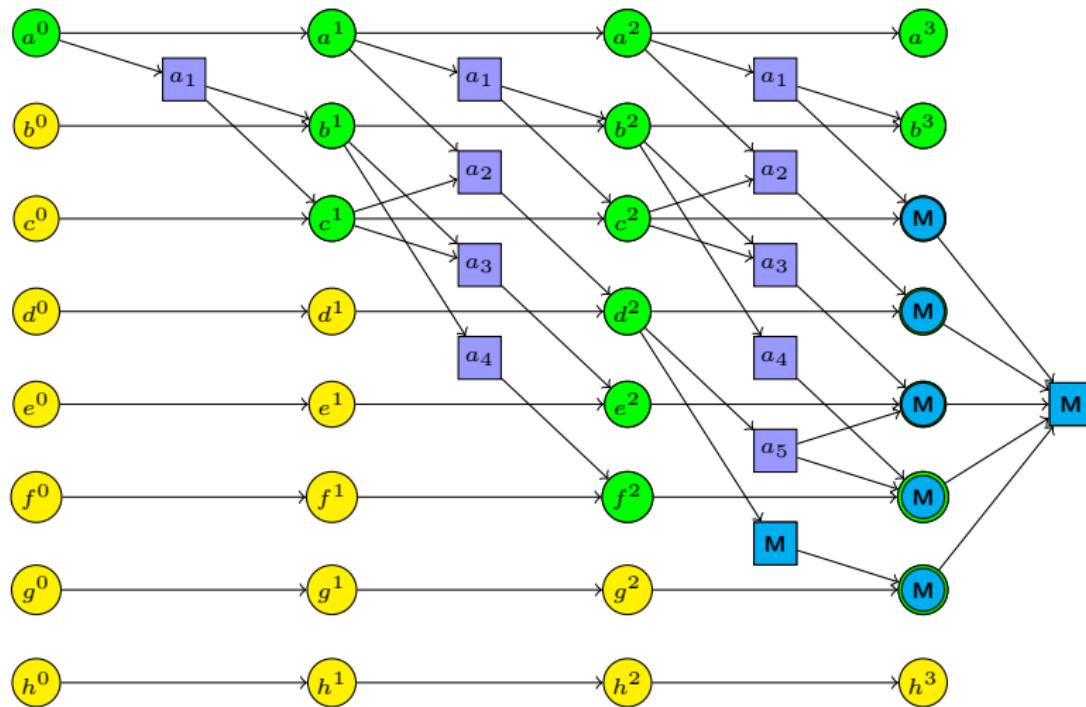
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

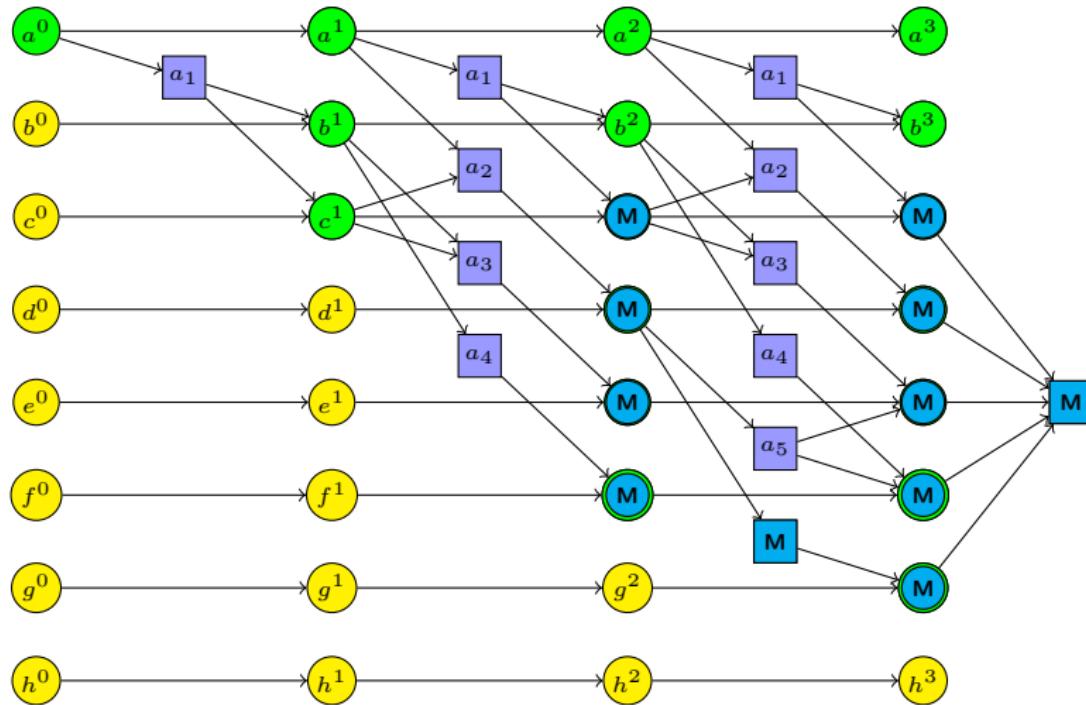
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

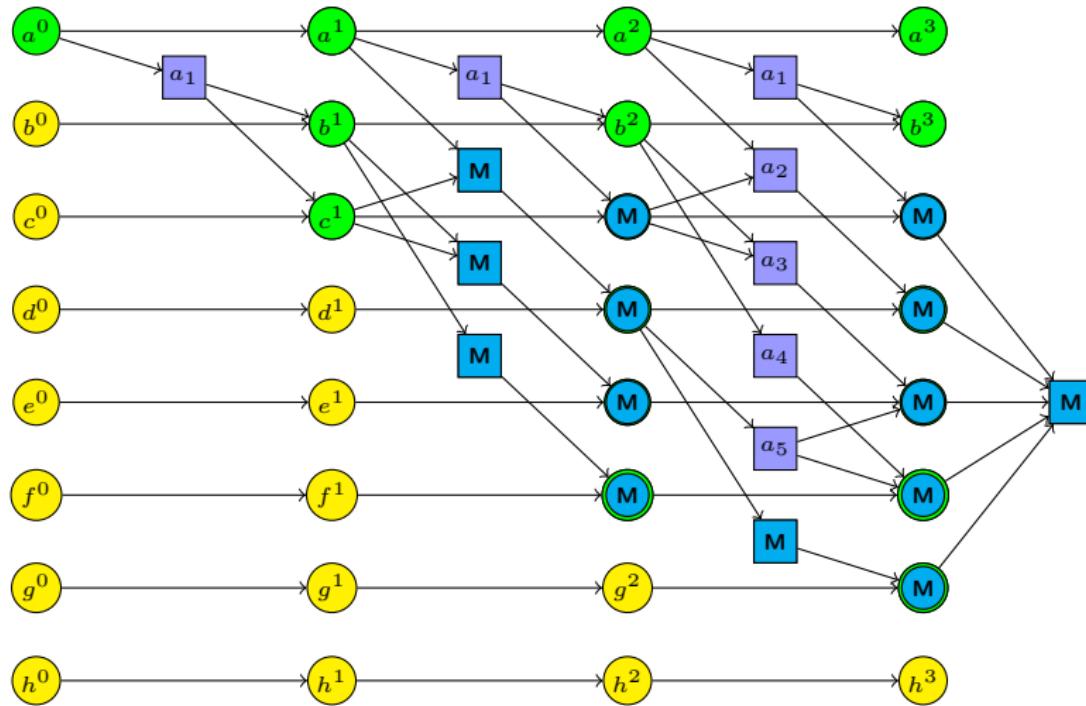
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

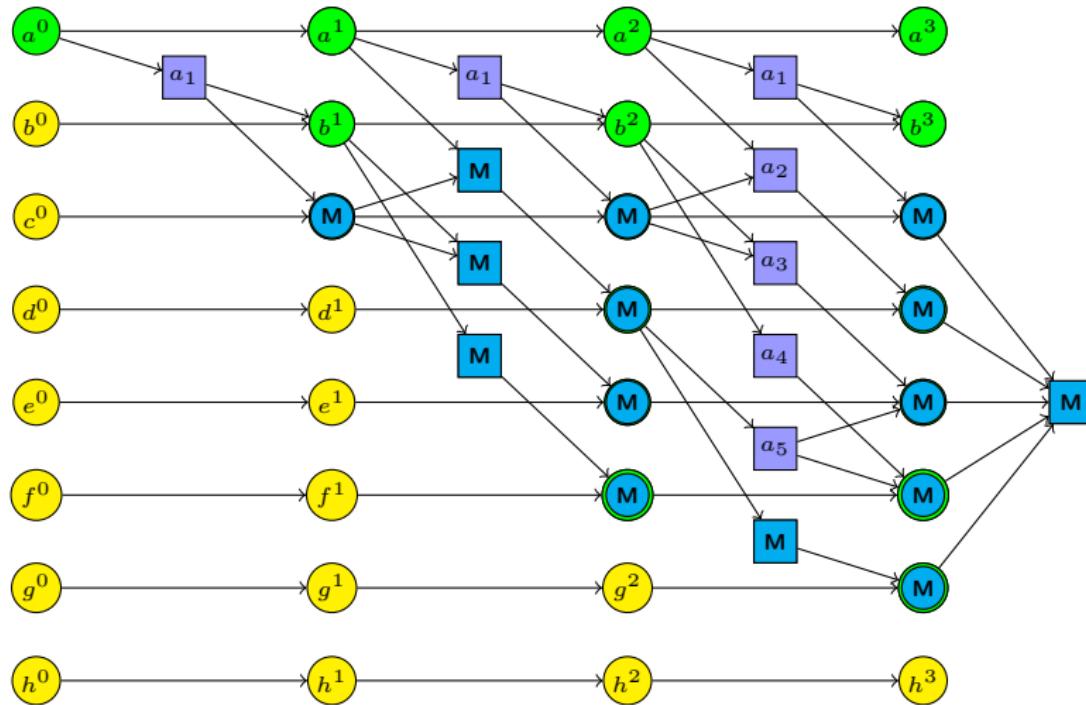
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

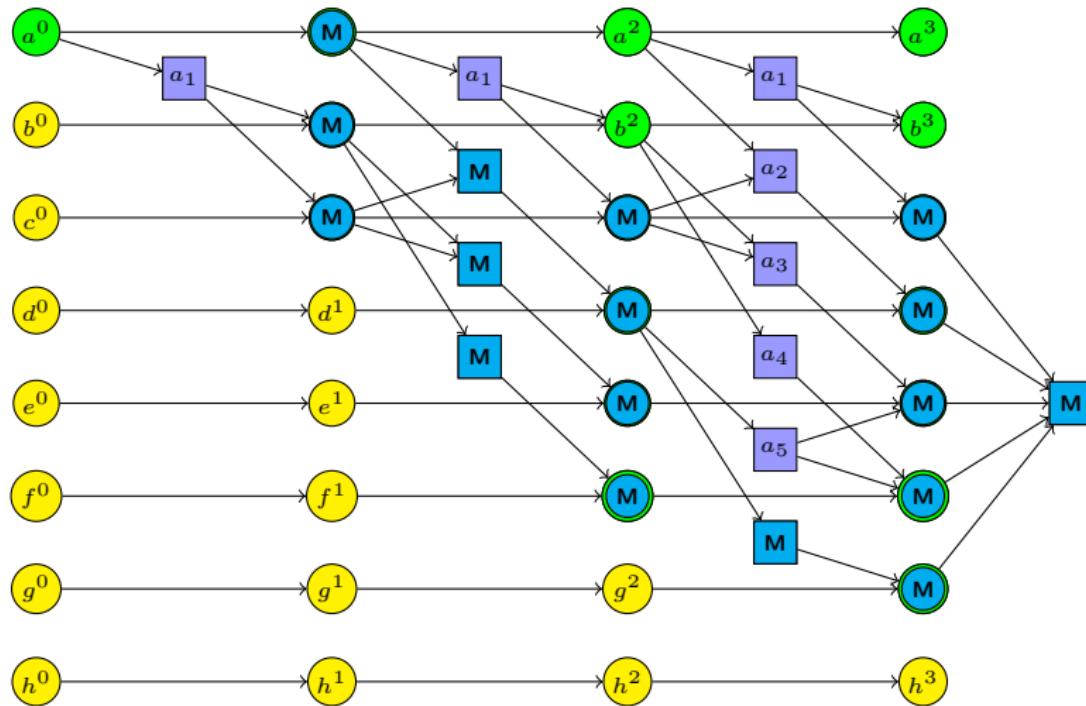
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

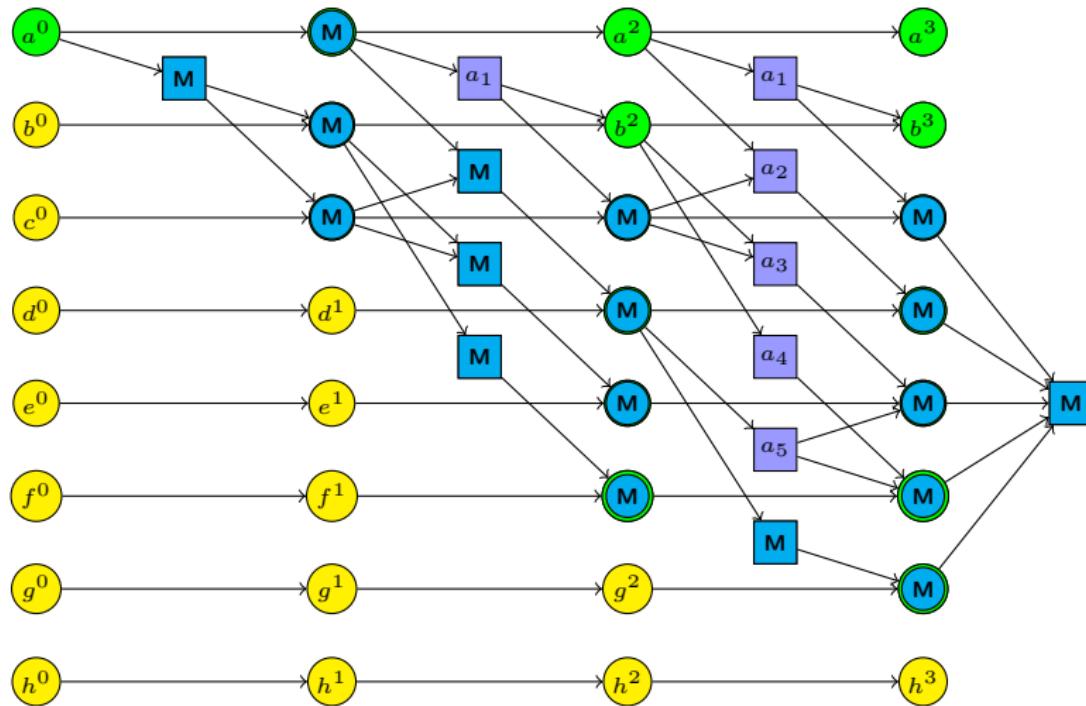
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

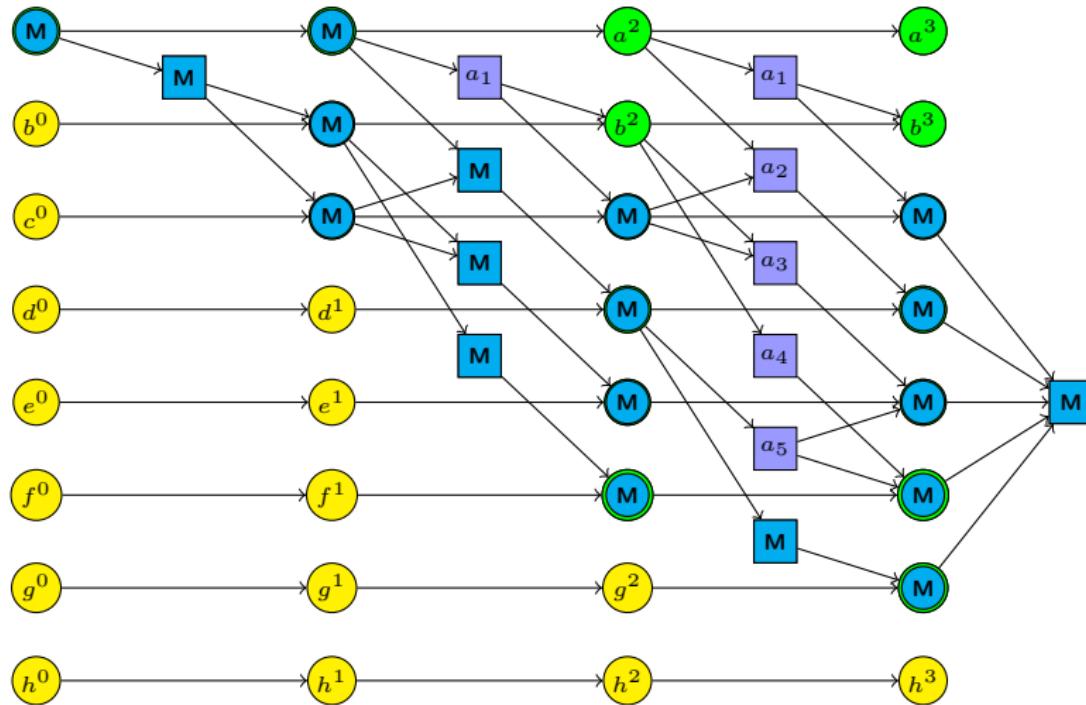
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

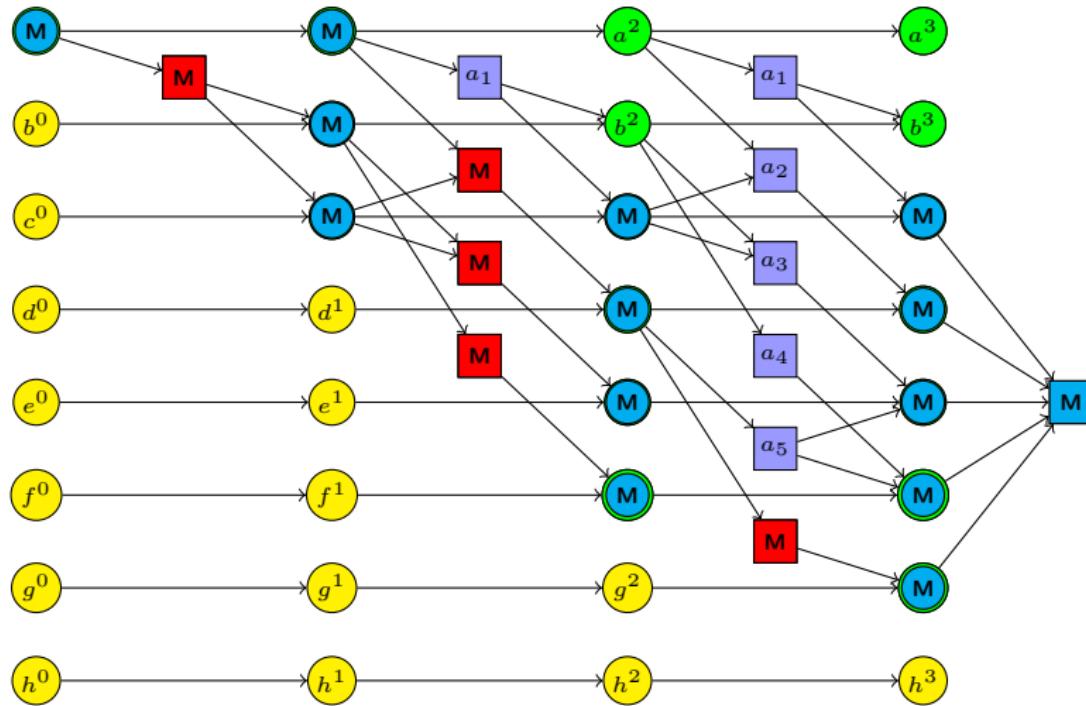
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$   
Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$   
Comparison &  
practice

# Remarks on $h_{\text{FF}}$

- Like  $h_{\text{add}}$ ,  $h_{\text{FF}}$  is **safe** and **goal-aware**, but neither **admissible** nor **consistent**.
- Always more accurate than  $h_{\text{add}}$  with respect to  $h^+$ .
  - Marked actions define a **relaxed plan**.
- $h_{\text{FF}}$  can be computed in **linear time**.
  - The  $h_{\text{FF}}$  value depends on tie-breaking when the marking rules allow several possible choices, so  $h_{\text{FF}}$  is **not well-defined** without specifying the tie-breaking rule.
  - The best implementations of FF use additional rules of thumb to try to reduce the size of the generated relaxed plan.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h^{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Comparison of relaxation heuristics

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

## Relationship between relaxation heuristics

Let  $s$  be a state of planning task  $\langle P, I, O, G \rangle$ . Then:

- $h_{\max}(s) \leq h^+(s) \leq h^*(s)$
- $h_{\max}(s) \leq h^+(s) \leq h_{\text{FF}}(s) \leq h_{\text{add}}(s)$
- $h^*$  and  $h_{\text{FF}}$  are pairwise incomparable
- $h^*$  and  $h_{\text{add}}$  are incomparable

Moreover,  $h^+$ ,  $h_{\max}$ ,  $h_{\text{add}}$ , and  $h_{\text{FF}}$  assign  $\infty$  to the same set of states.

כertain cases הינה יתנו

Note: For **inadmissible** heuristics, dominance is in general neither desirable nor undesirable. For relaxation heuristics, the objective is usually to get as close to  $h^+$  as possible.

כרגעו מטרת  $h^+$  נזק

# Introduction to AI

## Domain-independent planning

Carmel Domshlak

IE&M — Technion

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

# Reminder: Are we solver?

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

## General idea

(Admissible) heuristic functions obtained as (optimal) cost functions of relaxed problems

- OK, but heuristic is **yet another input** to our agent!
- Satisfactory for general solvers?
- Satisfactory in special purpose solvers?

## Towards domain-independent agents

- How to get heuristics **automatically**?
- Can such automatically derived heuristics **dominate** the domain-specific heuristics crafted by hand?

# Model-based Problem Solving

We want planning to take the form of **model-based solving**

Problem  $\Rightarrow$  Language  $\Rightarrow$  **Planner**  $\Rightarrow$  Solution

- ① **models** for defining, classifying, and understanding problems
  - what is a *planning problem*
  - what is a *solution (plan)*, and
  - what is an *optimal solution*
- ② **languages** for representing problems
- ③ **algorithms** for solving them

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification  
State variables  
Tasks  
Action  
Languages

Obtaining  
heuristics

Concrete  
Heuristics

# Succinct representation of transition systems

- More **compact** representation of actions than as relations is often
  - **possible** because of symmetries and other regularities,
  - **unavoidable** because the relations are too big.
- Represent different aspects of the world in terms of different **state variables**.  
  ~ A state is a **valuation of state variables**.
- Represent actions in terms of changes to the state variables.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

State variables

Tasks

Action

Languages

Obtaining  
heuristics

Concrete  
Heuristics

# State variables

- The state of the world is described in terms of a **finite set** of **finite-valued** state variables.

## Example

*hour*:  $\{0, \dots, 23\} = 13$

*minute*:  $\{0, \dots, 59\} = 55$

*location*:  $\{51, 52, 82, 101, 102\} = 101$

*weather*:  $\{\text{sunny}, \text{cloudy}, \text{rainy}\} = \text{cloudy}$

*holiday*:  $\{\text{T}, \text{F}\} = \text{F}$

- Actions change the values of the state variables.

הפעולות מוחזקות במשת�לים  
במקרה של מושג אחד או יותר

Introduction  
to AI

C. Domshlak

Introduction

Problem

specification

State variables

Tasks

Action

Languages

Obtaining  
heuristics

Concrete  
Heuristics

# Blocks world with state variables

Introduction  
to AI

C. Domshlak

Introduction

Problem

specification

State variables

Tasks

Action

Languages

Obtaining  
heuristics

Concrete  
Heuristics

State variables:

*location-of-A*: {B, C, table}

*location-of-B*: {A, C, table}

*location-of-C*: {A, B, table}

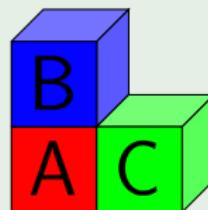
Example

תא ני תרבותה לְפָנֶיךָ אֵין  
.S P3NP ↓

$s(\text{location-of-}A) = \text{table}$

$s(\text{location-of-}B) = A$

$s(\text{location-of-}C) = \text{table}$



Not all valuations correspond to an intended blocks world state, e.g.  $s$  such that  $s(\text{location-of-}A) = B$  and  $s(\text{location-of-}B) = A$ .

# Blocks world with Boolean state variables

## Example

‘עכטן דיאז’

$$s(A\text{-on-}B) = 0$$

$$s(A\text{-on-}C) = 0$$

$$s(A\text{-on-table}) = 1$$

$$s(B\text{-on-}A) = 1$$

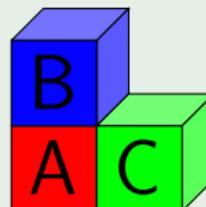
$$s(B\text{-on-}C) = 0$$

$$s(B\text{-on-table}) = 0$$

$$s(C\text{-on-}A) = 0$$

$$s(C\text{-on-}B) = 0$$

$$s(C\text{-on-table}) = 1$$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

State variables

Tasks

Action  
Languages

Obtaining  
heuristics

Concrete  
Heuristics

# Deterministic planning tasks

Introduction  
to AI

C. Domshlak

## Definition (deterministic planning task)

A **deterministic planning task** is a 4-tuple  $\Pi = \langle V, I, A, G \rangle$

- $V$  is a finite set of **state variables**,
- $I$  is an **initial state** over  $V$ ,
- $A$  is a finite set of **actions** over  $V$ , and
- $G$  is a **formula** over  $V$  describing the **goal states**.
  - in what follows: partial assignment to  $V$

"B on C == 1" : few ↘

~ Mapping planning tasks to transition systems

Introduction

Problem  
specification

State variables

Tasks

Action  
Languages

Obtaining  
heuristics

Concrete  
Heuristics

# Planning Languages

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification  
State variables  
Tasks  
Action  
Languages

Obtaining  
heuristics

Concrete  
Heuristics

## Key issue

Models represented **implicitly** in a **declarative language**

## Play two roles

- **specification**: concise model description
- **computation**: reveal useful info about problem's *structure*

# The STRIPS language

Introduction  
to AI

C. Domshlak

Introduction

Problem specification

State variables

Tasks

Action Languages

Obtaining heuristics

Concrete Heuristics

A problem in **STRIPS** is a tuple  $\langle P, A, I, G \rangle$

- $P$  stands for a finite set of **atoms** (boolean vars)
- $I \subseteq P$  stands for **initial situation**
- $G \subseteq P$  stands for **goal situation** → *Goal State*
- $A$  is a finite set of **actions**  $a$  specified via *Preconditions*, *Add effects*, and *Delete effects*, all subsets of  $P$
- States are **collections** of atoms *.PICKUP* *IN* *ON*
- An action  $a$  is applicable in a state  $s$  iff  $\text{pre}(a) \subseteq s$
- Applying an applicable action  $a$  at  $s$  results in  $s' = (s \setminus \text{del}(a)) \cup \text{add}(a)$

*.T Pre p.NICU DOOR IN ON & C B*

# Why STRIPS is interesting

Pre(a) - מושג יסוד בSTRIPS  
הנורמליזציה של  
הACTION מושג יסוד  
הACTION מושג יסוד.

add(a) - מושג יסוד בSTRIPS  
del(a) - מושג יסוד בSTRIPS

- STRIPS actions are **particularly simple**, yet expressive enough to capture general planning problems.
- In particular, STRIPS planning is **no easier** than general planning problems.
- Many algorithms in the planning literature are **easier to present in terms of STRIPS**.

.STRIPS - הפשטה מושג יסוד

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

State variables

Tasks

Action  
Languages

Obtaining  
heuristics

Concrete  
Heuristics

# A simple heuristic for deterministic planning

STRIPS סטריפס

STRIPS (Fikes & Nilsson, 1971) used the number of state variables that differ in current state  $s$  and a STRIPS goal  
 $G = \{g_1, \dots, g_k\}$ :

$$h(s) := |G \setminus s|.$$

Intuition: more true goal literals  $\rightsquigarrow$  closer to the goal

$\rightsquigarrow$  STRIPS heuristic (properties?)

- goal aware ✓

- admissible X

הנ"ט מבחן צפויותי י"

הנ"ט 2 כפאות נור

הנ"ט 1 אהכטן.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Concrete  
Heuristics

# Criticism of the STRIPS heuristic

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics  
The relaxation  
lemma  
Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

## What is wrong with the STRIPS heuristic?

- quite **uninformative**:  
the range of heuristic values in a given task is small;  
typically, most successors have the same estimate
  - very sensitive to **reformulation**:  
can easily transform any planning task into an equivalent  
one where  $h(s) = 1$  for all non-goal states (how?)
  - ignores almost all **problem structure**:  
heuristic value does not depend on the set of actions!
- ~ need a better, principled way of coming up with heuristics

now 4.015  
as per your  
KS the gen 23N

# Coming up with heuristics in a principled way

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristics  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

## General procedure for obtaining a heuristic

Solve an easier version of the problem.

Two common methods:

- **relaxation:** consider **less constrained** version of the problem
- **abstraction:** consider **smaller** version of real problem

Both have been very successfully applied in planning  
(separately and *together*).

We consider **relaxation**.

# Relaxing a problem

How do we relax a problem?

## Example (Route planning for a road network)

The road network is formalized as a weighted graph over points in the Euclidean plane. The weight of an edge is the **road distance** between two locations.

A relaxation **drops constraints** of the original problem.

## Example (Relaxation for route planning)

Use the **Euclidean distance**  $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$  as a heuristic for the road distance between  $(x_1, x_2)$  and  $(y_1, y_2)$ . This is a **lower bound** on the road distance ( $\leadsto$  admissible).

$\leadsto$  We drop the constraint of having to travel on roads.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Relaxations for planning

- Relaxation is a general technique for heuristic design:
  - Straight-line heuristic (route planning): Ignore the fact that one must stay on roads.
  - Manhattan heuristic (15-puzzle): Ignore the fact that one cannot move through occupied tiles.
- We want to apply the idea of relaxations to planning.
- Informally, we want to ignore **bad side effects** of applying actions.

## Example (8-puzzle)

If we move a tile from  $x$  to  $y$ , then the **good effect** is (in particular) that  $x$  is now free.

The **bad effect** is that  $y$  is not free anymore, preventing us from moving tiles through it.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Relaxed planning tasks: idea

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Concrete  
Heuristics

In STRIPS, good and bad effects are easy to distinguish:

- Effects that make atoms true are good  
(**add effects**).
- Effects that make atoms false are bad  
(**delete effects**).

Idea for the heuristic: **Ignore all delete effects.**

# Relaxed planning tasks: idea

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Concrete  
Heuristics

In STRIPS, good and bad effects are easy to distinguish:

- Effects that make atoms true are good  
(**add effects**).  $\text{add}(a)$
- Effects that make atoms false are bad  
(**delete effects**).  $\text{del}(a)$

Idea for the heuristic: **Ignore all delete effects.**

# Relaxed planning tasks

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

## Definition (relaxation of actions)

The **relaxation**  $a^+$  of a STRIPS action

$a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$  is the action

$a^+ = \langle \text{pre}(a), \text{add}(a), \emptyset \rangle$ .

## Definition (relaxation of planning tasks)

The **relaxation**  $\Pi^+$  of a STRIPS planning task  $\Pi = \langle P, A, I, G \rangle$

is the planning task  $\Pi^+ := \langle P, \{a^+ \mid a \in A\}, I, G \rangle$ .

## Definition (relaxation of action sequences)

The **relaxation** of an action sequence  $\rho = a_1 \dots a_n$  is the action

sequence  $\rho^+ := a_1^+ \dots a_n^+$ .

# Relaxed planning tasks: terminology

- STRIPS planning tasks without delete effects are called **relaxed planning tasks**.
- Plans for relaxed planning tasks are called **relaxed plans**.
- If  $\Pi$  is a STRIPS planning task and  $\rho^+$  is a plan for  $\Pi^+$ , then  $\rho^+$  is called a **relaxed plan for  $\Pi$** .

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristics  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Example: Logistics

: תרגום נבנ



⊗ Pre:  $\text{at}(T, \text{left})$   
add:  $\text{at}(T, \text{right})$   
del:  $\text{at}(T, \text{left})$

- Initial state  $I$ :  $\{\text{at}(A, \text{Left}), \text{at}(T, \text{Left}), \text{at}(B, \text{Right})\}$
- $f(I, \text{Drive(Left, Right)}) = \otimes$   
 $\{\text{at}(A, \text{Left}), \text{at}(T, \text{Right}), \text{at}(B, \text{Right})\}$  הזקקה גייתה משלך
- $f(I, \text{Drive(Left, Right)}^+) =$  ↙ משלך  
 $\{\text{at}(A, \text{Left}), \text{at}(T, \text{Left}), \text{at}(T, \text{Right}), \text{at}(B, \text{Right})\}$
- $f(I, \langle \text{Drive(Left, Right)}, \text{Load}(A, \text{Left}) \rangle)$  is undefined
- $f(I, \langle \text{Drive(Left, Right)}^+, \text{Load}(A, \text{Left})^+ \rangle) =$   
 $\{\text{at}(A, \text{Left}), \text{at}(T, \text{Left}), \text{at}(T, \text{Right}), \text{at}(B, \text{Right}), \text{in}(A, T)\}$  ממשית ריבועית יפה, אבל לא מוגדר

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

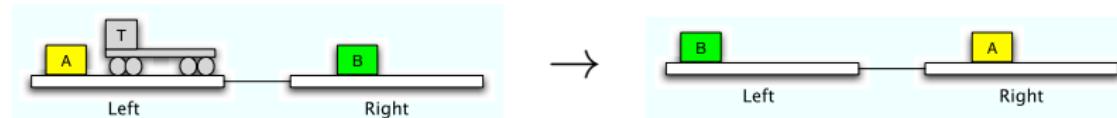
STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Example: Logistics



- Optimal plan:

- 1  $load(A, T, Left)$ ,
- 2  $drive(Left, Right)$ ,
- 3  $unload(A, T, Right)$ ,
- 4  $load(B, T, Right)$ ,
- 5  $drive(Right, Left)$ , ↖ נסעה מימין משמאל
- 6  $unload(B, T, Left)$

- Optimal relaxed plan: ??? (subsequence of the optimal plan)

- $h^*(I) = 6, h^+(I) = ???$  ↤

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

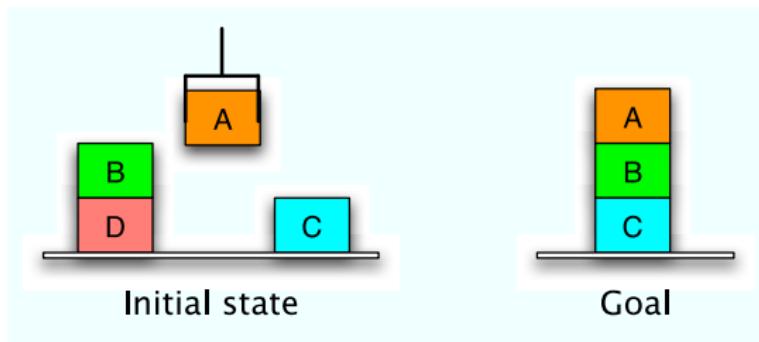
Optimality

Discussion

Concrete  
Heuristics

# Always subsequence? (Just curious)

An optimal relaxed plan can *not* always be obtained by skipping actions from the (real) optimal plan.



- Optimal plan:  
 $\langle \text{putdown}(A), \text{unstack}(B, D), \text{stack}(B, C), \text{pickup}(A), \text{stack}(A, B) \rangle$
- Optimal relaxed subsequence: ???  $\text{stack}(A, B), \text{pickup}(B)$   
 $\text{stack}(B, C)$
- Optimal relaxed plan: ??? *Subsequence is not always a plan*

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

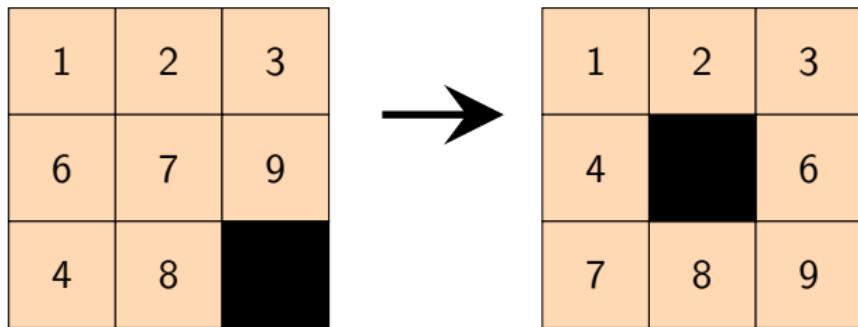
The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Concrete  
Heuristics

# Example: 8-Puzzle



- Real problem:
  - A tile can move from square A to square B if A is adjacent to B and B is blank
- Monotonically relaxed problem:
  - A tile can move from square A to square B if A is adjacent to B and B is blank (!!!)
  - In effect ...

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

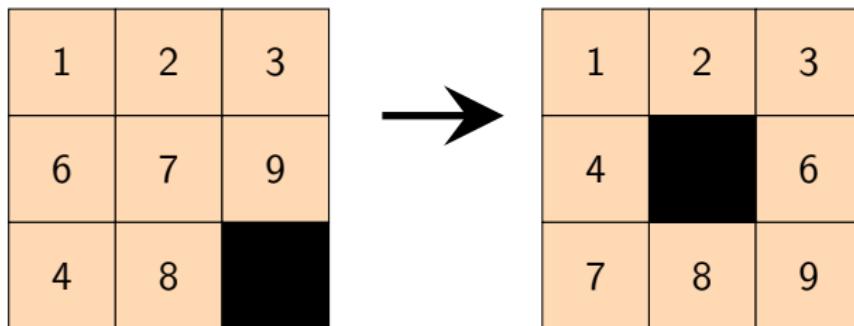
STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Example: 8-Puzzle



- A tile can move from square A to square B if A is adjacent to B and B is blank - solution distance  $h^*$
- A tile can move from square A to square B if A is adjacent to B - manhattan distance heuristic  $h^{MD}$
- A tile can move from square A to square B if A is adjacent to B and B is blank; in effect, the tile is at both A and B, and both A and B are blank -  $h^+$

Here:  $h^*(s_0) = 8$ ,  $h^{MD}(s_0) = 6$ ,  $h^+(s_0) = ???$

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Concrete  
Heuristics

# Example: 8-Puzzle

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 6 | 7 | 9 |
| 4 | 8 |   |



|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 |   | 6 |
| 7 | 8 | 9 |

Optimal MD plan:

- ①  $move(t_9, p_6, p_9)$
- ②  $move(t_7, p_5, p_8)$
- ③  $move(t_6, p_4, p_5)$
- ④  $move(t_6, p_5, p_6)$
- ⑤  $move(t_4, p_7, p_4)$
- ⑥  $move(t_7, p_8, p_7)$

Optimal relaxed plan:

- ①  $move(t_9, p_6, p_9)$
- ②  $move(t_8, p_8, p_9)$
- ③  $move(t_7, p_5, p_8)$
- ④  $move(t_6, p_4, p_5)$
- ⑤  $move(t_6, p_5, p_6)$
- ⑥  $move(t_4, p_7, p_4)$
- ⑦  $move(t_7, p_8, p_7)$

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Example: 8-Puzzle

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 6 | 7 | 9 |
| 4 | 8 |   |



|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 |   | 6 |
| 7 | 8 | 9 |

Optimal MD plan:

- ➊  $move(t_9, p_6, p_9)$
- ➋  $move(t_7, p_5, p_8)$
- ➌  $move(t_6, p_4, p_5)$
- ➍  $move(t_6, p_5, p_6)$
- ➎  $move(t_4, p_7, p_4)$
- ➏  $move(t_7, p_8, p_7)$

Optimal relaxed plan:

- ➊  $move(t_9, p_6, p_9)$
- ➋  $move(t_8, p_8, p_9)$
- ➌  $move(t_7, p_5, p_8)$
- ➍  $move(t_6, p_4, p_5)$
- ➎  $move(t_6, p_5, p_6)$
- ➏  $move(t_4, p_7, p_4)$
- ➐  $move(t_7, p_8, p_7)$

המקרה  
בז'ר

בז'ר  
בז'ר

בז'ר  
בז'ר

. זט"

So  $h^*(s_0) = 8$ ,  $h^{MD}(s_0) = 6$ ,  $h^+(s_0) = 7 (> h^{MD}!)$

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# 8-Puzzle: $h^+$ vs. $h^{MD}$

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 6 | 7 | 9 |
| 4 | 8 |   |



|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 |   | 6 |
| 7 | 8 | 9 |

*MP<sub>i</sub> (blank)  $\geq$  blank  $\oplus$  non-blank blanks.  $\Rightarrow$  MP  $\leq$  h<sup>+</sup>*

$h^+$  dominates  $h^{MD}$

- The goal is given as a conjunction of  $at(t_i, p_j)$  atoms
- Achieving each single one of them takes at least as many steps as the respective tile's Manhattan distance
- Each action moves a single tile only

And we have just seen that  $h^+$  strictly dominates  $h^{MD}$

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Concrete  
Heuristics

# Dominating states

תנאי חישוב גזע ב פס. ש-ה פס יוצר סדרה יפה נ-ו ב-  
. ש' פס סדרה יוניק ס-ה

A state  $s'$  **dominates** another state  $s$  iff  $s \subseteq s'$ .

## Lemma (relaxation)

Let  $s$  be a state, let  $s'$  be a state that dominates  $s$ , and let  $\rho$  be an action sequence which is applicable in  $s$ .

Then  $\rho^+$  is applicable in  $s'$  and  $app_{\rho^+}(s')$  dominates  $app_\rho(s)$ . Moreover, if  $\rho$  leads to a goal state from  $s$ , then  $\rho^+$  leads to a goal state from  $s'$ .

## Proof.

The “moreover” part is immediate from  $app_{\rho^+}(s')$  dominating  $app_\rho(s)$ . Prove the rest by induction over the length of  $\rho$ .

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Consequences of the relaxation lemma

לעגנץ

Corollary (relaxation leads to dominance and preserves plans)

Let  $\rho$  be an action sequence which is applicable in state  $s$ .

Then  $\rho^+$  is applicable in  $s$  and  $app_{\rho^+}(s)$  dominates  $app_{\rho}(s)$ .

If  $\rho$  is a plan for  $\Pi$ , then  $\rho^+$  is a plan for  $\Pi^+$ .

Proof.

Apply relaxation lemma with  $s' = s$ . □

- ~ Relaxations of plans are relaxed plans.
- ~ Relaxations are no harder to solve than the original task.
- ~ Optimal relaxed plans are never longer than optimal plans for original tasks.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Consequences of the relaxation lemma (ctd.)

15<5.3

## Corollary (relaxation preserves dominance)

Let  $s$  be a state, let  $s'$  be a state that dominates  $s$ , and let  $\rho^+$  be a relaxed action sequence applicable in  $s$ . Then  $\rho^+$  is applicable in  $s'$  and  $\text{app}_{\rho^+}(s')$  dominates  $\text{app}_{\rho^+}(s)$ .

## Proof.

Apply relaxation lemma with  $\rho^+$  for  $\rho$ , noting that  $(\rho^+)^+ = \rho^+$ .



- ~ If there is a relaxed plan starting from state  $s$ , the same plan can be used starting from a dominating state  $s'$ .
- ~ Making a transition to a dominating state never hurts in relaxed planning tasks.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality

Discussion

Concrete  
Heuristics

# Monotonicity of relaxed planning tasks

15/19

We need one final property before we can provide an algorithm for solving relaxed planning tasks.

## Lemma (monotonicity)

Let  $a^+ = \langle \text{pre}(a), \text{add}(a), \emptyset \rangle$  be a relaxed action and let  $s$  be a state in which  $a^+$  is applicable.

Then  $\text{app}_{a^+}(s)$  dominates  $s$ .

## Proof.

Since relaxed actions only have positive effects, we have  
 $s \subseteq s \cup \text{add}(a) = \text{app}_{a^+}(s)$ . □

~ Together with our previous results, this means that making a transition in a relaxed planning task **never** hurts.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Greedy algorithm for relaxed planning tasks

The relaxation and monotonicity lemmas suggest the following algorithm for solving relaxed planning tasks:

## Greedy planning algorithm for $\langle P, A^+, I, G \rangle$

$s := I$

$\rho^+ := \epsilon$  ← מערך ריק  
**forever:**

**if**  $G \subseteq s$ :

**return**  $\rho^+$

**else if** there is an action  $a^+ \in A^+$  applicable in  $s$

with  $app_{a^+}(s) \neq s$ :

Append such an action  $a^+$  to  $\rho^+$ .

$s := app_{a^+}(s)$

**else:**

**return** unsolvable

$$app_{a^+}(s) = s \cup add(a^+)$$

הנודים לא ירדו מ-  
הNODES לא ירדו מ-  
NODES :  
NODES  
הNODES לא ירדו מ-  
הNODES לא ירדו מ-

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic

Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm

Optimality  
Discussion

Concrete  
Heuristics

# Correctness of the greedy algorithm

The algorithm is **sound**:

- If it returns a plan, this is indeed a correct solution.
- If it returns “unsolvable”, the task is indeed unsolvable
  - Upon termination, there clearly is no relaxed plan from  $s$ .
  - By iterated application of the monotonicity lemma,  $s$  dominates  $I$ .
  - By the relaxation lemma, there is no solution from  $I$ .

What about **completeness** (termination) and **runtime**?

- Each iteration of the loop adds at least one atom to  $s$ .
- This guarantees termination after at most  $|P|$  iterations.
- Thus, the algorithm can clearly be implemented to run in polynomial time.
  - A good implementation runs in  $O(\|\Pi\|)$ .

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Using the greedy algorithm as a heuristic

מבחן גס נרמולן נון:  $h$

We can apply the greedy algorithm within heuristic search:

- In a search node  $\sigma$ , solve the relaxation of the planning task with  $state(\sigma)$  as the initial state.
- Set  $h(\sigma)$  to the length of the generated relaxed plan.

Is this an **admissible** heuristic?

- Yes if the relaxed plans are **optimal** (due to the plan preservation corollary).
- However, usually they are not, because our greedy planning algorithm is very poor.

(What about safety? Goal-awareness? Consistency?)

טבז לא יס סנו לא

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# The set cover problem

ה问题是怎样的？

To obtain an admissible heuristic, we need to generate optimal relaxed plans. Can we do this efficiently?

This question is related to the following problem:

## Problem (set cover)

**Given:** a finite set  $U$ , a collection of subsets  $C = \{C_1, \dots, C_n\}$  with  $C_i \subseteq U$  for all  $i \in \{1, \dots, n\}$ , and a natural number  $K$ .

**Question:** Does there exist a set cover of size at most  $K$ , i. e., a subcollection  $S = \{S_1, \dots, S_m\} \subseteq C$  with  $S_1 \cup \dots \cup S_m = U$  and  $m \leq K$ ?

The following is a classical result from complexity theory:

## Theorem

The set cover problem is NP-complete.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Hardness of optimal relaxed planning

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

## Theorem (optimal relaxed planning is hard)

*The problem of deciding whether a given relaxed planning task has a plan of length at most  $K$  is NP-complete.*

## Proof.

For membership in NP, guess a plan and verify. It is sufficient to check plans of length at most  $|P|$ , so this can be done in nondeterministic polynomial time.

For hardness, we reduce from the set cover problem.

# Hardness of optimal relaxed planning (ctd.)

## Proof (ctd.)

Given a set cover instance  $\langle U, C, K \rangle$ , we generate the following relaxed planning task  $\Pi^+ = \langle P, I, A^+, G \rangle$ :

- $P = U$
- $I = \emptyset \quad \equiv I = \{p = 0 \mid p \in P\}$
- $A^+ = \{\langle \emptyset, \bigcup_{p \in C_i} \{p\}, \emptyset \rangle \mid C_i \in C\}$
- $G = U$

If  $S$  is a set cover, the corresponding actions form a plan.

Conversely, each plan induces a set cover by taking the subsets corresponding to the actions. Clearly, there exists a plan of length at most  $K$  iff there exists a set cover of size  $K$ .

Moreover,  $\Pi^+$  can be generated from the set cover instance in polynomial time, so this is a polynomial reduction. □

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS

heuristic

Relaxation

heuristics

The relaxation

lemma

Greedy algorithm

Optimality

Discussion

Concrete

Heuristics

# Using relaxations in practice

How can we use relaxations for heuristic planning in practice?

Different possibilities:

- Implement an **optimal planner** for relaxed planning tasks and use its solution lengths as an estimate, even though it is NP-hard.  
~~~  $h^+$  heuristic (*not that realistic. why?*) ... condition p'noob
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.  
~~~  $h_{\max}$  heuristic,  $h_{\text{add}}$  heuristic
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but “reasonable”.  
~~~  $h_{\text{FF}}$  heuristic

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Using relaxations in practice

How can we use relaxations for heuristic planning in practice?

Different possibilities:

- Implement an **optimal planner** for relaxed planning tasks and use its solution lengths as an estimate, even though it is NP-hard.  
  ~  $h^+$  **heuristic** (*not that realistic. why?*)
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.  
  ~  $h_{\max}$  **heuristic**,  $h_{\text{add}}$  **heuristic**
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but “reasonable”.  
  ~  $h_{\text{FF}}$  **heuristic**

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Using relaxations in practice

How can we use relaxations for heuristic planning in practice?

Different possibilities:

- Implement an **optimal planner** for relaxed planning tasks and use its solution lengths as an estimate, even though it is NP-hard.  
  ~  $h^+$  **heuristic** (*not that realistic. why?*)
- Do not actually solve the relaxed planning task, but compute an estimate of its difficulty in a different way.  
  ~  $h_{\max}$  **heuristic**,  $h_{\text{add}}$  **heuristic**
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but “reasonable”.  
  ~  $h_{\text{FF}}$  **heuristic**

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

STRIPS  
heuristic  
Relaxation  
heuristics

The relaxation  
lemma

Greedy algorithm  
Optimality  
Discussion

Concrete  
Heuristics

# Reminder: Greedy algorithm for relaxed planning tasks

## Greedy planning algorithm for $\langle P, A^+, I, G \rangle$

$s := I$

$\rho^+ := \epsilon$

**forever:**

**if**  $G \subseteq s$ :

**return**  $\rho^+$

**else if** there is an action  $a^+ \in A^+$  applicable in  $s$

        with  $app_{a^+}(s) \neq s$ :

            Append such an action  $a^+$  to  $\rho^+$ .

$s := app_{a^+}(s)$

**else:**

**return** unsolvable

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

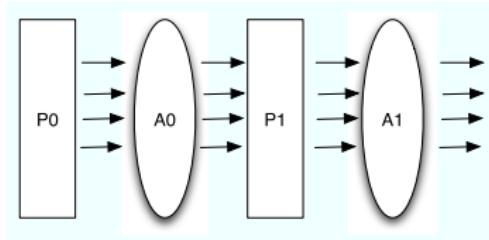
# Graphical “interpretation”: Relaxed planning graphs

- Build a layered **reachability graph**  $P_0, A_0, P_1, A_1, \dots$

$$P_0 = \{p \in I\}$$

$$A_i = \{a \in A \mid \text{pre}(a) \subseteq P_i\}$$

$$P_{i+1} = P_i \cup \{p \in \text{add}(a) \mid a \in A_i\}$$



- Terminate when  $G \subseteq P_i$

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h^{\text{max}}$   
 $h^{\text{add}}$   
 $h^{\text{FF}}$   
Comparison &  
practice

# Running example

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

$$P = \{a, b, c, d, e, f, g, h\}$$

$$I = \{a\}$$

$$G = \{c, d, e, f, g\}$$

$$a_1 = \langle \{a\}, \{b, c\}, \emptyset \rangle$$

$a_1$  *not yet closed*

$$a_2 = \langle \{a, c\}, \{d\}, \emptyset \rangle$$

*not yet closed*

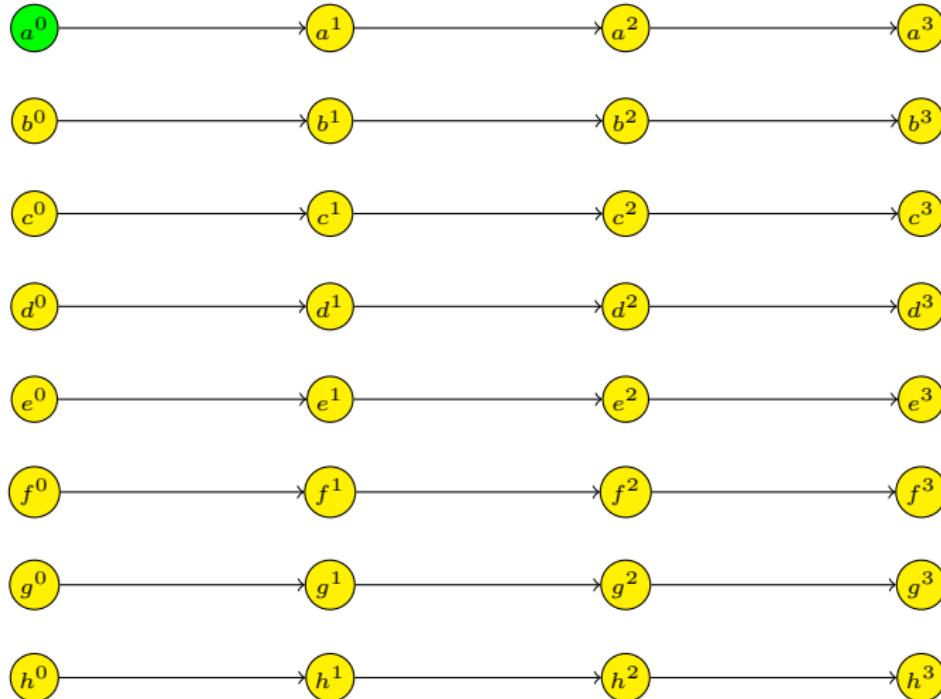
$$a_3 = \langle \{b, c\}, \{e\}, \emptyset \rangle$$

$$a_4 = \langle \{b\}, \{f\}, \emptyset \rangle$$

$$a_5 = \langle \{d\}, \{e, f\}, \emptyset \rangle$$

$$a_6 = \langle \{d\}, \{g\}, \emptyset \rangle$$

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

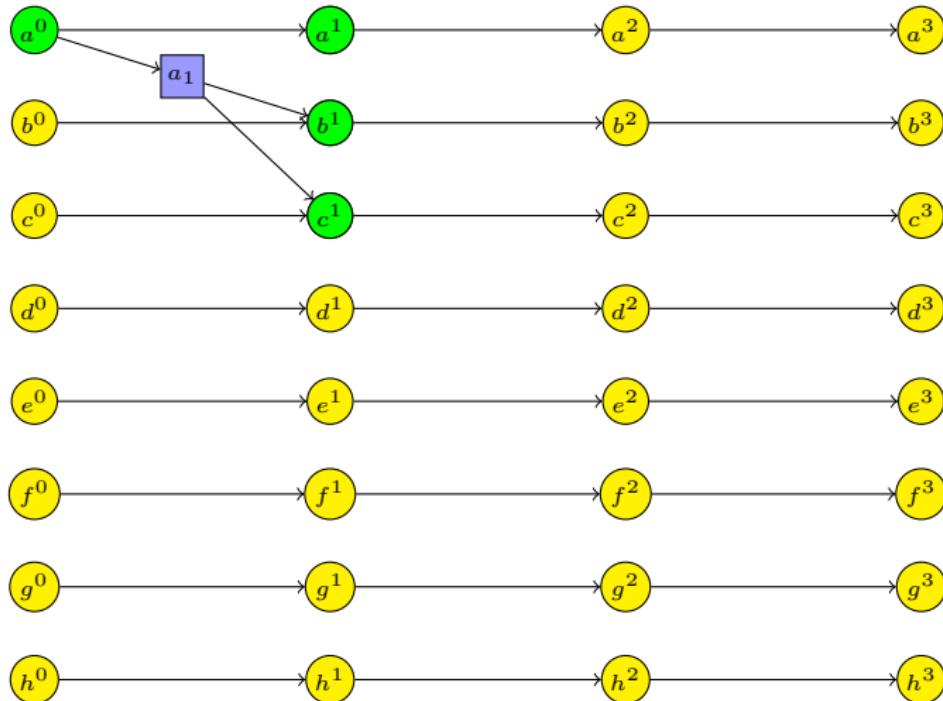
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

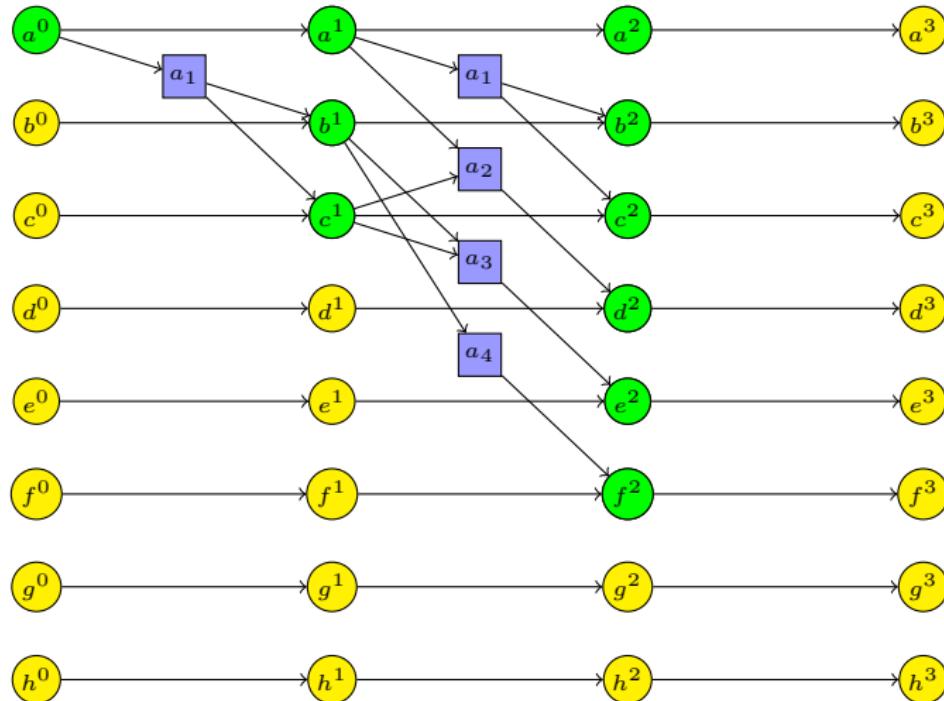
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

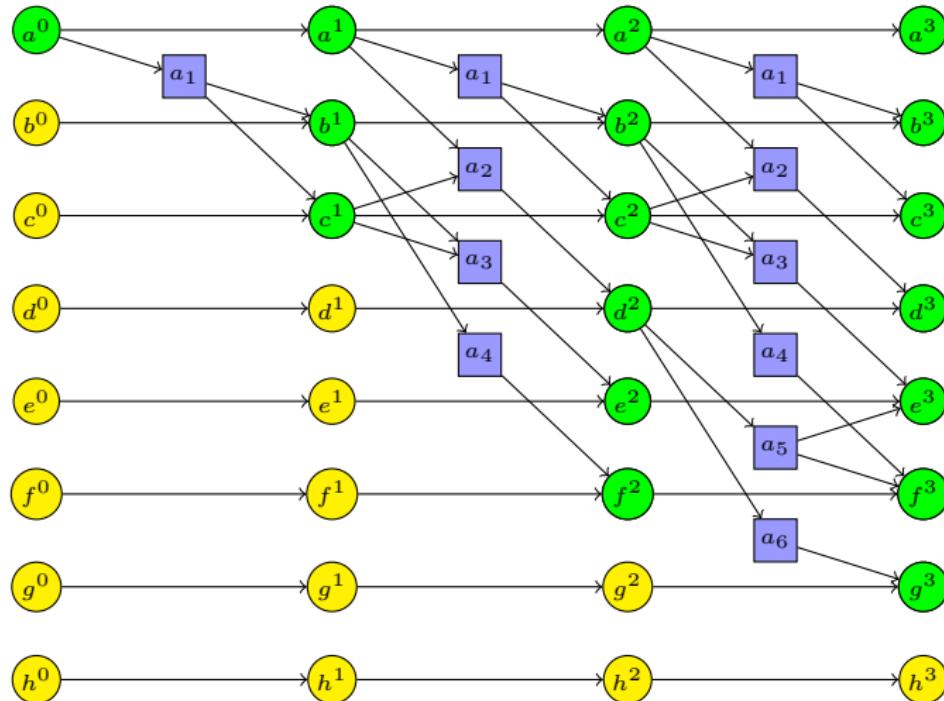
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

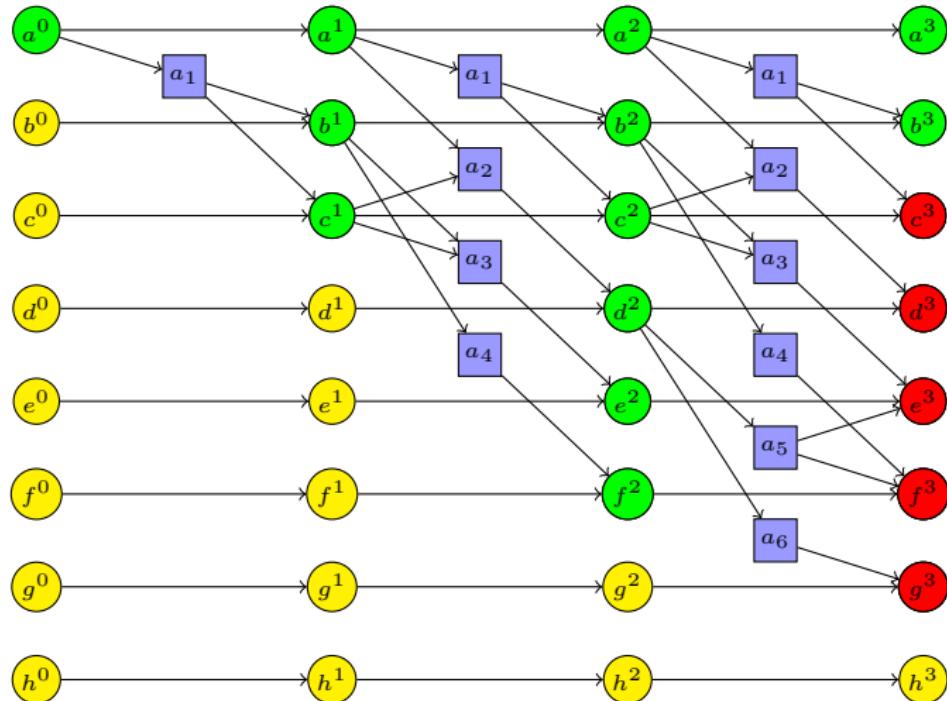
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: Relaxed planning graph



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

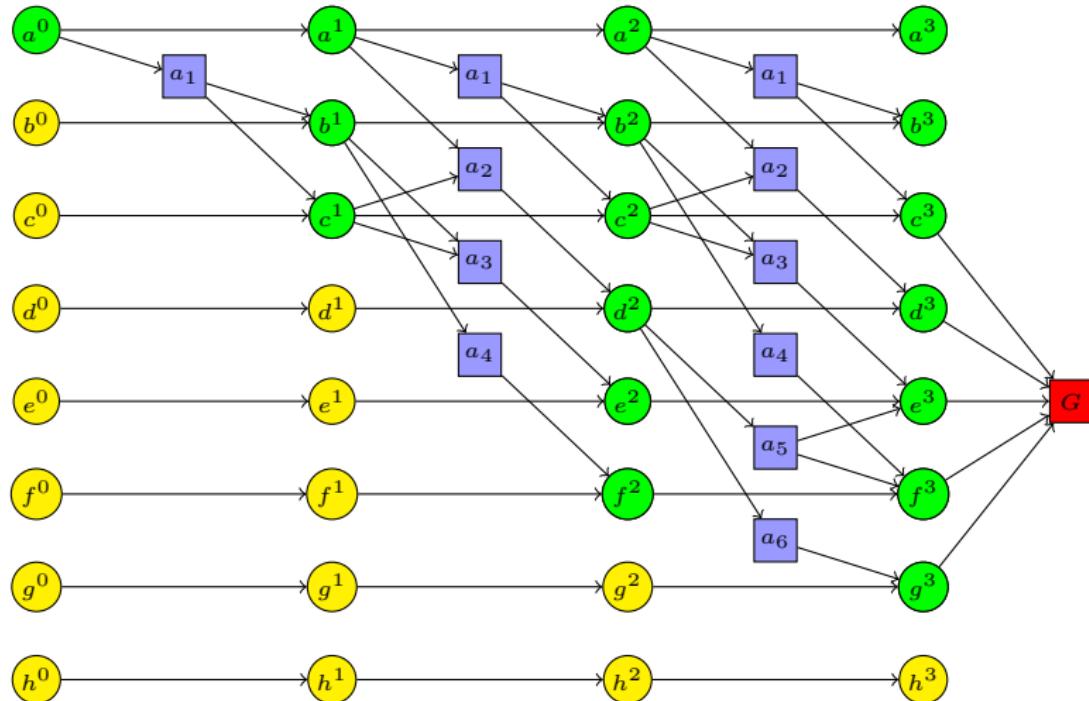
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: Relaxed planning graph



Introduction  
to AI  
C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Generic relaxed planning graph heuristics

## Computing heuristics from relaxed planning graphs

```
def generic-rpg-heuristic(<P, I, O, G>, s):
     $\Pi^+ := \langle P, s, O^+, G \rangle$ 
    for k in {0, 1, 2, ...}:
        rpg := RPGk( $\Pi^+$ )
        if  $G \subseteq P_k$ :
            Annotate nodes of rpg.
            if termination criterion is true:
                return heuristic value from annotations
        else if k = |P|:
            return  $\infty$ 
```

- ~ generic template for heuristic functions
- ~ to get concrete heuristic: fill in highlighted parts

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Concrete examples for the generic heuristic

Many planning heuristics fit the generic template:

- max heuristic  $h_{\max}$
- additive heuristic  $h_{\text{add}}$
- FF heuristic  $h_{\text{FF}}$
- ...

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

## Forward cost heuristics

## Introduction to AI

C. Domshlak

## Introduction

## Obtaining heuristics

## Concrete Heuristics

## Template

## Comparison & practice

- The simplest relaxed planning graph heuristics are **forward cost heuristics**.
  - Examples:  $h_{\max}$ ,  $h_{\text{add}}$
  - Here, node annotations are **cost values** (natural numbers).
  - The cost of a node estimates how expensive (in terms of required operators) it is to make this node true.

**↙ איק זוגות: כנה פוליגמיות זיקת נקבה**

# Forward cost heuristics: fitting the template

## Forward cost heuristics

### Computing annotations:

- Propagate cost values bottom-up using a combination rule for action nodes and a combination rule for proposition nodes.
- At **action nodes**, add 1 after applying combination rule.

↗ f<sub>00</sub>  
↓

↑  
PNICIC

### Termination criterion:

- **stability:** terminate if  $P_k = P_{k-1}$  and cost for each proposition node  $p^k \in P_k$  equals cost for  $p^{k-1} \in P_{k-1}$

### Heuristic value:

- The heuristic value is the cost of the auxiliary goal node.
- Different forward cost heuristics only differ in their choice of combination rules.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# The max heuristic $h_{\max}$ (again)

Forward cost heuristics: max heuristic  $h_{\max}$

Combination rule for action nodes:

- $\text{cost}(u) = \max(\{\text{cost}(v_1), \dots, \text{cost}(v_k)\})$   
(with  $\max(\emptyset) := 0$ )

Combination rule for proposition nodes:

- $\text{cost}(u) = \min(\{\text{cost}(v_1), \dots, \text{cost}(v_k)\})$

In both cases,  $\{v_1, \dots, v_k\}$  is the set of immediate predecessors of  $u$ .

Intuition:

- **Action rule:** If we have to achieve several preconditions, estimate this by the **most expensive** cost. → !נורא מושך
- **Proposition rule:** If we have a choice how to achieve a proposition, pick the **cheapest** possibility.

Introduction  
to AI

C. Domshlak

Introduction

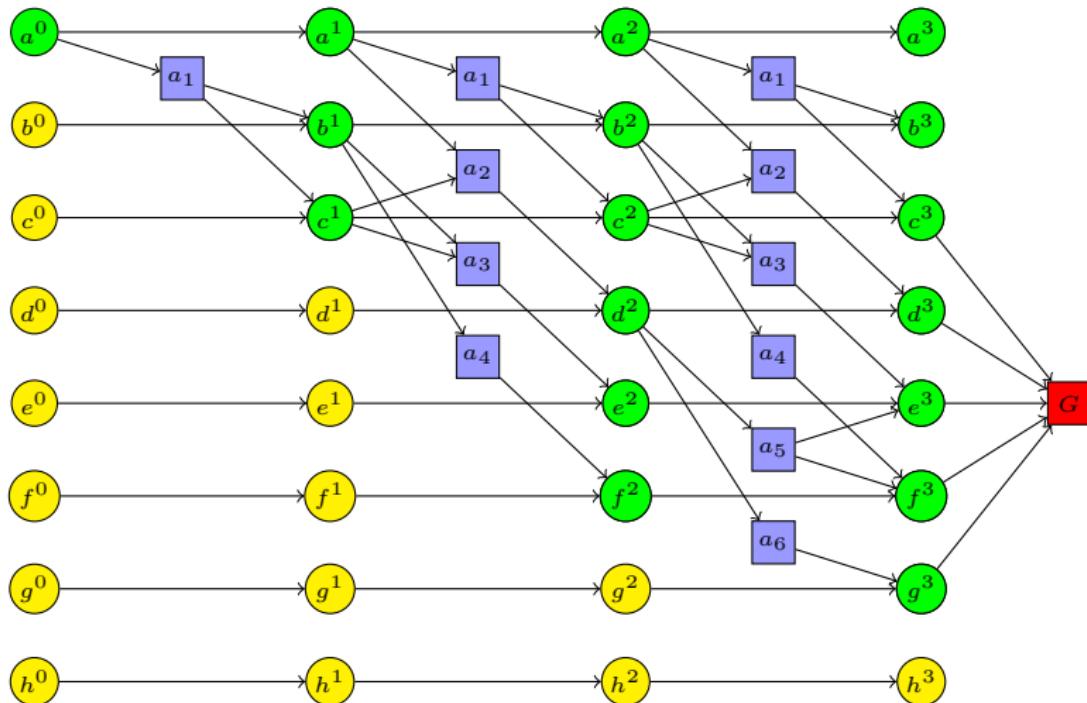
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

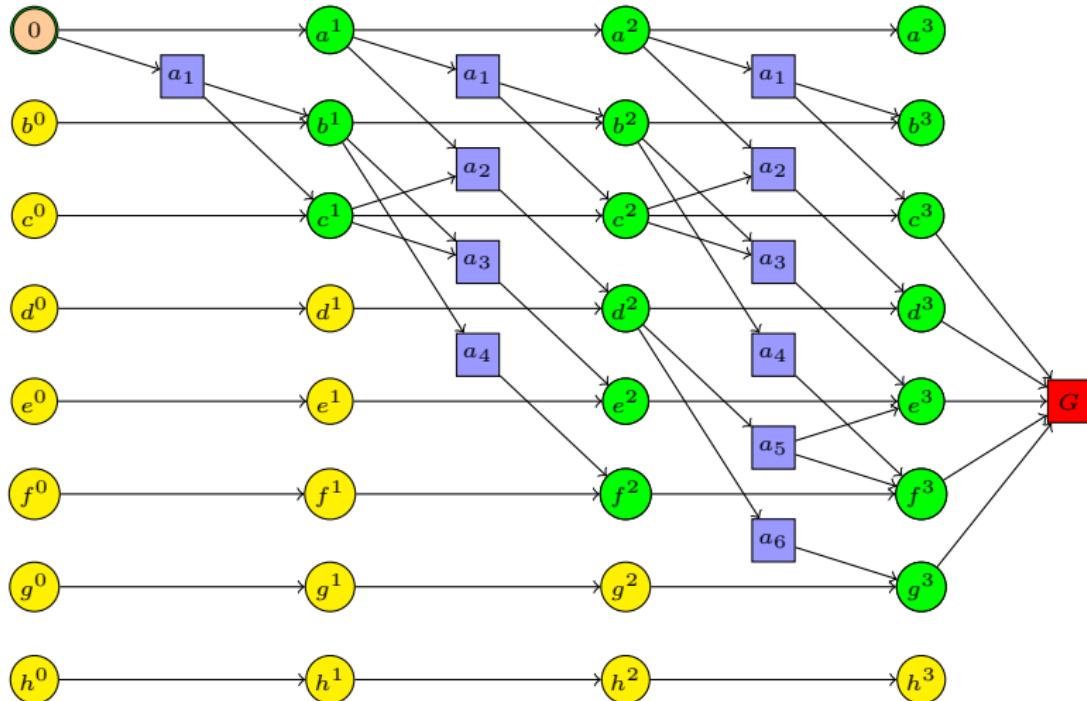
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

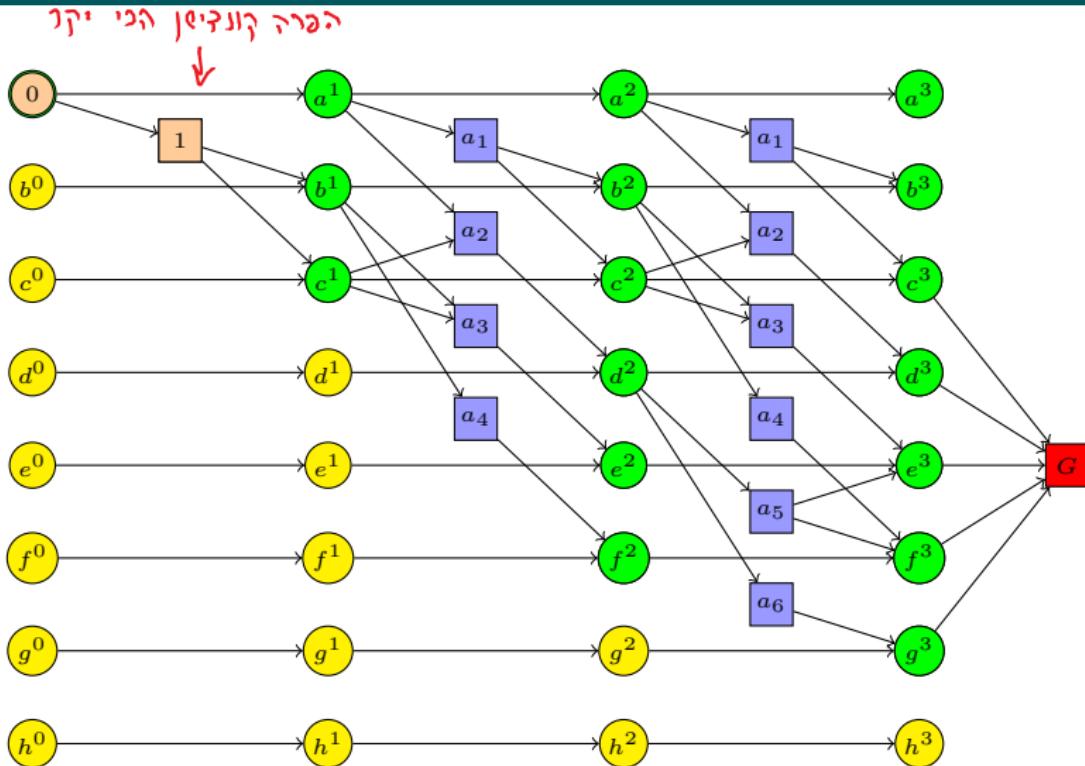
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

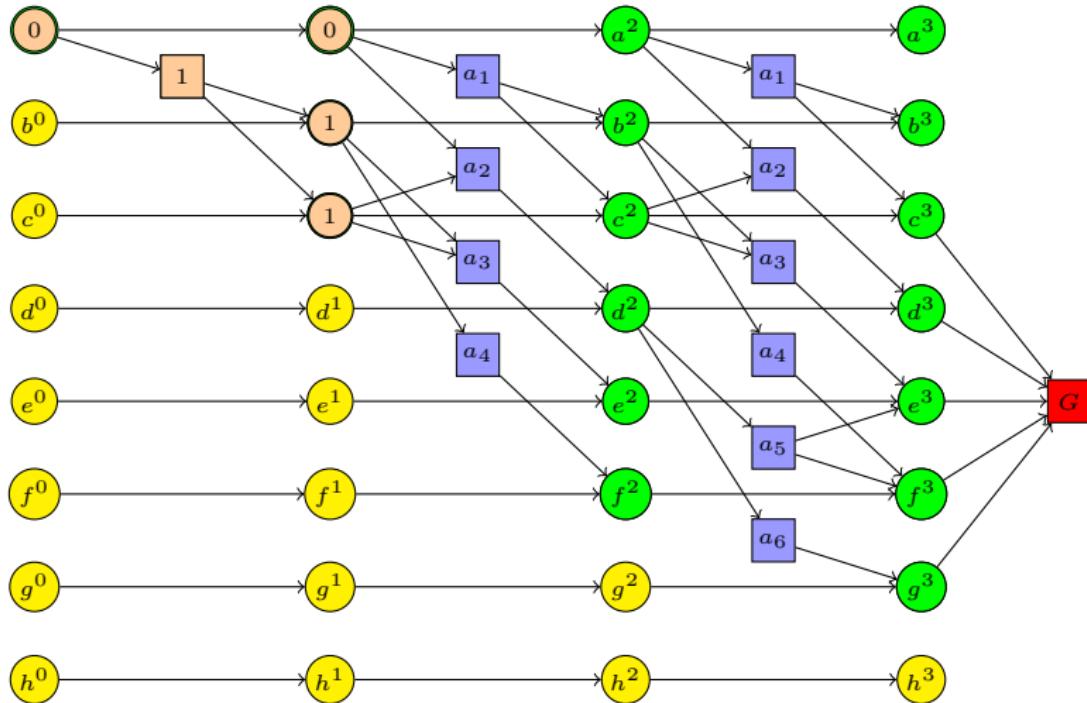
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

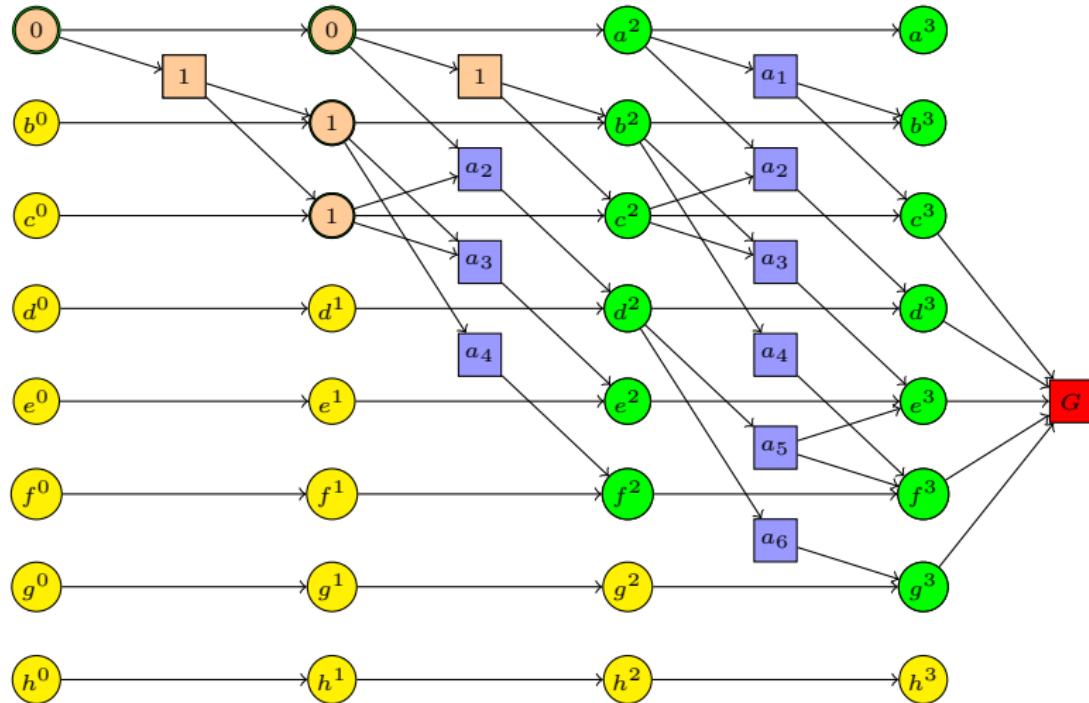
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

## Running example: $h_{\max}$



## Introduction to AI

C. Domshlak

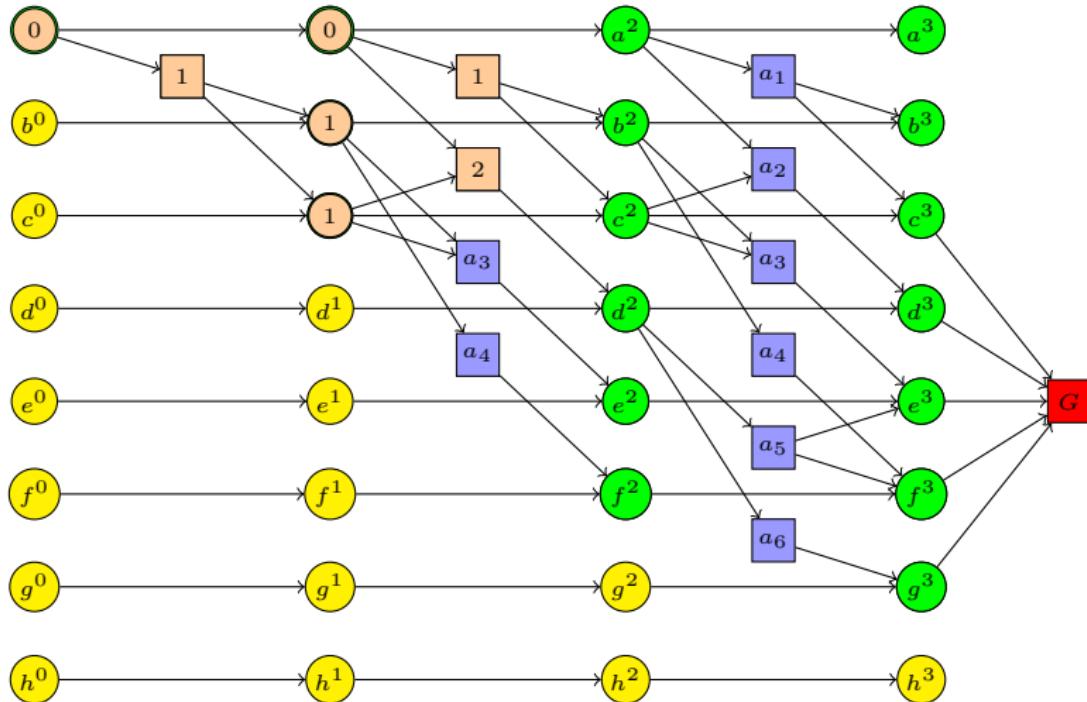
## Obtaining heuristics

## Concrete Heuristics

$h_{\max}$

## Comparison & practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

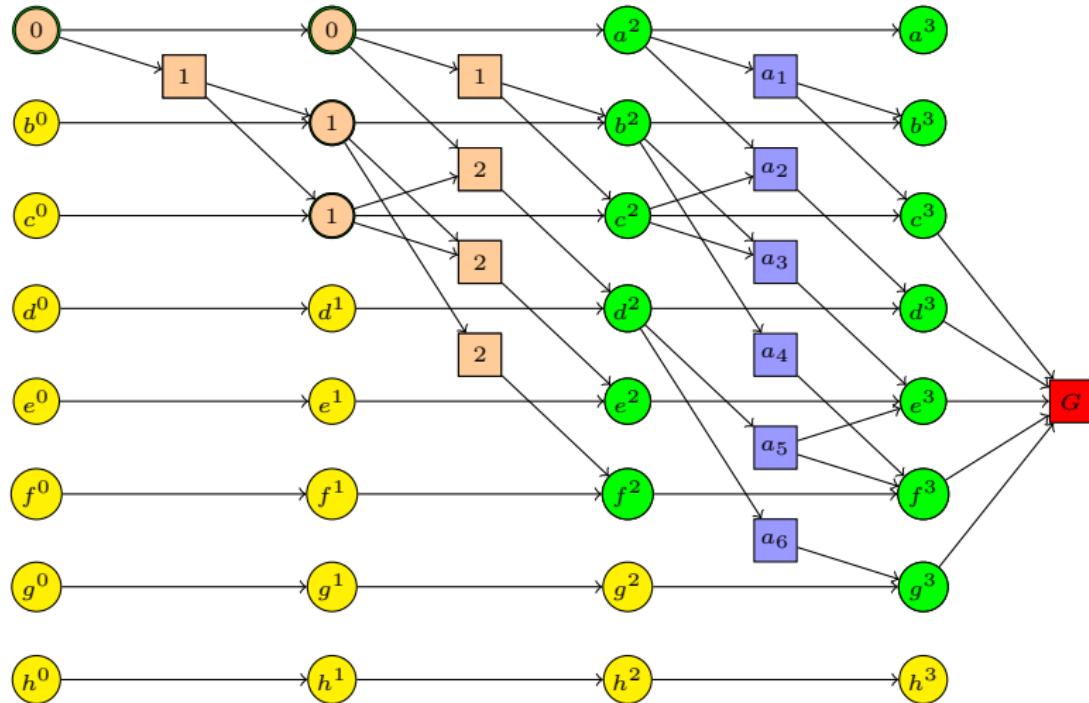
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

## Running example: $h_{\max}$



## Introduction to AI

C. Domshlak

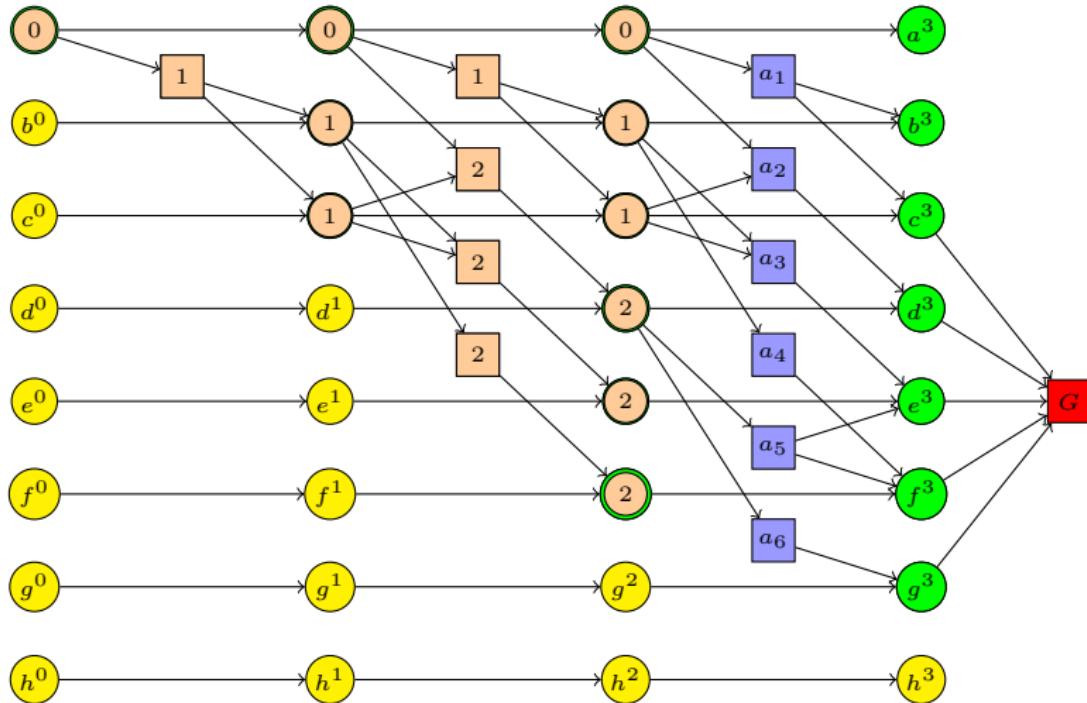
## Obtaining heuristics

## Concrete Heuristics

$h_{\max}$

## Comparison & practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

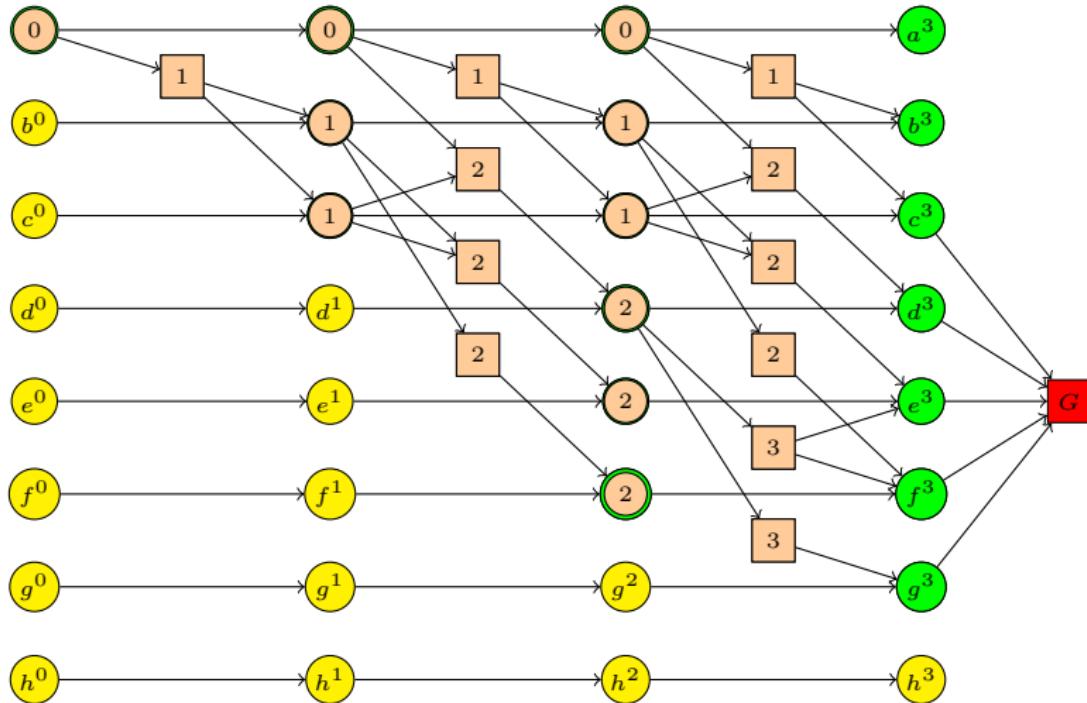
$h_{\max}$

$h_{\text{Add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

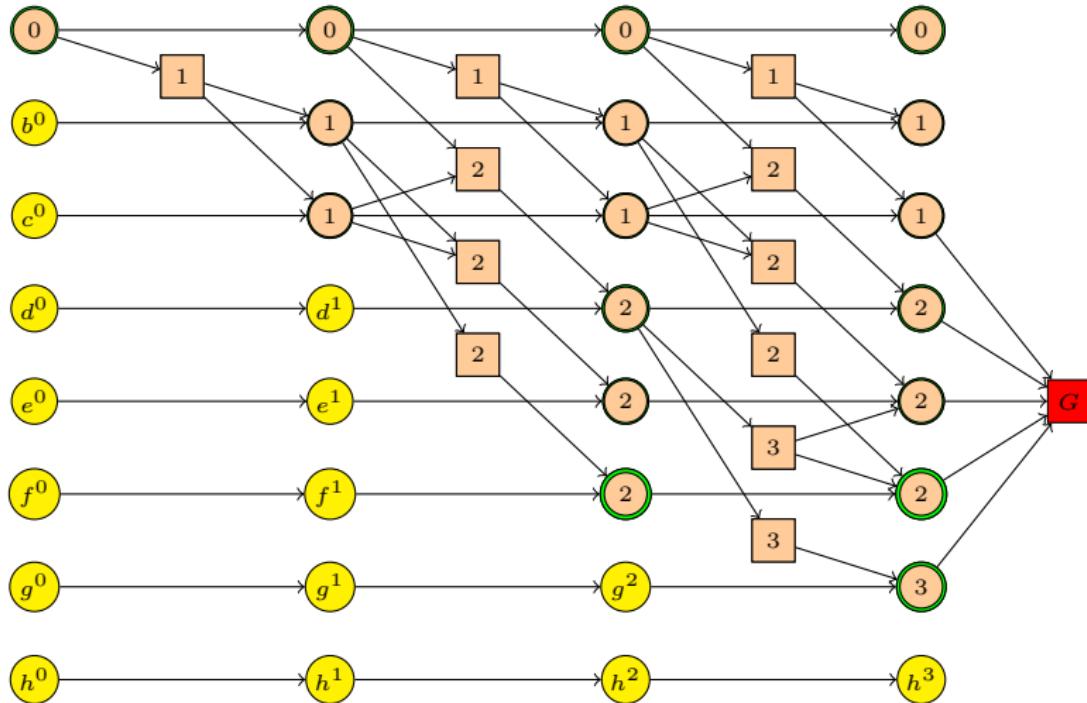
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

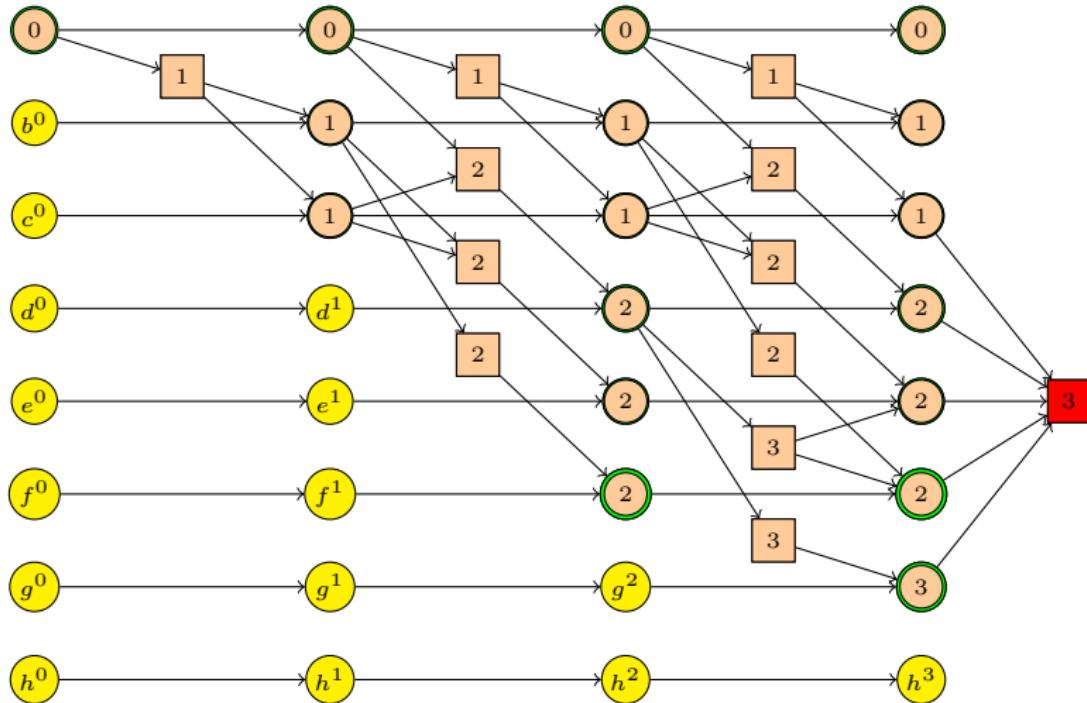
$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\max}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# The additive heuristic

Forward cost heuristics: additive heuristic  $h_{\text{add}}$

Combination rule for action nodes:

- $\text{cost}(u) = \text{cost}(v_1) + \dots + \text{cost}(v_k)$   
(with  $\sum(\emptyset) := 0$ )

Combination rule for proposition nodes:

- $\text{cost}(u) = \min(\{\text{cost}(v_1), \dots, \text{cost}(v_k)\})$

In both cases,  $\{v_1, \dots, v_k\}$  is the set of immediate predecessors of  $u$ .

Intuition:

- **Action rule:** If we have to achieve several preconditions, estimate this by the cost of achieving **each in isolation**. !שניהם נטול ←
- **Proposition rule:** If we have a choice how to achieve a proposition, pick the **cheapest** possibility.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

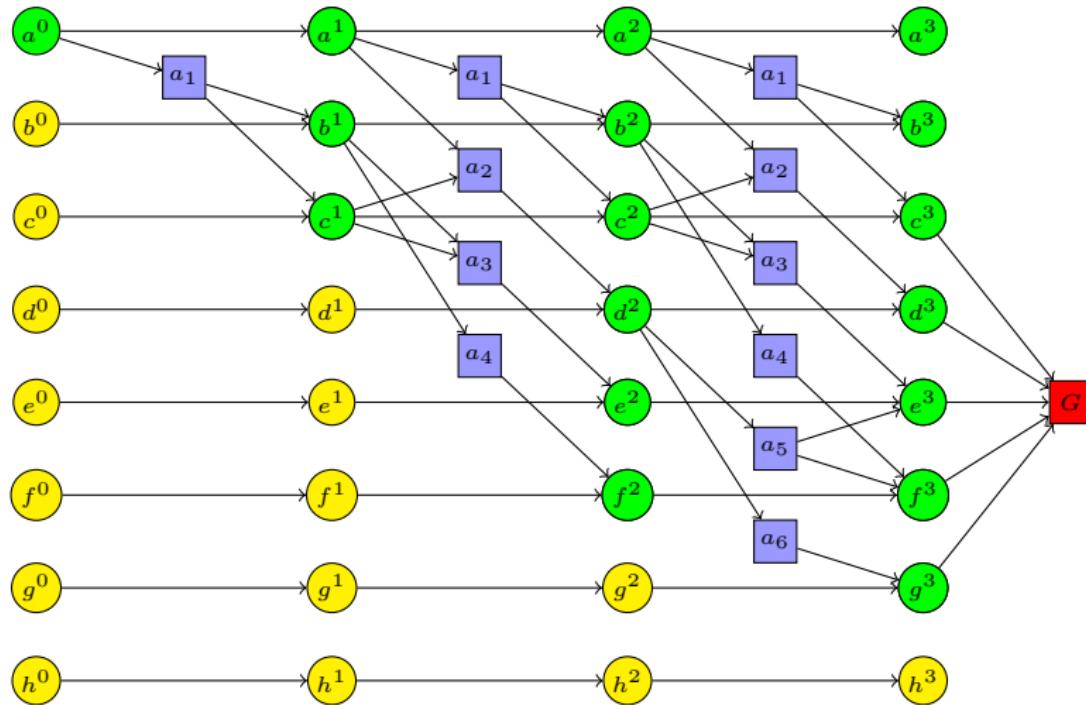
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

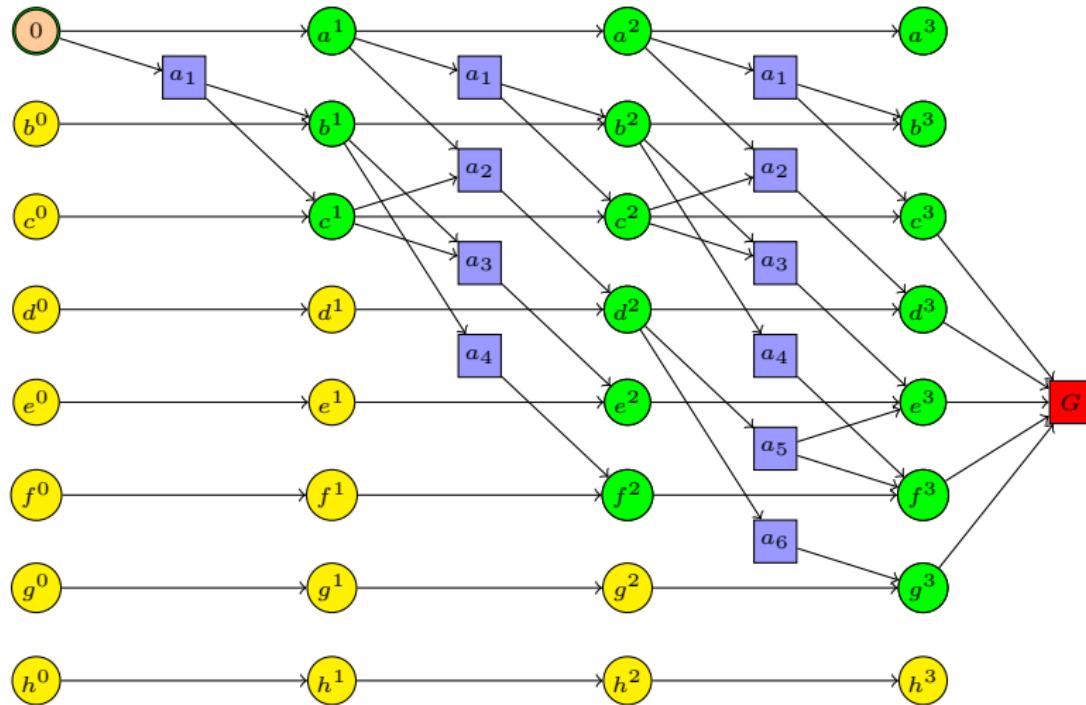
$h^{\max}$

$h_{\text{add}}$

$h^{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

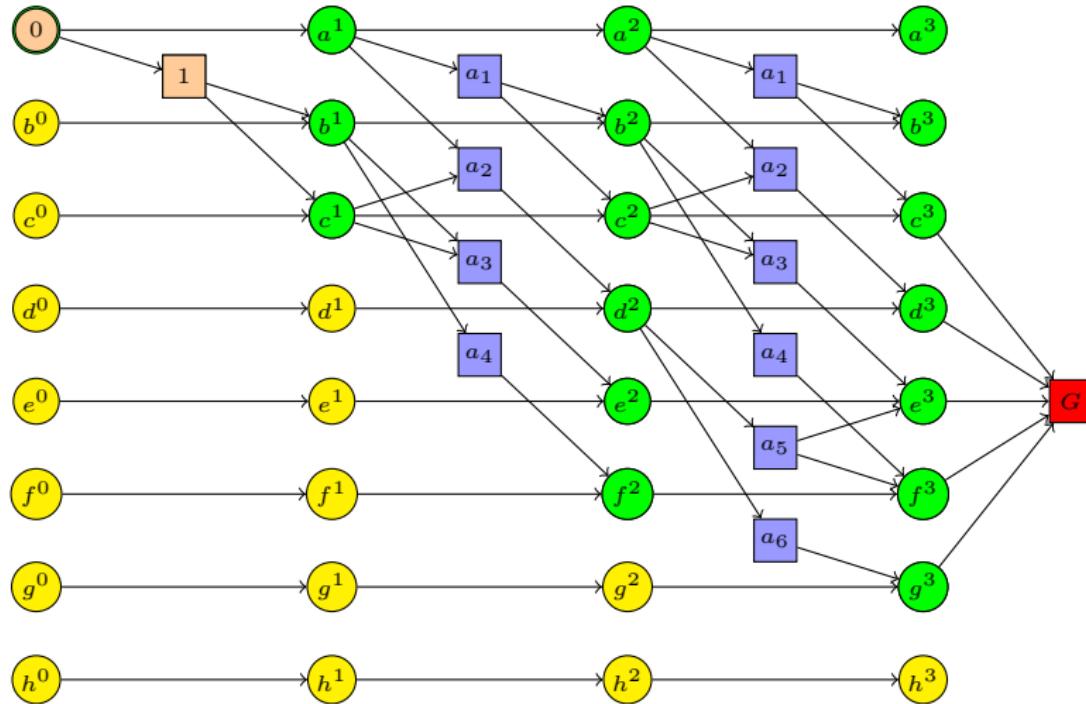
$h^{\max}$

$h_{\text{add}}$

$h^{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

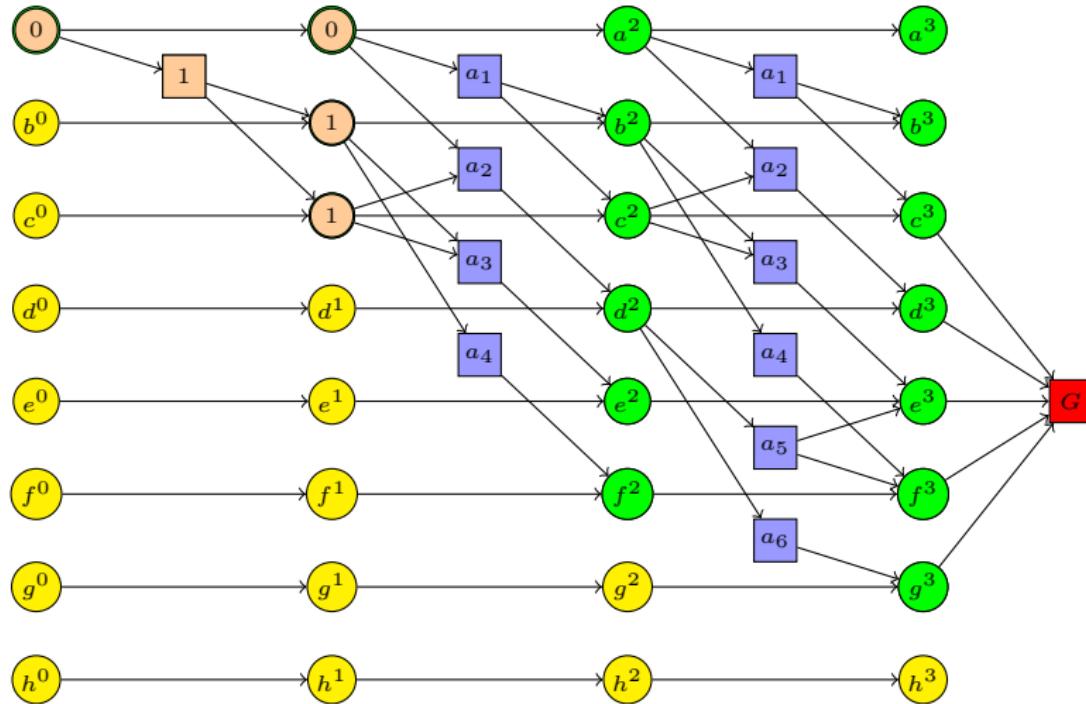
$h^{\max}$

$h_{\text{add}}$

$h^{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

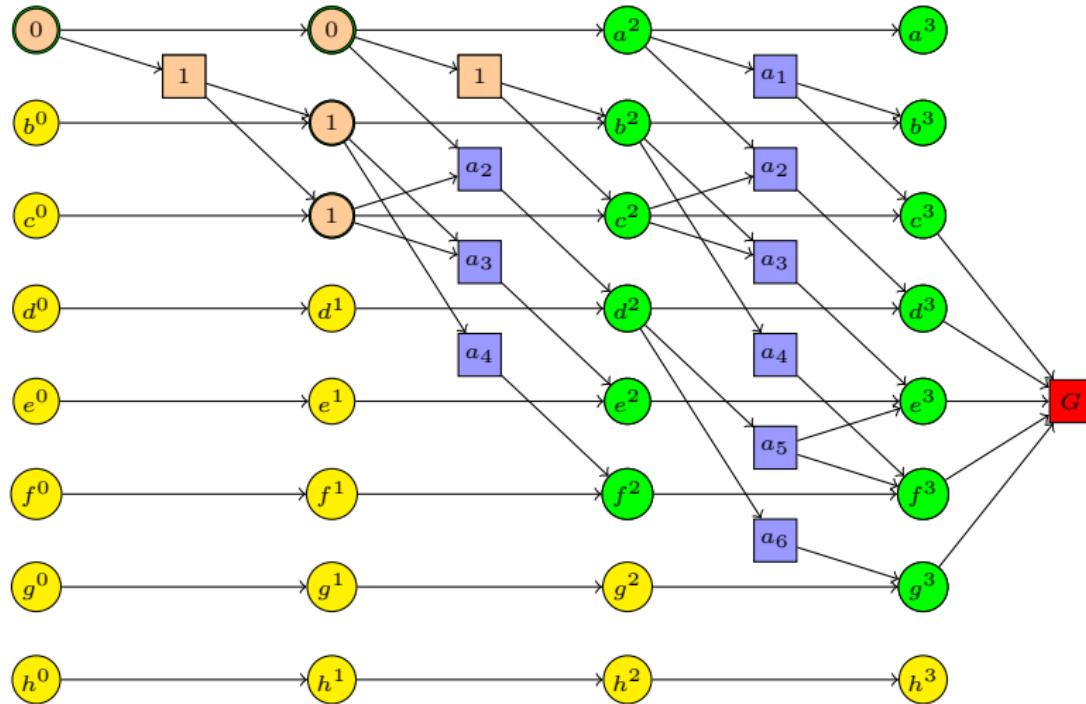
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

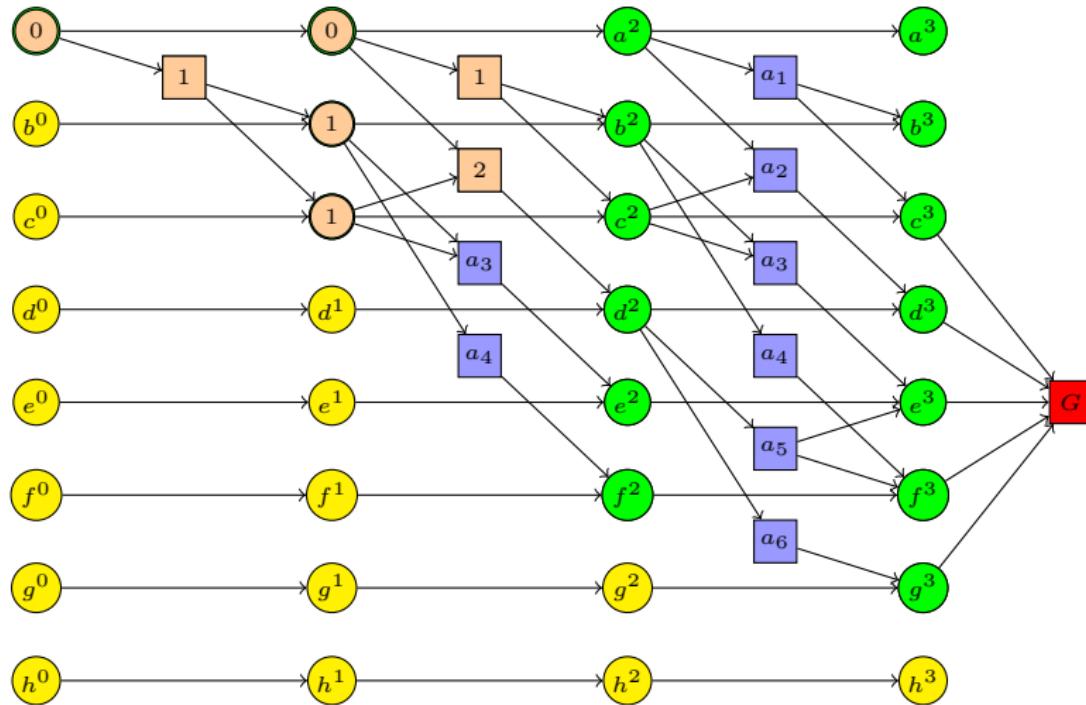
$h^{\max}$

$h_{\text{add}}$

$h^{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

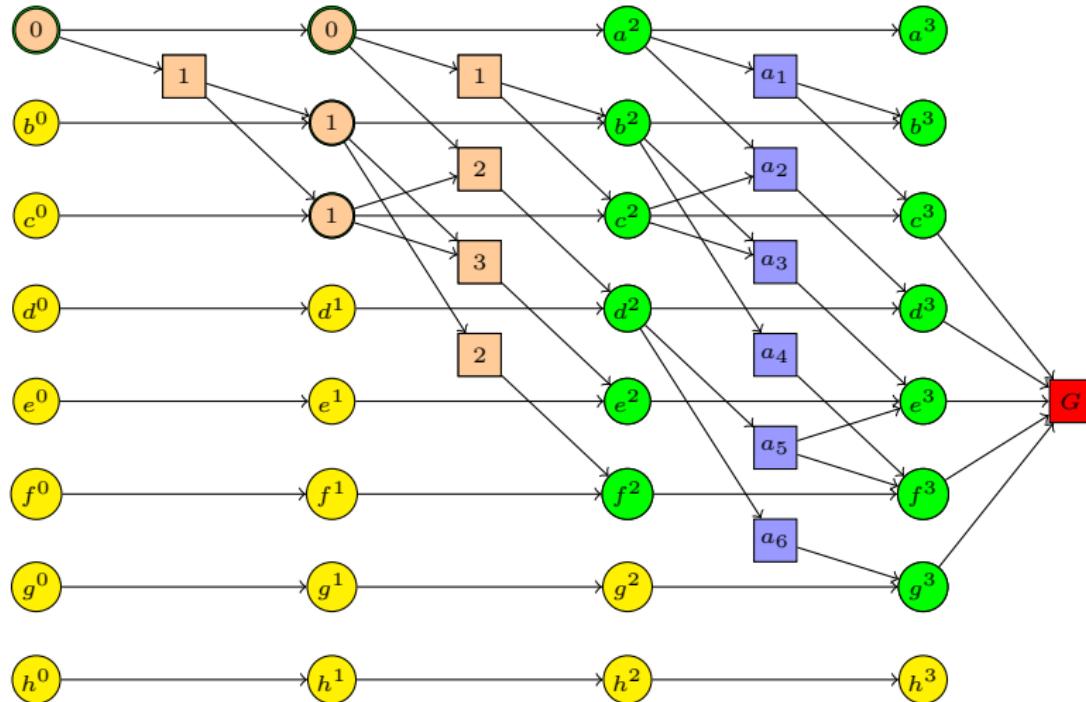
$h^{\max}$

$h_{\text{add}}$

$h^{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

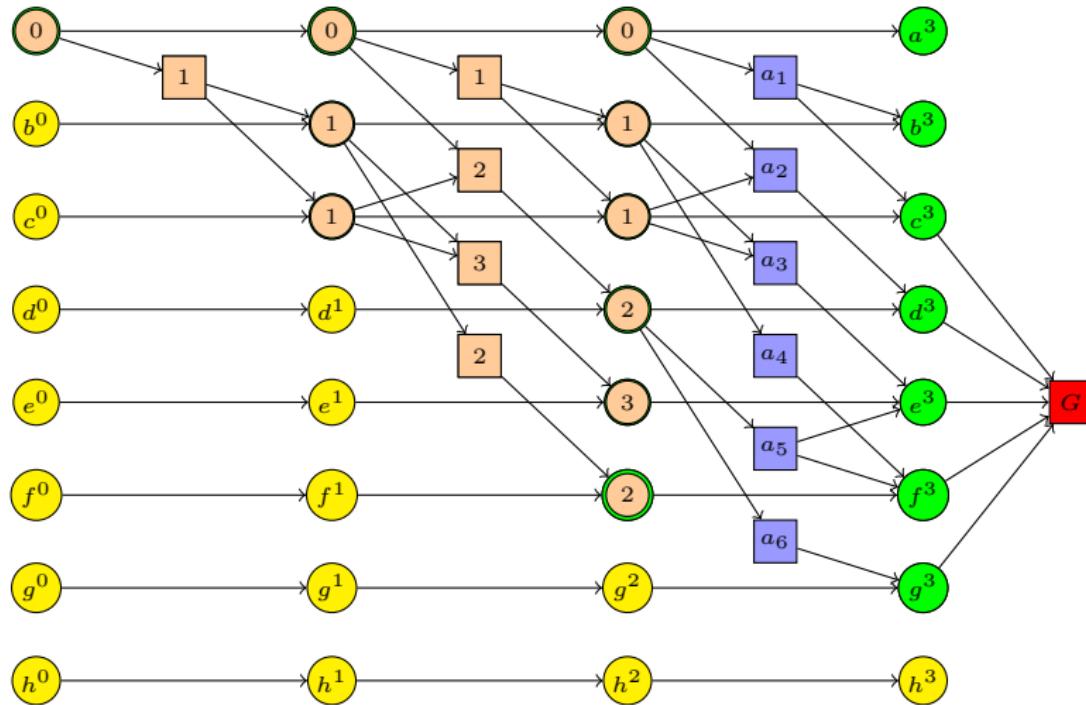
$h^{\max}$

$h_{\text{add}}$

$h^{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

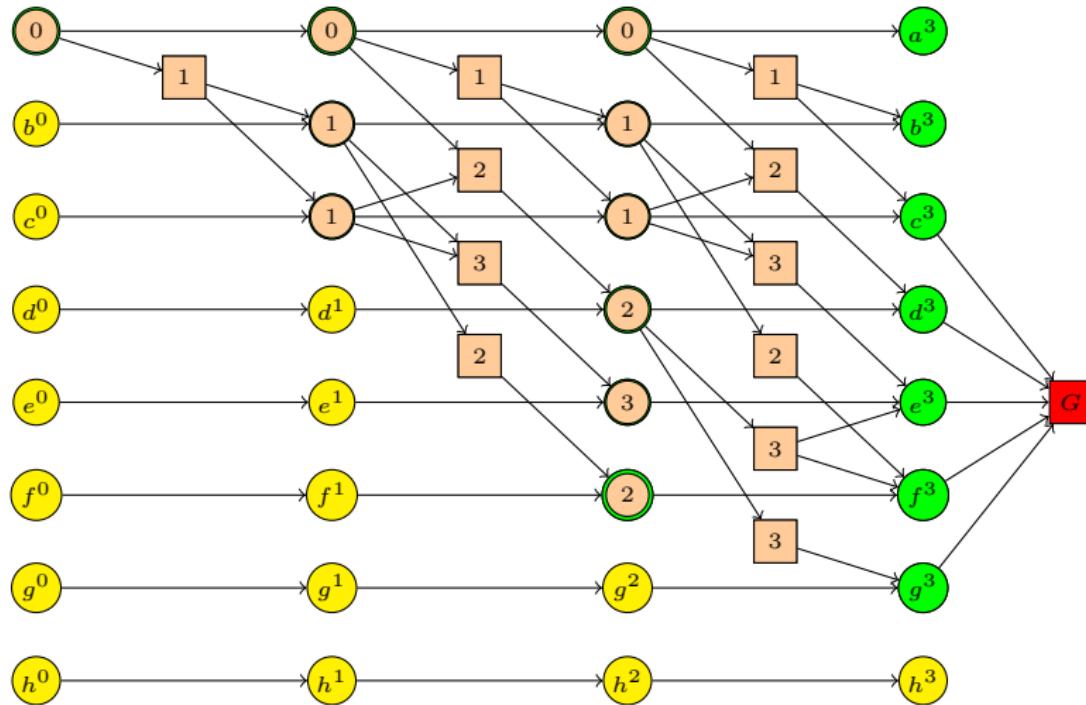
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

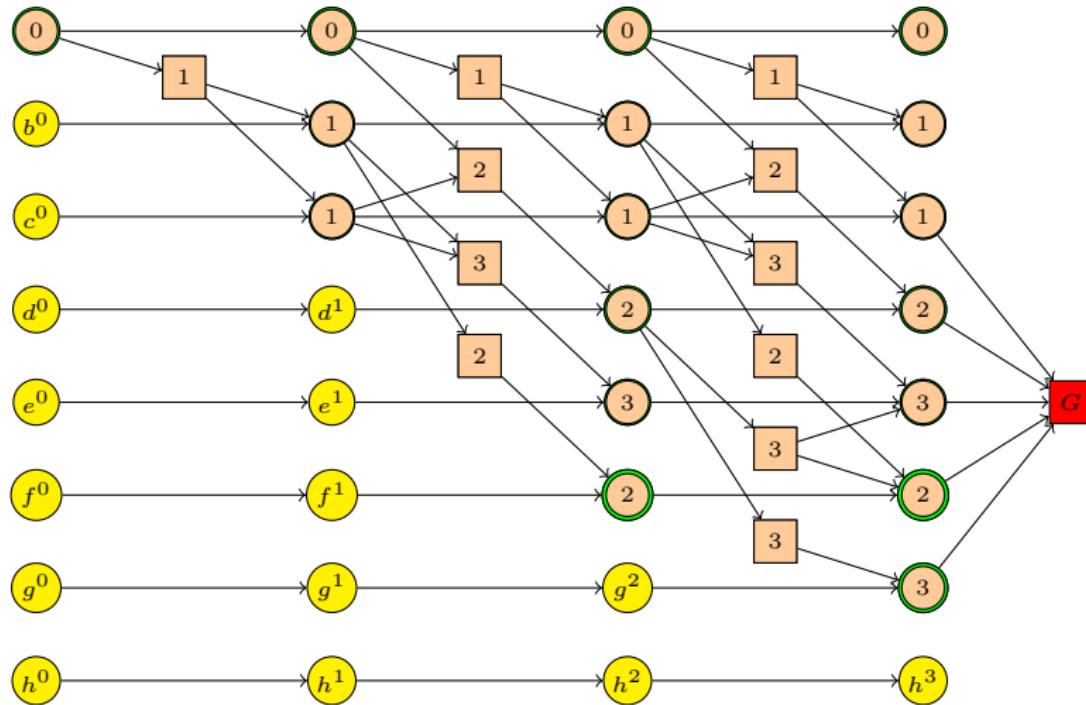
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

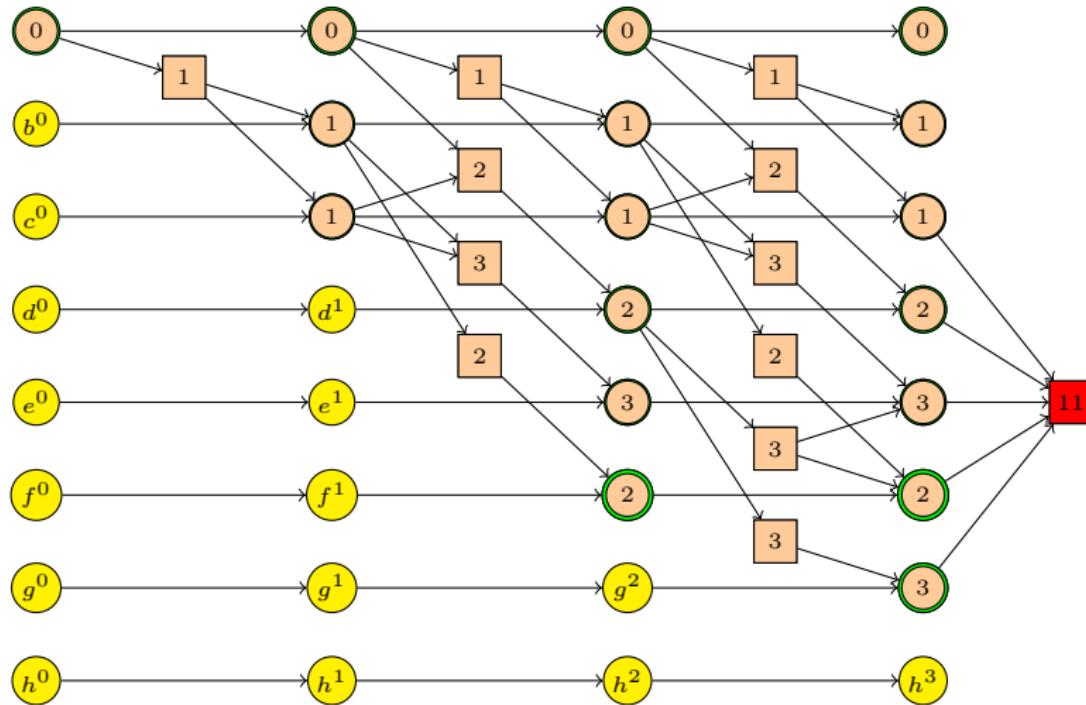
$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{\text{add}}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\text{max}}$

$h_{\text{add}}$

$h_{\text{FF}}$

Comparison &  
practice

# Remarks on $h_{\text{add}}$

- $h_{\text{add}}$  is **safe** and **goal-aware**.
- Unlike  $h_{\text{max}}$ ,  $h_{\text{add}}$  is a **very informative** heuristic in many planning domains.
- Q: Intuitively, when it will be informative? ↪
- The price for this is that it is **not admissible** (and hence also **not consistent**), so not suitable for optimal planning.
- In fact, it **almost always** overestimates the  $h^+$  value because it does not take **positive interactions** into account.

הערך המינימלי של  $h_{\text{add}}$  יהיה שווה לערך  $h_{\text{max}}$ .  
ולוק גיילן, נמר עלינו מאר השינויים מילוי.  
הערך המינימלי של  $h_{\text{add}}$  יהיה שווה לערך  $h_{\text{max}}$ .

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\text{max}}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# FF heuristic: fitting the template

fast forward

## The FF heuristic $h_{FF}$

### Computing annotations:

- Annotations are **Boolean values**, computed top-down.

A node is **marked** when its annotation is set to 1 and **unmarked** if it is set to 0. Initially, the goal node is marked, and all other nodes are unmarked.

We say that an action node is **justified** if all its true immediate predecessors are marked, and that a proposition node is **justified** if at least one of its immediate predecessors is marked.

...

רוויאיון ב-  
הACTION ו-  
הPROPOSITION  
.goal ->

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

$h_{\max}$

$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice

# FF heuristic: fitting the template (ctd.)

## The FF heuristic $h_{\text{FF}}$ (ctd.)

Computing annotations:

- ...

Apply these rules until **all marked nodes are justified**:

- ① Mark all immediate predecessors of a marked unjustified ACTION node.
- ② Mark the immediate predecessor of a marked unjustified PROP node with only one immediate predecessor.
- ③ Mark an immediate predecessor of a marked unjustified PROP node connected via an idle arc.
- ④ Mark any immediate predecessor of a marked unjustified PROP node.

The rules are given in priority order: earlier rules are preferred if applicable.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h^{\max}$   
 $h^{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# FF heuristic: fitting the template (ctd.)

## The FF heuristic $h_{\text{FF}}$ (ctd.)

Termination criterion:

- Always terminate at first layer where goal node is true.

Heuristic value:

- The heuristic value is the number of marked action nodes.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

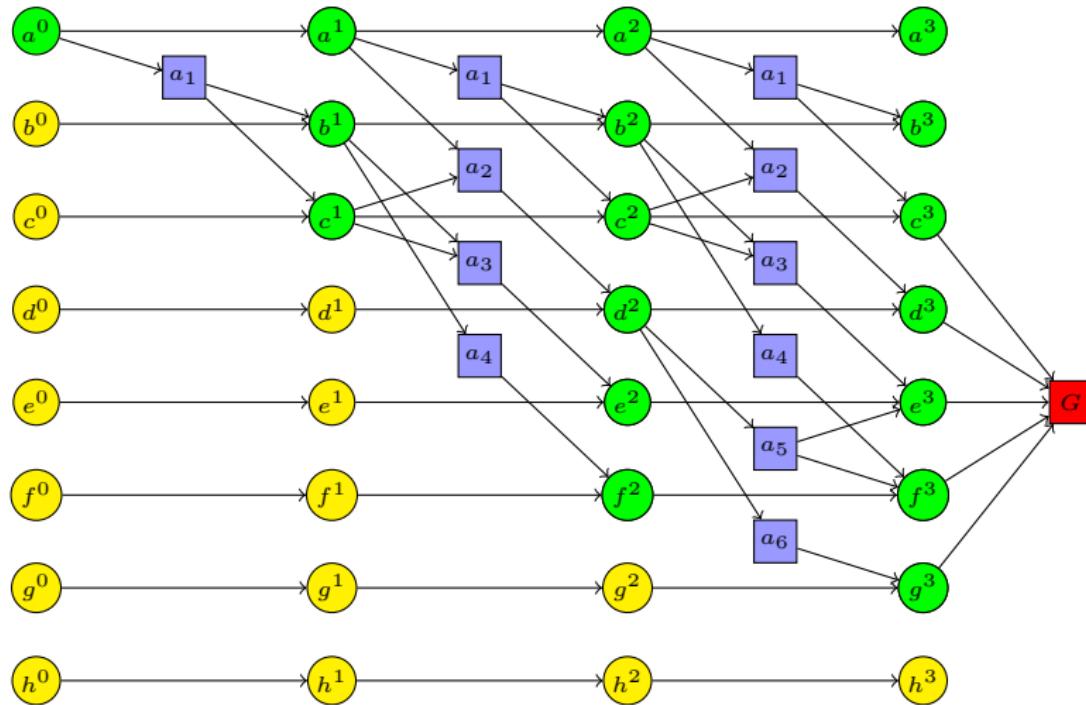
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h^{\max}$   
 $h^{\text{add}}$   
 $h_{\text{FF}}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

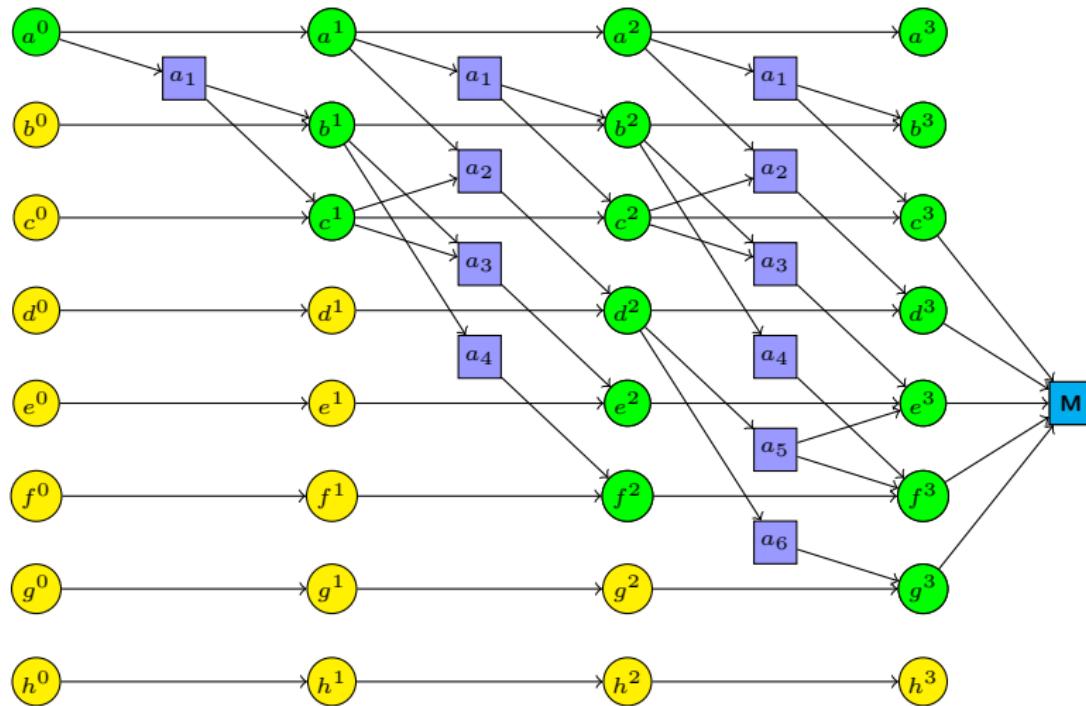
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$   
Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template

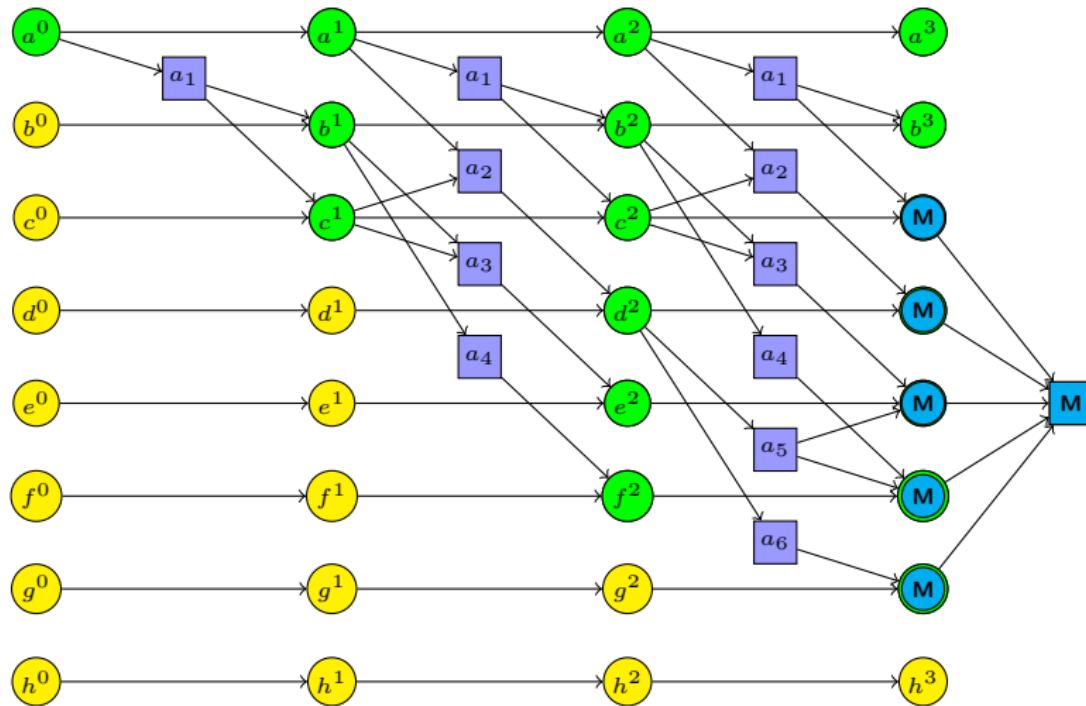
$h_{\max}$

$h_{\text{add}}$

$h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

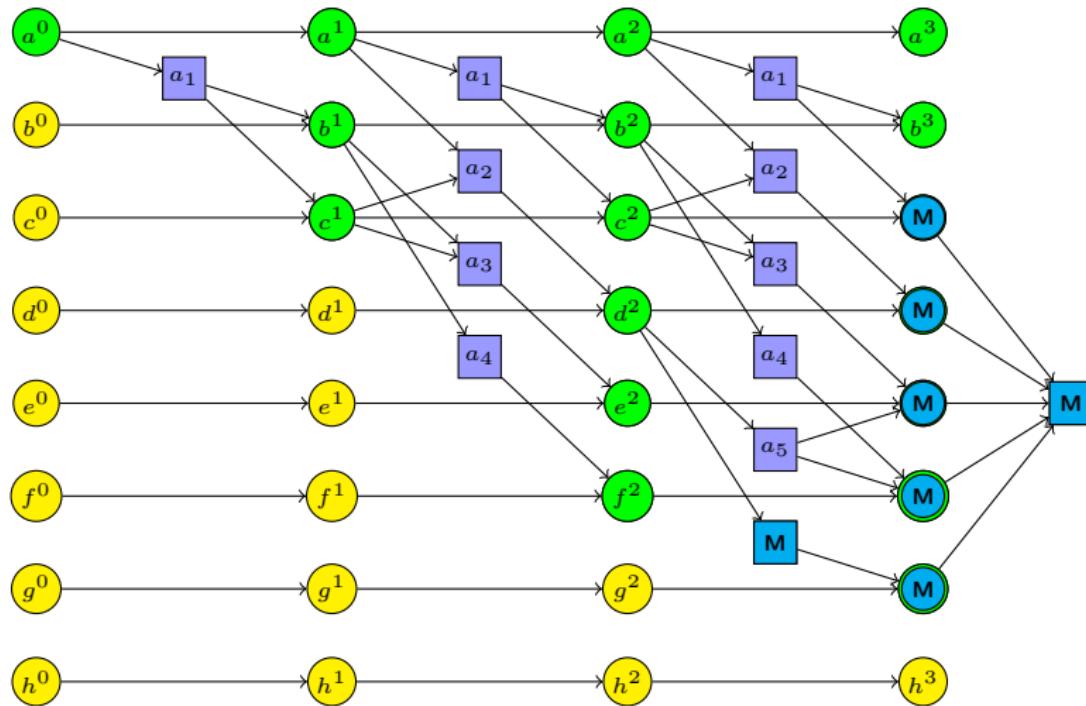
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

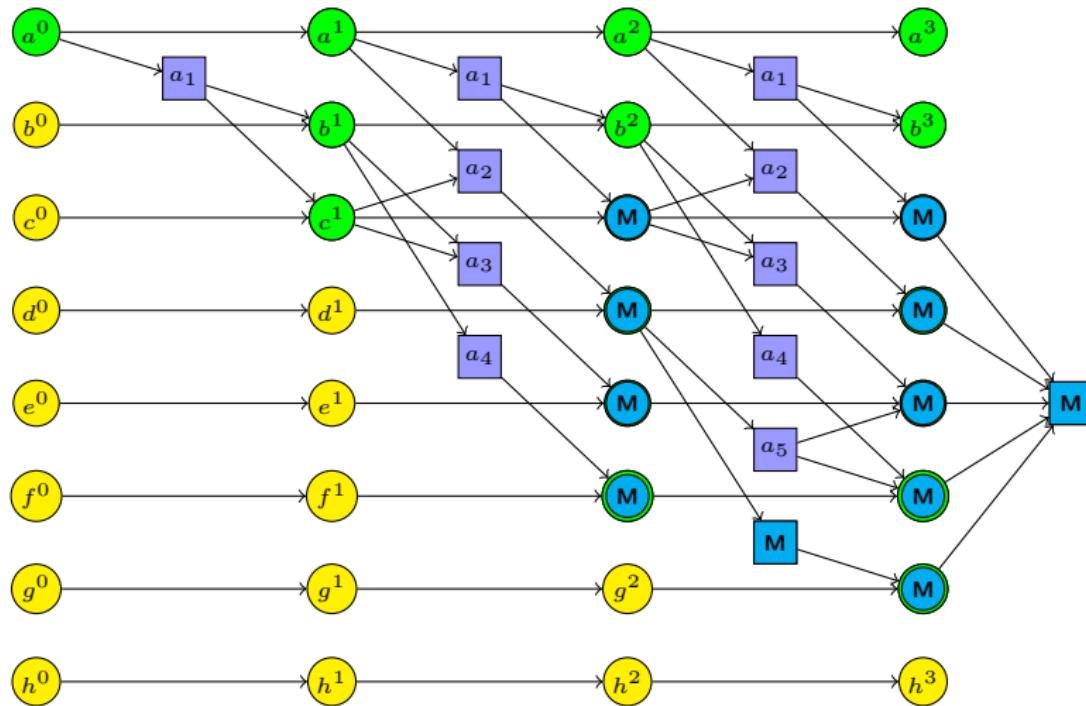
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

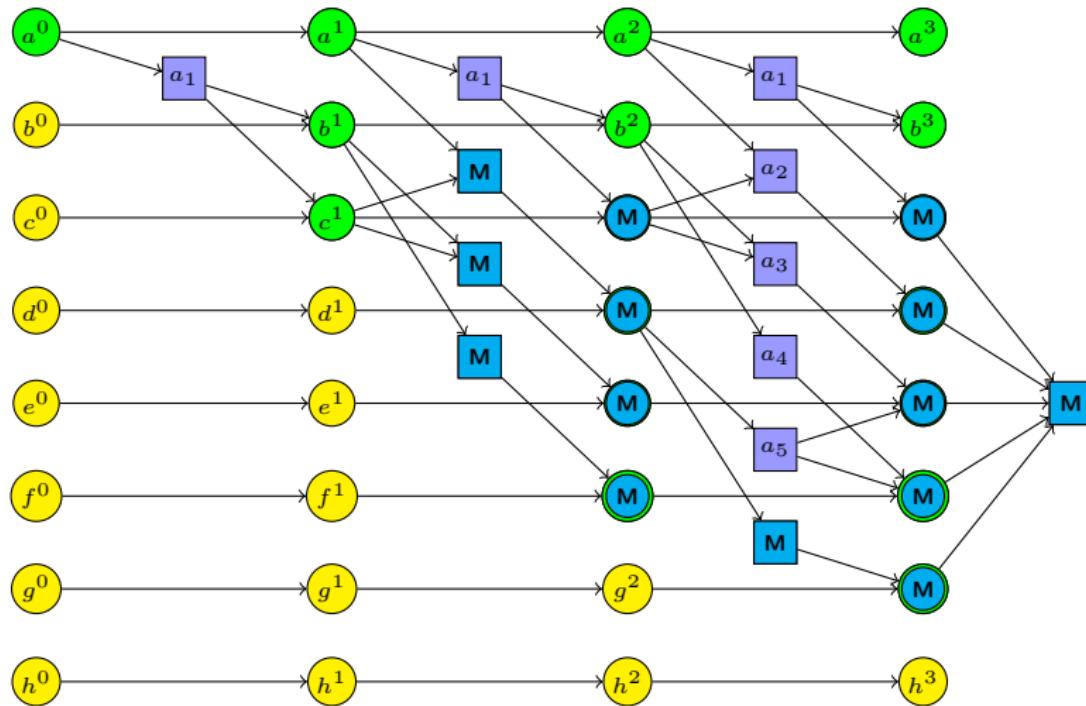
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

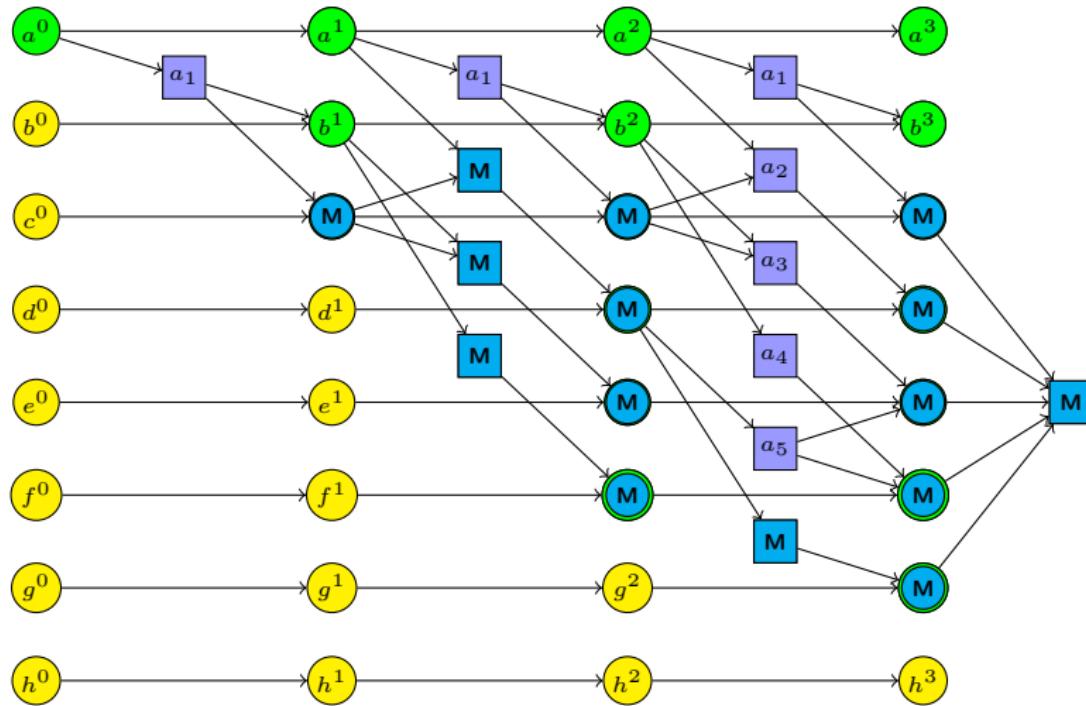
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

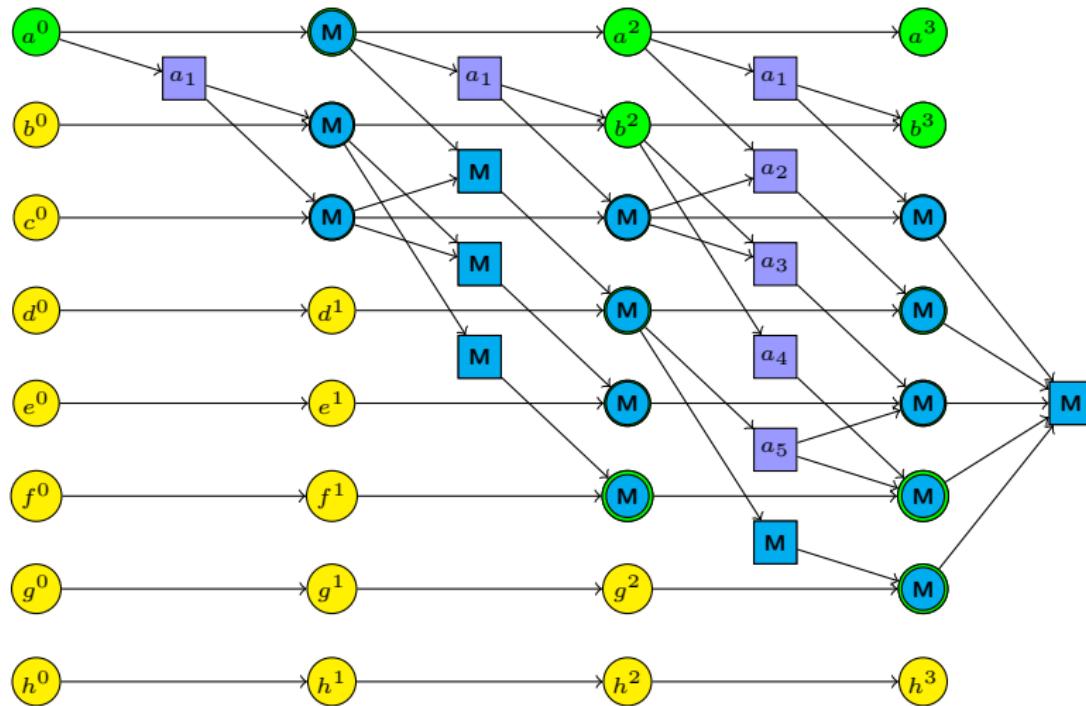
Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$   
Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

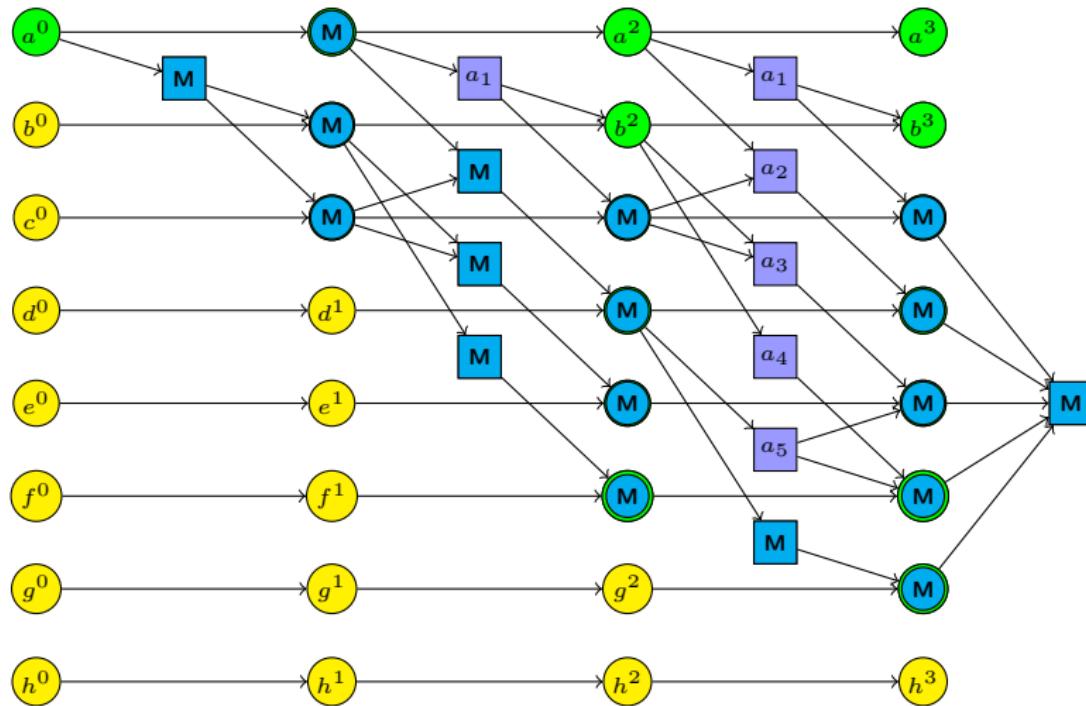
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

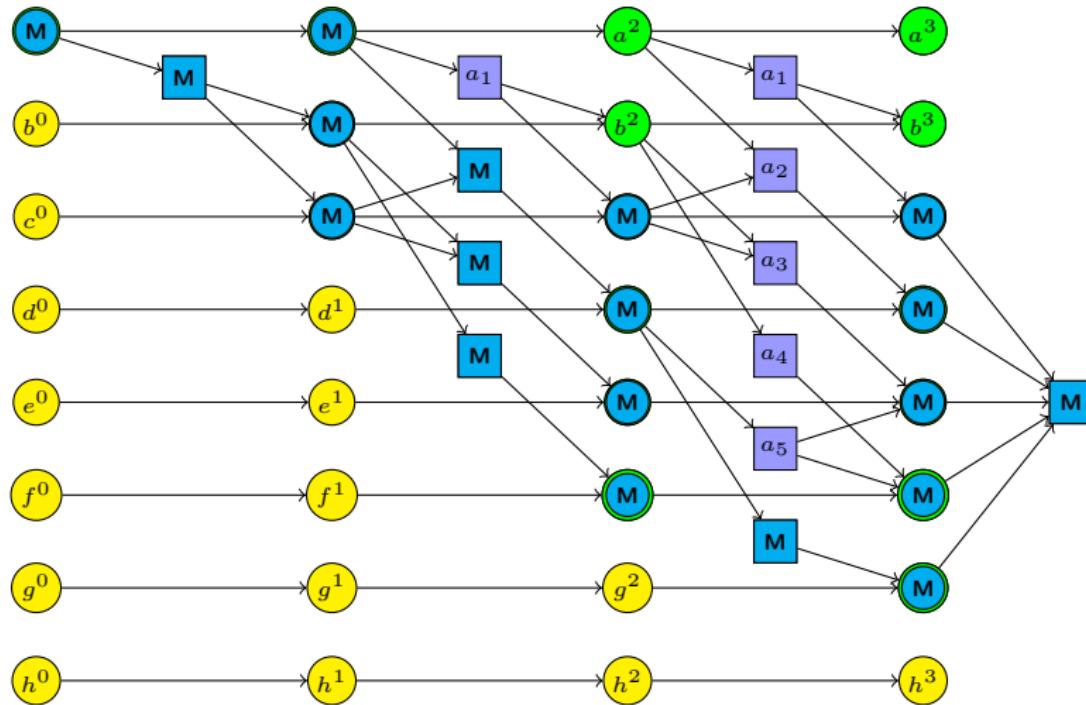
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

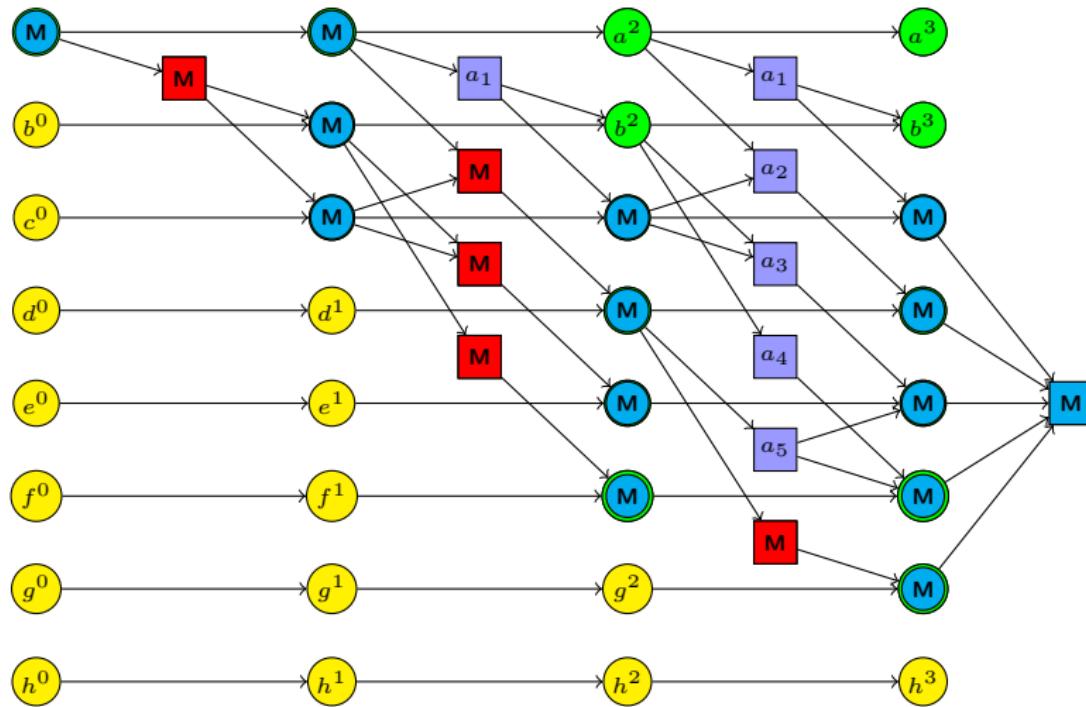
Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Running example: $h_{FF}$



Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{FF}$

Comparison &  
practice

# Remarks on $h_{\text{FF}}$

- Like  $h_{\text{add}}$ ,  $h_{\text{FF}}$  is **safe** and **goal-aware**, but neither **admissible** nor **consistent**.
- Always more accurate than  $h_{\text{add}}$  with respect to  $h^+$ .
  - Marked actions define a **relaxed plan**.
- $h_{\text{FF}}$  can be computed in **linear time**.
  - The  $h_{\text{FF}}$  value depends on tie-breaking when the marking rules allow several possible choices, so  $h_{\text{FF}}$  is **not well-defined** without specifying the tie-breaking rule.
  - The best implementations of FF use additional rules of thumb to try to reduce the size of the generated relaxed plan.

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h^{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

# Comparison of relaxation heuristics

Introduction  
to AI

C. Domshlak

Introduction

Problem  
specification

Obtaining  
heuristics

Concrete  
Heuristics

Template  
 $h_{\max}$   
 $h_{\text{add}}$   
 $h_{\text{FF}}$   
Comparison &  
practice

## Relationship between relaxation heuristics

Let  $s$  be a state of planning task  $\langle P, I, O, G \rangle$ . Then:

- $h_{\max}(s) \leq h^+(s) \leq h^*(s)$
- $h_{\max}(s) \leq h^+(s) \leq h_{\text{FF}}(s) \leq h_{\text{add}}(s)$
- $h^*$  and  $h_{\text{FF}}$  are pairwise incomparable
- $h^*$  and  $h_{\text{add}}$  are incomparable

Moreover,  $h^+$ ,  $h_{\max}$ ,  $h_{\text{add}}$ , and  $h_{\text{FF}}$  assign  $\infty$  to the same set of states.

כertain cases הינה יתנו

Note: For **inadmissible** heuristics, dominance is in general neither desirable nor undesirable. For relaxation heuristics, the objective is usually to get as close to  $h^+$  as possible.

כרגעו מטרת  $h^+$  דה

# Introduction to AI

Stochastic Planning: Markov Decision Processes

Erez Karpas  
Slides by Carmel Domshlak

IE&M — Technion

# Classical deterministic planning

- dynamics: **deterministic**, nondeterministic or probabilistic
  - observability: **full**, partial or **none**
  - horizon: **finite** or infinite
  - ...
- ➊ **classical planning**
  - ➋ conformant planning
  - ➌ conditional planning with full observability
  - ➍ conditional planning with partial observability
  - ➎ Markov decision processes (MDP)
  - ➏ partially observable MDPs (POMDP)

# Stochastic (decision-theoretic) planning

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- dynamics: deterministic, nondeterministic or **probabilistic**
  - observability: **full**, partial or none
  - horizon: **finite** or **infinite**
  - ...
- 
- ➊ classical planning
  - ➋ conformant planning
  - ➌ conditional planning with full observability
  - ➍ conditional planning with partial observability
  - ➎ **Markov decision processes (MDP)**
  - ➏ partially observable MDPs (POMDP)

# Markov Decision Processes

## Transition systems

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

### Definition (MDP)

An **MDP** is  $\langle S, A, R, T \rangle$  where

- $S$  is a finite set of **states** ( $|S| = n$ )
- $A$  is a finite set of **actions** ( $|A| = m$ )
- **transition function**  $Tr : S \times A \times S \mapsto [0, 1]$  with  
 $Tr(s, a, s') = P(s' | s, a) \xrightarrow{\text{sum}} \sum_{s' \in S} P(s'|s,a) = 1$ 
  - Probability of going to state  $s'$  after taking action  $a$  in state  $s$
  - How many parameters does it take to represent?  $m \cdot n^2$
- bounded, real-valued **reward function**  $R : S \mapsto [0, r_{\max}]$ 
  - $R(s)$  is immediate reward we get for reaching state  $s$

# Markov Decision Processes

## Objective

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

### Objection (informally)

- select actions in order to **maximize total reward**
- example: in a goal-based domain,

$$R(s) = \begin{cases} 1, & s \text{ is a goal state} \\ 0, & \text{otherwise} \end{cases}$$

# Assumptions

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- **Markovian dynamics** (history independence)
  - Next state only depends on current state and action
$$P(S^{t+1} | A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = P(S^{t+1} | A^t, S^t)$$
- **Markovian reward process**
  - Reward only depends on current state
$$P(R^t | A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = P(R^t | S^t)$$
  - We assume deterministic rewards
- **Stationary** (time independent) **dynamics**
  - The world dynamics do not depend on the absolute time
$$P(S^{t+1} | A^t, S^t) = P(S^{k+1} | A^k, S^k) \text{ for all } t, k$$
- **Full observability**
  - Though we can't predict exactly which state we will reach when we execute an action, once it is executed, we know what the state is

# Assumptions

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- **Markovian dynamics** (history independence)
  - Next state only depends on current state and action
$$P(S^{t+1} | A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = P(S^{t+1} | A^t, S^t)$$
- **Markovian reward process**
  - Reward only depends on current state
$$P(R^t | A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = P(R^t | S^t)$$
  - We assume deterministic rewards
- **Stationary** (time independent) **dynamics**
  - The world dynamics do not depend on the absolute time
$$P(S^{t+1} | A^t, S^t) = P(S^{k+1} | A^k, S^k) \text{ for all } t, k$$
- **Full observability**
  - Though we can't predict exactly which state we will reach when we execute an action, once it is executed, we know what the state is

# Assumptions

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- **Markovian dynamics** (history independence)
  - Next state only depends on current state and action
$$P(S^{t+1} | A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = P(S^{t+1} | A^t, S^t)$$
- **Markovian reward process**
  - Reward only depends on current state
$$P(R^t | A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = P(R^t | S^t)$$
  - We assume deterministic rewards
- **Stationary** (time independent) **dynamics**
  - The world dynamics do not depend on the absolute time
$$P(S^{t+1} | A^t, S^t) = P(S^{k+1} | A^k, S^k) \text{ for all } t, k$$
- **Full observability**
  - Though we can't predict exactly which state we will reach when we execute an action, once it is executed, we know what the state is

# Assumptions

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- **Markovian dynamics** (history independence)
  - Next state only depends on current state and action
$$P(S^{t+1} | A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = P(S^{t+1} | A^t, S^t)$$
- **Markovian reward process**
  - Reward only depends on current state
$$P(R^t | A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = P(R^t | S^t)$$
  - We assume deterministic rewards
- **Stationary** (time independent) **dynamics**
  - The world dynamics do not depend on the absolute time
$$P(S^{t+1} | A^t, S^t) = P(S^{k+1} | A^k, S^k) \text{ for all } t, k$$
- **Full observability**
  - Though we can't predict exactly which state we will reach when we execute an action, once it is executed, we know what the state is

# Assumptions

- **Markovian dynamics** (history independence)
  - Next state only depends on current state and action
$$P(S^{t+1} | A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = P(S^{t+1} | A^t, S^t)$$
- **Markovian reward process**
  - Reward only depends on current state
$$P(R^t | A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = P(R^t | S^t)$$
  - We assume deterministic rewards
- **Stationary** (time independent) **dynamics**
  - The world dynamics do not depend on the absolute time
$$P(S^{t+1} | A^t, S^t) = P(S^{k+1} | A^k, S^k) \text{ for all } t, k$$
- **Full observability**
  - Though we can't predict exactly which state we will reach when we execute an action, once it is executed, we know what the state is

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

# Solution Concept

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- Classical planning  $\leadsto$  sequence of actions  $\langle a_1, a_2, \dots, a_k \rangle$
- For MDPs, action sequences are insufficient
  - Actions have stochastic effects, so the state we end up in is uncertain
  - We might end up in states where the remainder of the action sequence is not attractive (or even doesn't apply)
  - A solution should tell us **what the best action is for any possible situation that might arise**

# Policies (“plans” for MDPs)

- A solution to an MDP is a **policy**
  - Two types of policies: **nonstationary** and **stationary**
- Nonstationary (NS) policies are used when we are given a finite planning horizon  $H$ 
  - We are told how many actions we will be allowed to take
- NS policy is a function from states and times to actions
  - $\pi : S \times T \mapsto A$ , where  $T$  is the non-negative integers
  - $\pi(s, t)$  tells us what action to take at state  $s$  when there are  $t$  stages-to-go

# Policies (“plans” for MDPs)

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- What if we want to continue taking actions indefinitely?
- A stationary policy is a function from states to actions
  - $\pi : S \mapsto A$
  - $\pi(s)$  is action to take at state  $s$  (regardless of time)
  - specifies a continuously reactive controller
- Note that both nonstationary and stationary policies assume or have these properties:
  - full observability
  - history independence
  - deterministic action choice

# Policies (“plans” for MDPs)

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- What if we want to continue taking actions indefinitely?
- A stationary policy is a function from states to actions
  - $\pi : S \mapsto A$
  - $\pi(s)$  is action to take at state  $s$  (regardless of time)
  - specifies a continuously reactive controller
- Note that both nonstationary and stationary policies assume or have these properties:
  - full observability
  - history independence
  - deterministic action choice

# Value of a Policy

- How good is a policy  $\pi$ ?
  - How do we measure reward “accumulated by  $\pi$ ”
- **Value function**  $V : S[\times T] \mapsto \mathbb{R}$  associates value with each state (or each state and time for non-stationary  $\pi$ )
- $V_\pi(s)$  denotes **value** of policy  $\pi$  at state  $s$ 
  - Depends on immediate reward, but also what you achieve subsequently by following  $\pi$
  - An **optimal policy** is one that is no worse than any other policy at any state
- The goal of MDP planning is to compute an optimal policy (method depends on how we define value)

# Finite-Horizon Value Functions

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

We first consider maximizing total reward over a finite horizon

- Assumes the agent has  $H$  time steps to live
- To act optimally, should the agent use a stationary or non-stationary policy?
  - If you had only one week to live would you act the same way as if you had fifty years to live?

# Finite-Horizon Problems

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- Value (utility) depends on stage-to-go
  - use a nonstationary policy

מצב קיומו של תוארו  
בזמן?

- $V_{\pi}^k(s)$  is  $k$ -stage-to-go value function of  $\pi$

- expected total reward for executing  $\pi$  starting in  $s$  for  $k$  time steps

$t$  הינו *Reward*

$$V_{\pi}^k(s) = \mathbb{E} \left[ \sum_{t=0}^k R_t \mid \pi, s \right]$$

כזה מוגדר *הערך אפקטיבי*  $t$

like Reward -> price of action =  
the price of action  $\rightarrow$   $\pi$  is a policy

$$= \mathbb{E} \left[ \sum_{t=0}^k R(s_t) \mid a_t = \pi(s_t, k-t), s_0 = s \right]$$

*MOPt Policy* =  
הערך אפקטיבי

- $R_t$  and  $s_t$  are *random variables*

# Computational Problems

## Policy Evaluation

Given an MDP, a nonstationary policy  $\pi$ , and  $t \in T$ , compute finite-horizon value function  $V_\pi^t(s)$ .

## Policy Optimization

Given an MDP, and a horizon  $H$ , compute an optimal finite-horizon policy  $\pi^*$

- How many finite-horizon policies are there?  $|A|^{Hn}$
- Simple enumeration will not work ...
- We'll show that policy optimization is equivalent to computing "optimal value function"

# Computational Problems

## Policy Evaluation

Given an MDP, a nonstationary policy  $\pi$ , and  $t \in T$ , compute finite-horizon value function  $V_\pi^t(s)$ .

## Policy Optimization

Given an MDP, and a horizon  $H$ , compute an optimal finite-horizon policy  $\pi^*$

infinitely many policies

- How many finite-horizon policies are there?  $|A|^{Hn}$
- Simple enumeration will not work ...
- We'll show that policy optimization is equivalent to computing "optimal value function"

## Finite-Horizon Policy Evaluation

- Can use dynamic programming to compute  $V_\pi^t(s)$   
 $\leadsto$  Markov property is critical for this

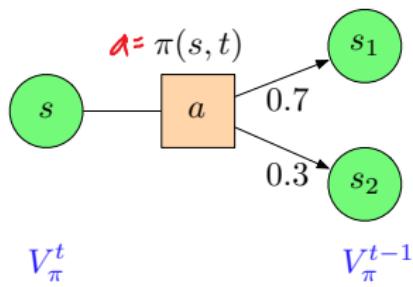
# Introduction to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

$\forall s \in S :$

$$V_\pi^0(s) = R(s)$$

$$V_\pi^t(s) = R(s) + \sum P(s' | s, \pi(s, t)) \cdot V_\pi^{t-1}(s')$$



בנין ה- $s'$  הוא תוצאה של הסדרת הפעולות  $s$ .

# Policy Optimization: Bellman Backups

How can we compute  $V_{\pi^*}^t(s)$  given  $V_{\pi^*}^{t-1}(s)$ ?

Introduction  
to AI

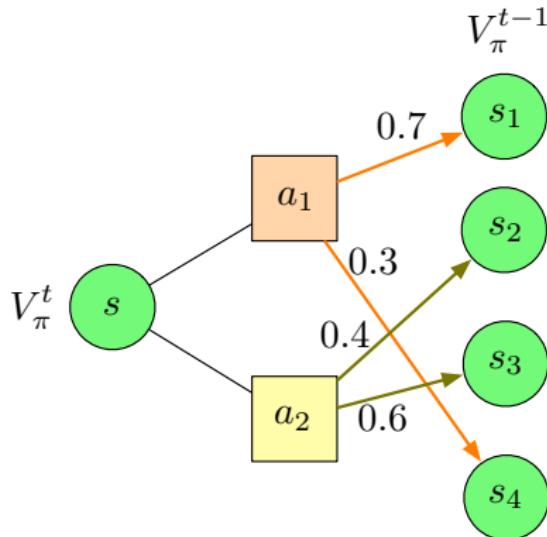
Erez Karpas  
Slides by  
Carmel  
Domshlak

# Policy Optimization: Bellman Backups

How can we compute  $V_{\pi^*}^t(s)$  given  $V_{\pi^*}^{t-1}(s)$ ?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

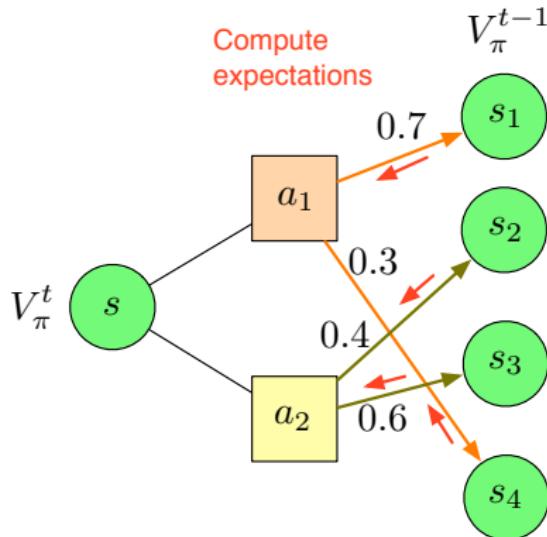


# Policy Optimization: Bellman Backups

How can we compute  $V_{\pi^*}^t(s)$  given  $V_{\pi^*}^{t-1}(s)$ ?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

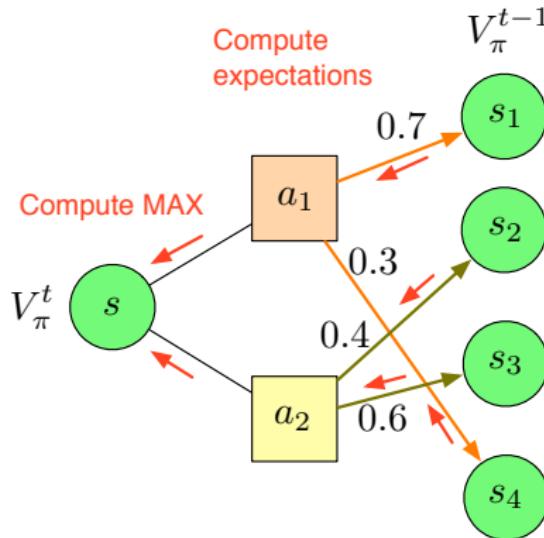


# Policy Optimization: Bellman Backups

How can we compute  $V_{\pi^*}^t(s)$  given  $V_{\pi^*}^{t-1}(s)$ ?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak



# Value Iteration: Finite Horizon Case

Markov property allows exploitation of DP principle for optimal policy construction

- no need to enumerate  $|A|^{Hn}$  possible policies!

## Value Iteration

$\forall s \in S :$

$$V_{\pi^*}^0(s) = R(s)$$

$$V_{\pi^*}^t(s) = R(s) + \max_a \sum_{s'} P(s' | s, a) \cdot V_{\pi^*}^{t-1}(s')$$

# Value Iteration: Finite Horizon Case

Markov property allows exploitation of DP principle for optimal policy construction

- no need to enumerate  $|A|^{Hn}$  possible policies!

## Value Iteration

$\forall s \in S :$

$$V_{\pi^*}^0(s) = R(s)$$

$$V_{\pi^*}^t(s) = R(s) + \max_a \sum_{s'} P(s' | s, a) \cdot V_{\pi^*}^{t-1}(s')$$

$$\pi^*(s, t) = \arg \max_a \sum_{s'} P(s' | s, a) \cdot V_{\pi^*}^{t-1}(s')$$

# Value Iteration: Finite Horizon Case

Markov property allows exploitation of DP principle for optimal policy construction

- no need to enumerate  $|A|^{Hn}$  possible policies!

## Value Iteration

$\forall s \in S :$

$$V_{\pi^*}^0(s) = R(s)$$

$$V_{\pi^*}^t(s) = R(s) + \max_a \sum_{s'} P(s' | s, a) \cdot V_{\pi^*}^{t-1}(s')$$

$$\pi^*(s, t) = \arg \max_a \sum_{s'} P(s' | s, a) \cdot V_{\pi^*}^{t-1}(s')$$

$V_{\pi^*}^t(s)$  is optimal  $t$ -stage-to-go value function

$\pi^*(s, t)$  is optimal  $t$ -stage-to-go policy

# Value Iteration: Complexity

- DP: optimal solution to  $t - 1$  stage problem can be used without modification as part of optimal solution to  $t$ -stage problem
- Complexity?
  - $H$  iterations
  - At each iteration, each of  $n$  states, computes expectation for  $|A|$  actions
  - Each expectation takes  $O(n)$  time
- Total time complexity:  $O(H|A|n^2)$ 
  - Polynomial in the number of states. Is that good?

# Value Iteration: Complexity

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- DP: optimal solution to  $t - 1$  stage problem can be used without modification as part of optimal solution to  $t$ -stage problem
- Complexity?
  - $H$  iterations
  - At each iteration, each of  $n$  states, computes expectation for  $|A|$  actions
  - Each expectation takes  $O(n)$  time
- Total time complexity:  $O(H|A|n^2)$ 
  - Polynomial in the number of states. Is that good?

הזמן אונליין

# Summary: Finite Horizon

- Value Iteration provides us with an optimal policy

$$V_{\pi^*}^t(s) \geq V_\pi^t(s), \quad \forall \pi, s, t$$

~ convince yourself (prove) by induction on  $t$

- Note: optimal value function is unique, but optimal policy is not

# Discounted Infinite Horizon MDPs

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- Defining value as total reward is problematic with infinite horizons
  - many or all policies have infinite expected reward
  - some MDPs are ok (e.g., zero-cost absorbing states)
- “Trick”: introduce **discount factor**  $0 \leq \gamma < 1$ 
  - future rewards discounted by  $\gamma$  per time step

$$V_\pi(s) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_t \mid \pi, s \right]$$

- Why? The value gets **bounded**.

$$V_\pi(s) \leq \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_{max} \right] = \frac{1}{1-\gamma} R_{max}$$

- Motivation: economic? probability of death? convenience?

# Discounted Infinite Horizon MDPs

- Defining value as total reward is problematic with infinite horizons
  - many or all policies have infinite expected reward
  - some MDPs are ok (e.g., zero-cost absorbing states)
- “Trick”: introduce **discount factor**  $0 \leq \gamma < 1$ 
  - future rewards discounted by  $\gamma$  per time step

$$V_{\pi}(s) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_t \mid \pi, s \right]$$

- Why? The value gets **bounded**.

$$V_{\pi}(s) \leq \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_{max} \right] = \frac{1}{1 - \gamma} R_{max}$$

- Motivation: economic? probability of death? convenience?

# Notes: Discounted Infinite Horizon

- Optimal policy maximizes value at each state
- Optimal policies guaranteed to exist! (Howard, 1960)
- Furthermore there is always an **optimal stationary policy**
  - Intuition: why would we change action at  $s$  at a new time when there is always forever ahead?
  - We define  $V^*(s) = V_{\pi^*}(s)$  for some optimal policy  $\pi^*$

↓  
וינטואיזציה  
וינטואיזציה  
 $V^*(s) = V_{\pi^*}(s)$

# Policy Evaluation

- Value equation for **fixed** policy
  - Immediate reward + Expected discounted future reward

$$V_{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) \cdot V_{\pi}(s')$$

Bellman update

- How can we compute  $V_{\pi}$ ?

# Policy Evaluation

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- Value equation for **fixed** policy

- Immediate reward + Expected discounted future reward

$$V_{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) \cdot V_{\pi}(s')$$

- How can we compute  $V_{\pi}$ ?

- we are given  $R$  (rewards) and  $P$  (action dynamics)
  - **linear system** with  $n$  variables and  $n$  constraints
    - variables:  $V_{\pi}(s_1), \dots, V_{\pi}(s_n)$
    - constraints: one value equation (above) per state
  - use linear algebra to solve this system and get  $V_{\pi}$

# Computing an Optimal Value Function

- Bellman equation for optimal value function

$$V^*(s) = R(s) + \gamma \cdot \max_a \sum_{s'} P(s' | s, a) \cdot V^*(s')$$

- Bellman proved this is always true for an optimal value function

# Computing an Optimal Value Function

- Bellman equation for optimal value function

$$V^*(s) = R(s) + \gamma \cdot \max_a \sum_{s'} P(s' | s, a) \cdot V^*(s')$$

- Bellman proved this is always true for an optimal value function
- How can we compute  $V^*$ ?
  - The MAX operator makes the system non-linear, so the problem is more difficult than policy evaluation

# Computing an Optimal Value Function

- Bellman equation for optimal value function

$$V^*(s) = R(s) + \gamma \cdot \max_a \sum_{s'} P(s' | s, a) \cdot V^*(s')$$

- Bellman proved this is always true for an optimal value function
- How can we compute  $V^*$ ?

- The MAX operator makes the system non-linear, so the problem is more difficult than policy evaluation

- Notice that the optimal value function is a **fixed-point** of the **Bellman Backup** operator  $\mathcal{B}$  (i.e.  $\mathcal{B}[V^*] = V^*$ )

$$\mathcal{B}[V](s) = R(s) + \gamma \cdot \max_a \sum_{s'} P(s' | s, a) \cdot V(s')$$

# Value Iteration

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- Can compute optimal policy using value iteration based on Bellman backups, just like finite-horizon problems (but include discount term)

$$V^0(s) = R(s)$$

$$V^t(s) = R(s) + \gamma \cdot \max_a \sum_{s'} P(s' | s, a) \cdot V^{t-1}(s')$$

- Converges to optimal value function? YES

$$\lim_{k \rightarrow \infty} V^k = V^*$$

- When should we stop in practice?

# Convergence

- Bellman Backup  $\mathcal{B}$  is a **contraction operator** on value functions
  - For any  $V$  and  $V'$ ,  $\|\mathcal{B}[V] - \mathcal{B}[V']\| \leq \gamma \cdot \|V - V'\|$ 
    - $\|V\|$  is the max-norm:  $\|(0.1, 100, 5, 12)\| = 100$
  - So applying a Bellman backup to any two value functions causes them to get closer together in the max-norm sense!
- Convergence is assured from *any*  $V$ 
$$\|V^* - \mathcal{B}[V]\| = \|\mathcal{B}[V^*] - \mathcal{B}[V]\| \leq \gamma \cdot \|V^* - V\|$$
$$\Rightarrow \lim_{k \rightarrow \infty} V^k = V^*$$
- When to stop in practice? When  $\|V^k - V^{k-1}\| \leq \epsilon$ 
  - ensures  $\|V^* - V^k\| \leq \frac{\epsilon\gamma}{1-\gamma}$  (be able to verify that)
  - so stop when  $V^k$  and  $V^{k-1}$  are similar enough

# Convergence

- Bellman Backup  $\mathcal{B}$  is a **contraction operator** on value functions
  - For any  $V$  and  $V'$ ,  $||\mathcal{B}[V] - \mathcal{B}[V']|| \leq \gamma \cdot ||V - V'||$ 
    - $||V||$  is the max-norm:  $||(0.1, 100, 5, 12)|| = 100$
  - So applying a Bellman backup to any two value functions causes them to get closer together in the max-norm sense!
- Convergence is assured from *any*  $V$ 
$$||V^* - \mathcal{B}[V]|| = ||\mathcal{B}[V^*] - \mathcal{B}[V]|| \leq \gamma \cdot ||V^* - V||$$
$$\Rightarrow \lim_{k \rightarrow \infty} V^k = V^*$$
- When to stop in practice? When  $||V^k - V^{k-1}|| \leq \epsilon$ 
  - ensures  $||V^* - V^k|| \leq \frac{\epsilon\gamma}{1-\gamma}$  (be able to verify that)
  - so stop when  $V^k$  and  $V^{k-1}$  are similar enough

# Convergence

- Bellman Backup  $\mathcal{B}$  is a **contraction operator** on value functions
  - For any  $V$  and  $V'$ ,  $\|\mathcal{B}[V] - \mathcal{B}[V']\| \leq \gamma \cdot \|V - V'\|$ 
    - $\|V\|$  is the max-norm:  $\|(0.1, 100, 5, 12)\| = 100$
  - So applying a Bellman backup to any two value functions causes them to get closer together in the max-norm sense!
- Convergence is assured from *any*  $V$ 
$$\|V^* - \mathcal{B}[V]\| = \|\mathcal{B}[V^*] - \mathcal{B}[V]\| \leq \gamma \cdot \|V^* - V\|$$
$$\Rightarrow \lim_{k \rightarrow \infty} V^k = V^*$$
- When to stop in practice? When  $\|V^k - V^{k-1}\| \leq \epsilon$ 
  - ensures  $\|V^* - V^k\| \leq \frac{\epsilon\gamma}{1-\gamma}$  (be able to verify that)
  - so stop when  $V^k$  and  $V^{k-1}$  are similar enough

# How to Act?

- Given a  $V^k$  from value iteration that closely approximates  $V^*$ , what should we use as our policy?
- Use **greedy** policy (one step lookahead):

$$\text{greedy}[V^k](s) = \arg \max_a \sum_{s'} P(s' | s, a) \cdot V^k(s')$$

- Note: Value of greedy policy may not be equal to  $V^k$
- How close is  $V^{\text{greedy}}$  to  $V^*$ ?
  - if  $V^k$  is  $\epsilon$ -close to  $V^*$ , then  $V^{\text{greedy}}$  is  $\frac{2\epsilon\gamma}{1-\gamma}$ -close to  $V^*$

# How to Act?

- Given a  $V^k$  from value iteration that closely approximates  $V^*$ , what should we use as our policy?
- Use **greedy** policy (one step lookahead):

$$\text{greedy}[V^k](s) = \arg \max_a \sum_{s'} P(s' | s, a) \cdot V^k(s')$$

- Note: Value of greedy policy may not be equal to  $V^k$
- How close is  $V^{\text{greedy}}$  to  $V^*$ ?
  - if  $V^k$  is  $\epsilon$ -close to  $V^*$ , then  $V^{\text{greedy}}$  is  $\frac{2\epsilon\gamma}{1-\gamma}$ -close to  $V^*$

# Optimization via Policy Iteration

- Given fixed policy, can compute its value exactly:

$$V_\pi(s) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_t \mid \pi, s \right]$$

- **Policy Iteration** exploits this: iterates steps of policy evaluation and policy improvement

- ### ① Choose a random policy $\pi$

Digitized by srujanika@gmail.com

ב-הנָּסִים, וְעַל-אֶת-הַגְּדוֹלָה, כִּי-כֵן

- ## ② Loop:

- ### ① Evaluate $V_\pi$

רְשָׁוָאֵת וְעַמְּדָה בְּבִנְיָמִן

- ② For each  $s$ , set  $\pi'(s) := \arg \max_a \sum_{s'} P(s' | s, a) \cdot V_\pi(s')$
  - ③ Replace  $\pi$  with  $\pi'$

Until no improving action possible at any state

•π මුළු පිළිව නැංවාම හේදුවේ ගිවිසුද්ධිය

# Policy Iteration Notes

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- Each step of policy iteration is guaranteed to strictly improve the policy at some state when improvement is possible
- Convergence assured (Howard)
  - no local maxima in value space, and each policy must improve value;
  - since finite number of policies, will converge to optimal policy
- Gives exact value of optimal policy

# Value Iteration vs. Policy Iteration

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

- Which is faster: VI or PI?
  - depends on the problem instance
- VI takes more iterations than PI, but PI requires more time on each iteration
  - PI must perform policy evaluation on each step which involves solving a linear system
- Complexity
  - There are at most  $\exp(n)$  policies, so PI is no worse than exponential time in number of states
  - Empirically,  $O(n)$  iterations are required!
  - Polynomial bound on the number of PI iterations was open problem until very recently. **Closed!**

# Value Iteration vs. Policy Iteration

- Which is faster: VI or PI?
  - depends on the problem instance
- VI takes more iterations than PI, but PI requires more time on each iteration
  - PI must perform policy evaluation on each step which involves solving a linear system
- Complexity
  - There are at most  $\exp(n)$  policies, so PI is no worse than exponential time in number of states
  - Empirically,  $O(n)$  iterations are required!
  - Polynomial bound on the number of PI iterations was open problem until very recently. **Closed!**

# Introduction to AI

## Reinforcement Learning

רִינְפּוֹרְטְּמֵנְטָן  
אַלְגּוֹרִיתְמִים

Erez Karpas  
Slides by Carmel Domshlak

IE&M — Technion

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

# Three Approaches to Problem Solving

- ① **Programming:** specify control by hand
- ② **Model-oriented solving:**  
specify problem by hand, derive control automatically
- ③ **Learning:** learn control from experience

- All three have strengths and weaknesses;  
approaches not exclusive and often complementary.
- Planning is a form of model-oriented problem solving
- Today: from planning to learning

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

# Three Approaches to Problem Solving

- ① **Programming:** specify control by hand
- ② **Model-oriented solving:**  
specify problem by hand, derive control automatically
- ③ **Learning:** learn control from experience

- All three have strengths and weaknesses;  
approaches not exclusive and often complementary.
- Planning is a form of **model-oriented problem solving**
- Today: from planning to **learning**

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

# So far ...

- Given an MDP model we know how to find optimal policies
  - Value Iteration or Policy Iteration  
(or some approximate but faster algorithms)
- But what if we don't have any form of model?
  - Like when we were babies . . .
  - Like in many real-world applications
  - All we can do is wander around the world observing what happens, getting rewarded and punished
- Enters reinforcement learning

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

# So far ...

- Given an MDP model we know how to find optimal policies
  - Value Iteration or Policy Iteration  
(or some approximate but faster algorithms)
- But what if we don't have any form of model?
  - Like when we were babies . . .
  - Like in many real-world applications
  - All we can do is wander around the world observing what happens, getting rewarded and punished
- Enters reinforcement learning

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

# Reinforcement Learning

- No knowledge of environment
  - Can only **act** in the world and **observe** states and reward
  - We will assume the world behaves as an MDP  
*Restrictive?*
- Many factors make RL difficult:
  - Actions have non-deterministic effects
    - Which are initially unknown ...
  - Rewards / punishments are infrequent
    - Often at the end of long sequences of actions
    - How do we determine what action(s) were really responsible for reward or punishment?
  - World is large and complex
- Nevertheless learner **must decide** what actions to take

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

# Reinforcement Learning

- No knowledge of environment
  - Can only **act** in the world and **observe** states and reward
  - We will assume the world behaves as an MDP  
*(Full observability) מושג מלא*
- Many factors make RL difficult:
  - Actions have non-deterministic effects
    - Which are initially unknown ... ← מושגים נזקניים ידועים לא
  - Rewards / punishments are infrequent
    - Often at the end of long sequences of actions
    - How do we determine what action(s) were really responsible for reward or punishment?
  - World is large and complex
- Nevertheless learner **must decide** what actions to take

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

# Passive vs. Active learning

## Passive learning

- The agent has a **fixed policy** and tries to learn the utilities of states by observing the world go by
- Analogous to policy evaluation
- Often serves as a component of active learning algorithms
- Often inspires active learning algorithms

רְגֻלָּה מִזְמַרְתָּה אֲמֹדֵנָה יְגַדֵּל אֶלָּא ↵

## Active learning

- The agent tries to **find an optimal policy** (or at least good policy) by acting in the world
- Analogous to solving the underlying MDP, but without first being given the MDP model

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

# Model-Based vs. Model-Free RL

## Model-based approach to RL

- learn the MDP model, or an approximation of it
- use it for policy evaluation or to find the optimal policy

לימוד מודל  
לשימוש במודל

## Model-free approach to RL

- derive the optimal policy without explicitly learning the model
- useful when model is difficult to represent and/or learn

לימוד מודל  
במקרה של מודל קשה

We will consider both types of approaches

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

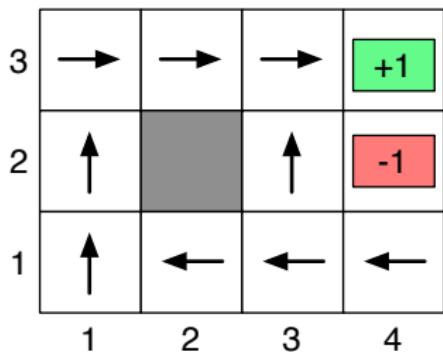
Passive RL

Active RL

Large state  
spaces

## Example: Passive RL

- Suppose given a fixed stationary policy  $\pi$ 
    - shown by arrows below
  - Want to determine how good it is
    - **Objective:** value function  $V_\pi$



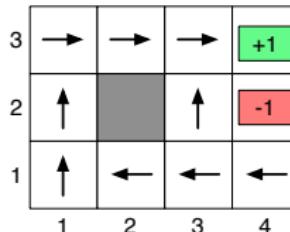
|   |       |       |       |       |
|---|-------|-------|-------|-------|
| 3 | 0.812 | 0.868 | 0.918 | +1    |
| 2 | 0.762 |       | 0.660 | -1    |
| 1 | 0.755 | 0.655 | 0.611 | 0.388 |

דוח תומך החק

# Passive RL

! Model Free ←

- Estimate  $V_\pi$ 
  - not given transition matrix
  - not given reward function



- Follow the policy for many epochs giving training sequences

(1, 1) → (1, 2) → (1, 3) → (1, 2) → (1, 3) → (2, 3) → (3, 3) → (3, 4) +1  
(1, 1) → (1, 2) → (1, 3) → (2, 3) → (3, 3) → (3, 2) → (3, 3) → (3, 4) +1  
(1, 1) → (2, 1) → (3, 1) → (3, 2) → (4, 2) -1

$$v(3,3) = r^3 \cdot 1 \quad v(3,3) = r \cdot 1$$

$$v(3,3) = r \cdot 1$$

- Assume that after entering +1 or -1 state the agent enters zero-reward terminal state

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# Also known as Monte-Carlo estimation

Estimate  $V_\pi(s)$  as average total reward of epochs containing  $s$   
(calculating from  $s$  to end of epoch)

- Reward-to-go of a state  $s =$   
the sum of the (discounted) rewards from that state until  
a terminal state is reached
- Key: use observed reward-to-go of the state as the direct  
evidence of the actual expected utility of that state
- Averaging the reward-to-go samples will converge to true  
value at state

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# Direct Estimation

- Converge very slowly to correct utilities values (requires a lot of sequences)
- Doesn't exploit Bellman constraints on policy values

$$V_{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) \cdot V_{\pi}(s')$$

.3,4 ře proš kež 3,3 ře proš řeñđ

- It is happy to consider estimates that violate this property badly
- How can we incorporate constraints?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# Approach 2: Adaptive Dynamic Programming (ADP)

- **Adaptive Dynamic Programming** (model based)

- Follow the policy for awhile
- Estimate transition model based on observations
- Learn reward function
- Use estimated model to compute utility of policy

$$V_{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) \cdot V_{\pi}(s')$$

- How can we estimate transition model  $P(s' | s, \pi(s))$ ?
  - Simply the fraction of times we see  $s'$  after taking  $a$  in state  $s$ .
  - (\*) Can bound error with Chernoff bound

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

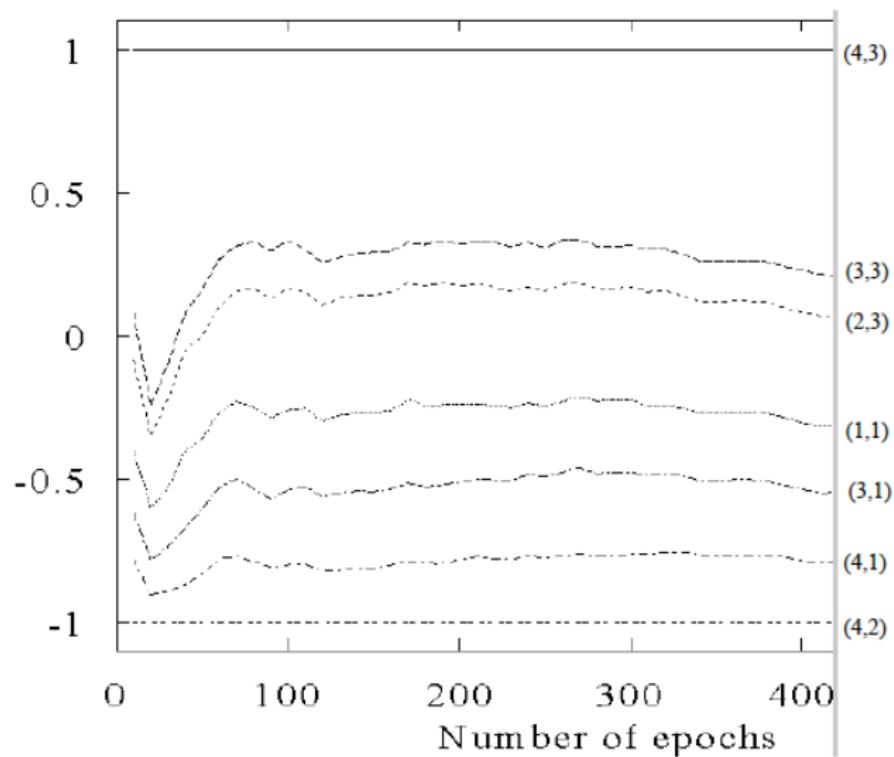
Introduction

Passive RL  
Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# ADP learning curves



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# Approach 3: Temporal Difference Learning (TD)

- Can we avoid the computational expense of full DP policy evaluation?
- Temporal Difference Learning (model free)
  - Do local updates of utility/value function on a per-action basis
  - Don't try to estimate entire transition function!
  - For each transition from  $s$  to  $s'$ , we perform the following update:

$$V_\pi(s) := V_\pi(s) + \alpha (R(s) + \gamma V_\pi(s') - V_\pi(s)),$$

with  $\alpha$  playing role of *learning rate*

- Intuitively moves us closer to satisfying Bellman constraint

$$V_\pi(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) \cdot V_\pi(s')$$

Why?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# Approach 3: Temporal Difference Learning (TD)

- Can we avoid the computational expense of full DP policy evaluation?
- Temporal Difference Learning (model free)
  - Do local updates of utility/value function on a per-action basis
  - Don't try to estimate entire transition function!
  - For each transition from  $s$  to  $s'$ , we perform the following update:

$$V_\pi(s) := V_\pi(s) + \alpha (R(s) + \gamma V_\pi(s') - V_\pi(s)),$$

with  $\alpha$  playing role of *learning rate*

- Intuitively moves us closer to satisfying Bellman constraint

$$V_\pi(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) \cdot V_\pi(s')$$

Why?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# Aside: Online Mean Estimation

Suppose that we want to **incrementally** compute the **mean of a sequence of numbers**

- Example: estimate the expected value of a random variable from a sequence of samples.
- Given a new sample  $x_{n+1}$ , the new mean is the old estimate (from  $n$  samples) plus the weighted difference between the new sample and old estimate

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL  
Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# Aside: Online Mean Estimation

Suppose that we want to **incrementally** compute the **mean of a sequence of numbers**

- Example: estimate the expected value of a random variable from a sequence of samples.
- Given a new sample  $x_{n+1}$ , the new mean is the old estimate (from  $n$  samples) plus the weighted difference between the new sample and old estimate

$$\begin{aligned}\hat{X}_{n+1} &= \frac{1}{n+1} \sum_{i=1}^{n+1} x_i \\ &= \frac{1}{n} \sum_{i=1}^n x_i + \frac{1}{n+1} \left( x_{n+1} - \frac{1}{n} \sum_{i=1}^n x_i \right) \\ &= \hat{X}_n + \underbrace{\frac{1}{n+1}}_{\text{New}} \underbrace{\left( x_{n+1} - \hat{X}_n \right)}_{\text{Difference}}\end{aligned}$$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL  
Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# Temporal Difference Learning (TD)

- TD update for transition from  $s$  to  $s'$ :

$$V_\pi(s) := V_\pi(s) + \alpha (R(s) + \gamma V_\pi(s') - V_\pi(s)),$$

- $\alpha$  is learning rate
- $R(s) + \gamma V_\pi(s')$  is (noisy) sample of utility based on next state
- So the update is ... maintaining a “mean” of (noisy) utility samples
- If the learning rate decreases appropriately with the number of samples (e.g.  $1/n$ ), then the utility estimates will converge to true values! (non-trivial)

$$V_\pi(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) \cdot V_\pi(s')$$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# Temporal Difference Learning (TD)

- TD update for transition from  $s$  to  $s'$ :

$$V_\pi(s) := V_\pi(s) + \alpha (R(s) + \gamma V_\pi(s') - V_\pi(s)),$$

- $\alpha$  is learning rate
- $R(s) + \gamma V_\pi(s')$  is (noisy) sample of utility based on next state
- So the update is ...  
maintaining a “mean” of (noisy) utility samples
- If the learning rate decreases appropriately with the number of samples (e.g.  $1/n$ ), then the utility estimates will converge to true values! (non-trivial)

$$V_\pi(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) \cdot V_\pi(s')$$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

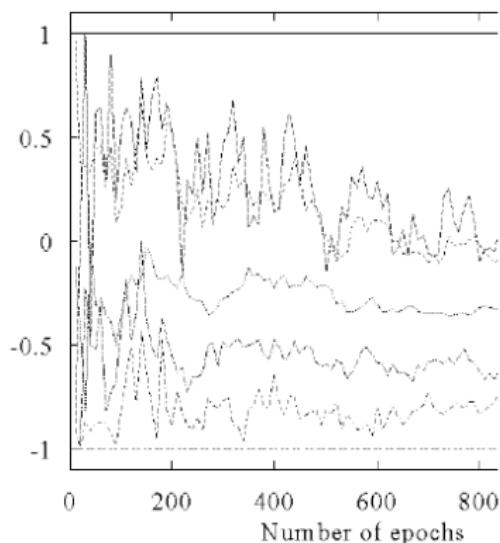
Passive RL

Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# The TD learning curve



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

- **Tradeoff:** requires more training experience (epochs) than ADP but much less computation per epoch
- **Choice:** relative cost of experience vs. computation

# Passive RL: Comparisons

- Monte-Carlo Direct Estimation (model free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints
  - Converges slowly
- Adaptive Dynamic Programming (model based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of updates)
- Temporal Difference Learning (model free)
  - Update speed and implementation similar to direct estimation
  - Partially exploits Bellman constraints—adjusts state to “agree” with observed successor
    - Not *all* possible successors
  - Convergence in between direct estimation and ADP

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL  
Direct  
estimation  
ADP  
TD

Active RL

Large state  
spaces

# Active Reinforcement Learning

תפקידו של אgent הוא לearn policy.

- So far, we've assumed agent **has** a policy
  - We just learned how good it is
- Now, suppose agent must learn a good policy (ideally optimal)
  - While acting in uncertain world

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Naive Approach

- ① Act randomly for a (long) time
- ② Learn transition function and reward function
- ③ Use value iteration, policy iteration,
- ④ Follow resulting policy thereafter.

Will this work? Yes, if we do step 1 long enough and there are no “dead-ends”

Any problems? We will act randomly for a long time before exploiting what we know

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL

Active RL

Naive  
approaches

Exploration vs.  
Exploitation

ADP-based RL

Optimistic RL

TD-based RL

Q-Learning

Large state  
spaces

## Naive Approach

- ① Act randomly for a (long) time
  - ② Learn transition function and reward function
  - ③ Use value iteration, policy iteration,
  - ④ Follow resulting policy thereafter.

**Will this work?** Yes, if we do step 1 long enough and there are no “dead-ends” 

**Any problems?** We will act randomly for a long time before exploiting what we know

**⊗ הינה היכי דעוי ג - kg ו-kg מוט נטן נטן גאנט ...  
טוטניגה עלי RL יי' מיניה. זו הנחה מאוזנת במיוחד - kg מואז  
נטן ק"מ' מטר נטן גאנט**

## Introduction to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL

Active RI

## Naive approaches

## Q-Learning

Large state  
spaces

# Revision of Naive Approach

השיטה הנטיבתית  
↓

- ① Start with initial (uninformed) model
- ② Solve for optimal policy given current model  
(using value or policy iteration)
- ③ Execute action suggested by policy in current state
- ④ Update estimated model based on observed transition
- ⑤ Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work? No. Can get stuck in local optimum

What can be done?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL

Active RL

Naive  
approaches

Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Revision of Naive Approach

- ① Start with initial (uninformed) model
- ② Solve for optimal policy given current model  
(using value or policy iteration)
- ③ Execute action suggested by policy in current state
- ④ Update estimated model based on observed transition
- ⑤ Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work? No. Can get stuck in local optimum

What can be done?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL

Active RL

Naive  
approaches

Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Revision of Naive Approach

- ① Start with initial (uninformed) model
- ② Solve for optimal policy given current model  
(using value or policy iteration)
- ③ Execute action suggested by policy in current state
- ④ Update estimated model based on observed transition
- ⑤ Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work? No. Can get stuck in local optimum

What can be done?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL

Active RL

Naive  
approaches

Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Exploration versus Exploitation

- Two reasons to take an action in RL

**Exploitation:** To try to get reward. We exploit our current knowledge to get a payoff.

**Exploration:** Get more information about the world. How do we know if there is not a pot of gold around the corner?

- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic **intuition** behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Exploration versus Exploitation

- Two reasons to take an action in RL

**Exploitation:** To try to get reward. We exploit our current knowledge to get a payoff.

**Exploration:** Get more information about the world. How do we know if there is not a pot of gold around the corner?

- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic **intuition** behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# ADP-based RL

- ① Start with initial model
- ② Solve for optimal policy given current model  
(using value or policy iteration)
- ③ Execute action suggested by an **explore/exploit policy**  
(explores more early on and gradually uses policy from 2)
- ④ Update estimated model based on observed transition
- ⑤ Goto 2

This is just ADP but we follow the explore/exploit policy

Will this work? Depends on the explore/exploit policy

Any ideas?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation

ADP-based RL

Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# ADP-based RL

- ① Start with initial model
- ② Solve for optimal policy given current model  
(using value or policy iteration)
- ③ Execute action suggested by an **explore/exploit policy**  
(explores more early on and gradually uses policy from 2)
- ④ Update estimated model based on observed transition
- ⑤ Goto 2

This is just ADP but we follow the explore/exploit policy

Will this work? Depends on the explore/exploit policy

Any ideas?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation

ADP-based RL

Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s' | s, a) \cdot V(s')$$

- where  $V$  is current optimal value function estimate (based on current model), and  $R$  and  $P$  are current estimates of model
- $Q(a, s)$  is the expected value of taking action  $a$  in state  $s$  and then getting the estimated value  $V(s')$  of the next state  $s'$
- Want an exploration policy that is greedy in the limit of infinite exploration (GLIE)
- GLIE Policy 1
  - On time step  $t$  select random action with probability  $p(t)$  and greedy action with probability  $1 - p(t)$
  - $p(t) = 1/t$  will lead to convergence, but is slow

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation

ADP-based RL

Optimistic RL

TD-based RL

Q-Learning

Large state  
spaces

# Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s' | s, a) \cdot V(s')$$

- where  $V$  is current optimal value function estimate (based on current model), and  $R$  and  $P$  are current estimates of model
- $Q(a, s)$  is the expected value of taking action  $a$  in state  $s$  and then getting the estimated value  $V(s')$  of the next state  $s'$
- Want an exploration policy that is greedy in the limit of infinite exploration (GLIE)
- GLIE Policy 1
  - On time step  $t$  select random action with probability  $p(t)$  and greedy action with probability  $1 - p(t)$
  - $p(t) = 1/t$  will lead to convergence, but is slow

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s' | s, a) \cdot V(s')$$

- where  $V$  is current value function estimate, and  $R$  and  $P$  are current estimates of model

- GLIE Policy 2: Boltzmann Exploration

- Select action  $a$  with probability

$$Pr(a | s) = \frac{\exp(Q(s, a)/T)}{\sum_{a' \in A} \exp(Q(s, a')/T)} \quad (\textcolor{red}{T > 0})$$

- $T$  is the “temperature”. Large  $T$  means that each action has about the same probability. Small  $T$  leads to more greedy behavior.
  - Typically: start with large  $T$  and decrease with time

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation

ADP-based RL

Optimistic RL

TD-based RL

Q-Learning

Large state  
spaces

# The Impact of Temperature

$$Pr(a | s) = \frac{\exp(Q(s, a)/T)}{\sum_{a' \in A} \exp(Q(s, a')/T)}$$

Suppose we have just two actions and that  $Q(s, a_1) = 1$ ,  
 $Q(s, a_2) = 2$ .

- $T = 10$  gives  $Pr(a_1 | s) = 0.48$ ,  $Pr(a_2 | s) = 0.52$ 
  - Almost equal probability, and so explore
- $T = 1$  gives  $Pr(a_1 | s) = 0.27$ ,  $Pr(a_2 | s) = 0.73$ 
  - Probabilities more skewed, so explore  $a_1$  less
- $T = 0.25$  gives  $Pr(a_1 | s) = 0.02$ ,  $Pr(a_2 | s) = 0.98$ 
  - Almost always exploit  $a_2$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation

ADP-based RL

Optimistic RL

TD-based RL

Q-Learning

Large state  
spaces

# Alternative Approach: Optimistic Exploration

- ① Start with initial model
- ② Solve for **optimistic policy** given current model  
(using optimistic value iteration)  
(inflates value of actions leading to unexplored regions)
- ③ Execute **greedy** action suggested by the **optimistic policy**  
(explores more early on and gradually uses policy from 2)
- ④ Update estimated model based on observed transition
- ⑤ Goto 2

Basically act as if all “unexplored” state-action pairs are maximally rewarding

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Alternative Approach: Optimistic Exploration

- ① Start with initial model
- ② Solve for **optimistic policy** given current model  
(using optimistic value iteration)  
(inflates value of actions leading to unexplored regions)
- ③ Execute **greedy** action suggested by the **optimistic policy**  
(explores more early on and gradually uses policy from 2)
- ④ Update estimated model based on observed transition
- ⑤ Goto 2

Basically act as if all “unexplored” state-action pairs are maximally rewarding

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Optimistic Exploration

- Recall that value iteration iteratively performs the following update at all states:

$$V(s) := R(s) + \gamma \cdot \max_a \sum_{s'} P(s' | s, a) \cdot V(s')$$

- Optimistic variant adjusts update to make actions that lead to unexplored regions look good

Value  
iteration

- Implement variant of VI that assigns the highest possible value  $V^{\max}$  to any state-action pair that has not been explored enough

- Maximum value is when we get maximum reward forever

$$V^{\max} = \sum_{t=0}^{\infty} \gamma^t \cdot R^{\max} = \frac{R^{\max}}{1 - \gamma}$$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Optimistic Exploration

- Recall that value iteration iteratively performs the following update at all states:

$$V(s) := R(s) + \gamma \cdot \max_a \sum_{s'} P(s' | s, a) \cdot V(s')$$

- Optimistic variant adjusts update to make actions that lead to unexplored regions look good
- Implement variant of VI that assigns the highest possible value  $V^{\max}$  to any state-action pair that has not been explored enough
- What do we mean by “explored enough”?
  - $N(s, a) > N_e$ , where  $N(s, a)$  is number of times action  $a$  has been tried in state  $s$  and  $N_e$  is a user selected parameter

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Optimistic Exploration

$$V(s) := R(s) + \gamma \cdot \max_a \sum_{s'} P(s' | s, a) \cdot V(s')$$

- Optimistic value iteration computes an optimistic value function  $V^+$  using updates

$$V^+(s) := R(s) + \gamma \cdot \max_a \begin{cases} V^{\max}, & N(s, a) < N_e \\ \sum_{s'} P(s' | s, a) \cdot V^+(s'), & N(s, a) \geq N_e \end{cases}$$

- The agent will behave initially as if there were wonderful rewards scattered all over around ( $\leadsto$  optimistic)
- But after actions are tried enough times, we will perform standard “non-optimistic” value updates

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# TD-based Active RL

תדריך מילוי מידע באמצעות דיסקונט

- ➊ Start with initial model
- ➋ Execute action from **explore/exploit policy**
- ➌ Update estimated model based on observed transition to state  $s'$
- ➍ Perform TD update

$$V(s) := V(s) + \alpha (R(s) + \gamma V(s') - V(s)) ,$$

- ➎ Goto 2

Just like TD for passive RL, but we follow explore/exploit policy

- Given the usual assumptions about learning rate and GLIE, TD will converge to an optimal value function!

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# TD-based Active RL

- ➊ Start with initial model
- ➋ Execute action from *explore/exploit policy*
- ➌ Update estimated model based on observed transition to state  $s'$
- ➍ Perform TD update

$$V(s) := V(s) + \alpha (R(s) + \gamma V(s') - V(s)) ,$$

- ➎ Goto 2

- Requires an estimated model. Why?  
*To compute  $Q(s,a)$  for greedy policy execution*
- Can we construct a model-free variant?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Q-Learning: Model-Free RL

- Instead of learning the optimal value function  $V$ , directly learn the optimal  $Q$  function.
  - Recall:  $Q(s, a)$  is the expected value of taking action  $a$  in state  $s$  and then following the optimal policy thereafter
- The optimal  $Q$ -function satisfies  $V(s) = \max_a Q(s, a)$ , which gives:

$$\begin{aligned} Q(s, a) &= R(s) + \gamma \sum_{s'} P(s' | s, a) V(s') \\ &= R(s) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a') \end{aligned}$$

- Given the  $Q$ -function, we can act optimally by selecting action greedily according to  $Q(s, a)$  without a model
  - How can we learn the  $Q$ -function directly?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Q-Learning: Model-Free RL

- Instead of learning the optimal value function  $V$ , directly learn the optimal  $Q$  function.
  - Recall:  $Q(s, a)$  is the expected value of taking action  $a$  in state  $s$  and then following the optimal policy thereafter
- The optimal  $Q$ -function satisfies  $V(s) = \max_a Q(s, a)$ , which gives:

$$\begin{aligned} Q(s, a) &= R(s) + \gamma \sum_{s'} P(s' | s, a) V(s') \\ &= R(s) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a') \end{aligned}$$

- Given the  $Q$ -function, we can act optimally by selecting action greedily according to  $Q(s, a)$  *without a model*
  - How can we learn the  $Q$ -function directly?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

## Q-Learning: Model-Free RL

## Bellman constraints on optimal $Q$ -function

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a')$$

- We can perform updates after each action just like in TD.
    - After taking **action  $a$**  in **state  $s$**  and reaching **state  $s'$**  do:  
(note that we directly observe reward  $R(s)$ )

$$Q(s, a) := Q(s, a) + \alpha \left( R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

- blue part: (noisy) sample of  $Q$ -value based on next state

## Introduction to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive approaches  
Exploration vs Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Q-Learning

- ➊ Start with initial  $Q$ -function (e.g., all zeros)
- ➋ Execute action from **explore/exploit policy** giving new state  $s'$
- ➌ Perform TD update

$$Q(s, a) := Q(s, a) + \alpha \left( R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

- ➍ Goto 2

- Does not require model since we learn  $Q$  directly!

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL

Active RL  
Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Active RL: Comparisons

- Methods
  - ① Adaptive Dynamic Programming (ADP)
  - ② Temporal Difference (TD)
  - ③ Q-learning
- All converge to optimal policy assuming a GLIE exploration strategy
- All methods assume the world is not too dangerous (no cliffs to fall off during exploration)
- Note: so far we have assumed small state spaces
  - *Which one is better? (Hmm ...)*

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Naive  
approaches  
Exploration vs.  
Exploitation  
ADP-based RL  
Optimistic RL  
TD-based RL  
Q-Learning

Large state  
spaces

# Large State Spaces

- When a problem has a large state space we can no longer represent the  $V$  or  $Q$  functions as explicit tables
- Even if we had enough memory
  - Never enough training data!
  - Learning takes too long
- What to do??

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

# Function Approximation

- Never enough training data!
  - Must **generalize** what is learned from one situation to other “similar” new situations
- Idea:
  - Instead of using large table to represent  $V$  or  $Q$ , use a **parameterized function**
  - The number of parameters should be small compared to number of states (generally exponentially fewer parameters)
  - Learn parameters from experience
  - When we update the parameters based on observations in one state, then our  $V$  or  $Q$  estimate will also change for other similar states
    - Parameterization facilitates generalization of experience!

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

# Function Approximation

- Never enough training data!
  - Must **generalize** what is learned from one situation to other “similar” new situations
- Idea:
  - Instead of using large table to represent  $V$  or  $Q$ , use a **parameterized function**
    - The number of parameters should be small compared to number of states (generally exponentially fewer parameters)
  - Learn parameters from experience
  - When we update the parameters based on observations in one state, then our  $V$  or  $Q$  estimate will also change for other similar states
    - Parameterization facilitates generalization of experience!

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

## Linear Function Approximation

# Linear Function Approximation

- Define a set of **state features**  $f_1(s), \dots, f_n(s)$ 
  - The features are used as our representation of states
  - States with similar feature values will be considered to be similar
- A common approximation is to represent  $V(s)$  as a weighted sum of the features (**linear approximation**)
$$V_\theta(s) = \theta_0 + \theta_1 f_1(s) + \dots + \theta_n f_n(s)$$
- The approximation accuracy is fundamentally limited by the information provided by the features
- Can we always define features that allow for a perfect linear approximation?
  - Yes. Assign each state an indicator feature.
  - Of course, this requires far too many features and gives no generalization.

וְאֵלּוּ בָּאַמָּנוּתִים נִזְנְצָה

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL  
Active RL

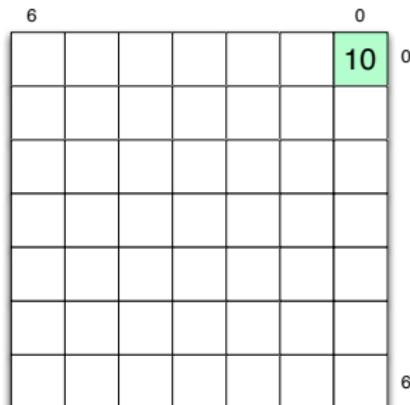
Large state  
spaces

Linear  
Approximation

# Example

Grid with no obstacles, deterministic actions U/D/L/R, no discounting, -1 reward everywhere except +10 at goal.

- Features for  $s = (x, y)$ :  
 $f_1(s) = x, f_2(s) = y$
- $V_\theta(s) = \theta_0 + \theta_1 x + \theta_2 y$
- Is there a good linear approximation?
  - Yes.
  - $\theta_0 = 10, \theta_1 = \theta_2 = -1$
  - note: upper right is origin
- $V_\theta(s) = 10 - x - y$   
(subtracts Manhattan distance from goal reward)



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

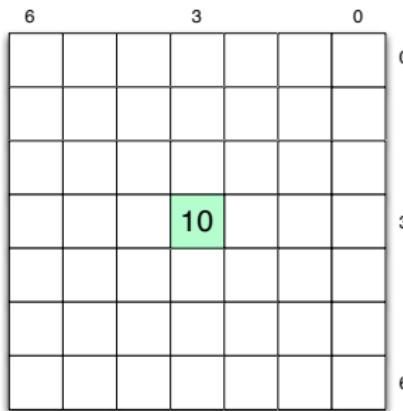
Large state  
spaces

Linear  
Approximation

# But What If We Change Reward ...

Grid with no obstacles, deterministic actions U/D/L/R, no discounting, -1 reward everywhere except +10 at goal.

- $V_\theta(s) = \theta_0 + \theta_1 x + \theta_2 y$
- Is there a good linear approximation?
  - No!



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

# But What If ...

Grid with no obstacles, deterministic actions U/D/L/R, no discounting, -1 reward everywhere except +10 at goal.

- Features for  $s = (x, y)$ :

$$f_1(s) = x, f_2(s) = y,$$

$$f_3(s) = |3 - x| + |3 - y|$$

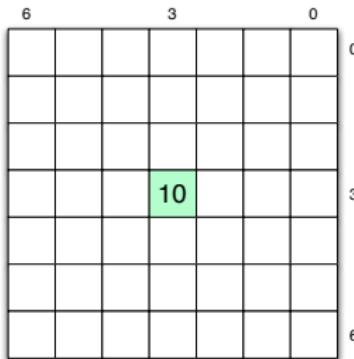
- $V_\theta(s) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 f_3(s)$

- Is there a good linear approximation?

- Yes.

- $\theta_0 = 10, \theta_1 = \theta_2 = 0,$   
 $\theta_3 = -1$

וועודו  
וועודו  
! וועודו



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

# Linear Function Approximation

- Define a set of state features  $f_1(s), \dots, f_n(s)$ 
  - The features are used as our representation of states
  - States with similar feature values will be considered to be similar
  - More complex functions require more complex features

$$V_\theta(s) = \theta_0 + \theta_1 f_1(s) + \dots + \theta_n f_n(s)$$

- Our goal is to **learn good parameter values** (i.e. feature weights) that approximate the value function well
  - How can we do this?
  - Use TD-based RL and somehow update parameters based on each experience.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

# TD-based RL for Linear Approximators

!תדריך למדעי המחשב פ.3

- ➊ Start with initial parameter values
- ➋ Execute action from explore/exploit policy
- ➌ Update estimated model (if model is not available)
- ➍ Perform TD update for each parameter:  $\theta_i := ?$
- ➋ Goto 2

What is a “TD update” for a parameter?

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL  
Active RL

Large state  
spaces  
Linear  
Approximation

# Aside: Gradient Descent

Given a function  $f(\theta_1, \dots, \theta_n)$  of  $n$  real values  $\theta = (\theta_1, \dots, \theta_n)$ , suppose we want to minimize  $f$  with respect to  $\theta$ .

## Gradient Descent

- The gradient of  $f$  at point  $\theta$ , denoted  $\nabla f(\theta)$  is an  $n$ -dimensional vector that points in the direction where  $f$  increases most steeply at point  $\theta$ .
- Calculus tells us that  $\nabla f(\theta)$  is just a vector of partial derivatives

$$\nabla f(\theta) = \left[ \frac{\partial f(\theta)}{\partial \theta_1}, \dots, \frac{\partial f(\theta)}{\partial \theta_n} \right]$$

$$\text{where } \frac{\partial f(\theta)}{\partial \theta_i} = \lim_{\varepsilon \rightarrow 0} \frac{f(\theta_1, \dots, \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \dots, \theta_n) - f(\theta)}{\varepsilon}$$

- Can decrease  $f$  by moving in negative gradient direction

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

# Aside: Gradient Descent

Given a function  $f(\theta_1, \dots, \theta_n)$  of  $n$  real values  $\theta = (\theta_1, \dots, \theta_n)$ , suppose we want to minimize  $f$  with respect to  $\theta$ .

## Gradient Descent

- The gradient of  $f$  at point  $\theta$ , denoted  $\nabla f(\theta)$  is an  $n$ -dimensional vector that points in the direction where  $f$  increases most steeply at point  $\theta$ .
- Calculus tells us that  $\nabla f(\theta)$  is just a vector of partial derivatives

$$\nabla f(\theta) = \left[ \frac{\partial f(\theta)}{\partial \theta_1}, \dots, \frac{\partial f(\theta)}{\partial \theta_n} \right]$$

$$\text{where } \frac{\partial f(\theta)}{\partial \theta_i} = \lim_{\varepsilon \rightarrow 0} \frac{f(\theta_1, \dots, \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \dots, \theta_n) - f(\theta)}{\varepsilon}$$

- Can decrease  $f$  by moving in negative gradient direction

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

## Aside: Gradient Descent for Squared Error

- Suppose that we have a sequence of states and target values for each state  $\langle s_1, v(s_1) \rangle, \langle s_2, v(s_2) \rangle, \dots$ 
  - for instance, produced by TD-based RL loop
- Our goal is to minimize the sum of squared errors between our estimated function and each target value:

$$E_j = \frac{1}{2} (V_\theta(s_j) - v(s_j))^2$$

- $E_j$  is squared error of example  $j$
  - $V_\theta(s_j)$  is our estimated value for  $s_j$
  - $v(s_j)$  is target value of  $s_j$
- 
- After seeing  $s_j$  the gradient descent rule tells us that we can decrease error by updating parameters by:

$$\theta_i := \theta_i - \alpha \frac{\partial E_j}{\partial \theta_i}$$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

## Aside: Gradient Descent for Squared Error

- Suppose that we have a sequence of states and target values for each state  $\langle s_1, v(s_1) \rangle, \langle s_2, v(s_2) \rangle, \dots$ 
  - for instance, produced by TD-based RL loop
- Our goal is to minimize the sum of squared errors between our estimated function and each target value:

$$E_j = \frac{1}{2} (V_\theta(s_j) - v(s_j))^2$$

- $E_j$  is squared error of example  $j$
  - $V_\theta(s_j)$  is our estimated value for  $s_j$
  - $v(s_j)$  is target value of  $s_j$
- After seeing  $s_j$  the **gradient descent rule** tells us that we can decrease error by updating parameters by:

$$\theta_i := \theta_i - \alpha \frac{\partial E_j}{\partial \theta_i}$$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

# Aside: continued ...

$$\begin{aligned}\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j}{\partial \theta_i} &= \theta_i - \alpha \frac{\partial E_j}{\partial V_\theta(s_j)} \frac{\partial V_\theta(s_j)}{\partial \theta_i} \\ &= \theta_i - \alpha (V_\theta(s_j) - v(s_j)) \frac{\partial V_\theta(s_j)}{\partial \theta_i} \\ &\stackrel{\text{linear}}{=} \theta_i - \alpha (V_\theta(s_j) - v(s_j)) f_i(s_j)\end{aligned}$$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL  
Active RL

Large state  
spaces  
Linear  
Approximation

## Aside: continued ...

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL

Active RL

Large state  
spaces

Linear  
Approximation

$$\begin{aligned}\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j}{\partial \theta_i} &= \theta_i - \alpha \frac{\partial E_j}{\partial V_\theta(s_j)} \frac{\partial V_\theta(s_j)}{\partial \theta_i} \\ &= \theta_i - \alpha(V_\theta(s_j) - v(s_j)) \frac{\partial V_\theta(s_j)}{\partial \theta_i} \\ &\stackrel{\text{linear}}{=} \theta_i - \alpha(V_\theta(s_j) - v(s_j)) f_i(s_j)\end{aligned}$$

- Thus the update becomes:  
 $\theta_i := \theta_i + \alpha (v(s_j) - V_\theta(s_j)) f_i(s_j)$
- (For linear functions) this update is guaranteed to converge to best approximation for suitable learning rate schedule

# TD-based RL for Linear Approximators

- ① Start with initial parameter values
- ② Execute action from *explore/exploit policy*
- ③ Update estimated model (if model is not available)
- ④ Perform TD update for each parameter:  
$$\theta_i := \theta_i + \alpha (v(s_j) - V_\theta(s_j)) f_i(s_j)$$
- ⑤ Goto 2

What should we use for “target value”  $v(s)$ ?

Use the TD prediction based on the next state  $s'$

$$v(s) = R(s) + \gamma V_\theta(s')$$

which is the same as previous TD method, only with approximation.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL  
Active RL

Large state  
spaces  
Linear  
Approximation

# TD-based RL for Linear Approximators

- ① Start with initial parameter values
- ② Execute action from *explore/exploit policy*
- ③ Update estimated model (if model is not available)
- ④ Perform TD update for each parameter:  
 $\theta_i := \theta_i + \alpha (v(s_j) - V_\theta(s_j)) f_i(s_j)$
- ⑤ Goto 2

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL  
Active RL

Large state  
spaces  
Linear  
Approximation

What should we use for “target value”  $v(s)$ ?

Use the TD prediction based on the next state  $s'$

$$v(s) = R(s) + \gamma V_\theta(s') \quad \text{השאלה היא בקשר ל} \quad \text{השאלה היא בקשר ל}$$

which is the same as previous TD method, only with approximation.

# TD-based RL for Linear Approximators

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL  
Active RL

Large state  
spaces  
Linear  
Approximation

- ① Start with initial parameter values
- ② Execute action from **explore/exploit policy**  
(should converge to greedy policy, i.e. GLIE)
- ③ Update estimated model (if model is not available)
- ④ Perform TD update for each parameter:

$$\theta_i := \theta_i + \alpha (R(s) + \gamma V_\theta(s') - V_\theta(s_j)) f_i(s_j)$$

- ⑤ Goto 2

- Step 2 requires a model to select greedy action
  - For applications such as Backgammon it is easy to get a simulation-based model
  - For others it is difficult to get a good model
- But we can do the same thing for model-free  $Q$ -learning

# TD-based RL for Linear Approximators

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL  
Active RL

Large state  
spaces  
Linear  
Approximation

- ① Start with initial parameter values
- ② Execute action from **explore/exploit policy**  
(should converge to greedy policy, i.e. GLIE)
- ③ Update estimated model (if model is not available)
- ④ Perform TD update for each parameter:

$$\theta_i := \theta_i + \alpha (R(s) + \gamma V_\theta(s') - V_\theta(s_j)) f_i(s_j)$$

- ⑤ Goto 2

- Step 2 requires a model to select greedy action
  - For applications such as Backgammon it is easy to get a simulation-based model
  - For others it is difficult to get a good model
- But we can do the same thing for model-free  $Q$ -learning

# Q-Learning for Linear Approximators

Features are function of states and actions:

$$Q_\theta(s, a) = \theta_0 + \theta_1 f_1(s, a) + \cdots + \theta_n f_n(s, a)$$

- ① Start with initial parameter values
- ② Execute action from **explore/exploit policy** giving  $s'$   
(should converge to greedy policy, i.e. GLIE)
- ③ Perform TD update for each parameter:

$$\theta_i := \theta_i + \alpha \left( R(s) + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a) \right) f_i(s, a)$$

- ④ Goto 2

- TD converges close to minimum error solution
- Q-learning can diverge!  
(Converges under some conditions.)

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL  
Active RL

Large state  
spaces

Linear  
Approximation

# Q-Learning for Linear Approximators

Features are function of states and actions:

$$Q_\theta(s, a) = \theta_0 + \theta_1 f_1(s, a) + \cdots + \theta_n f_n(s, a)$$

- ➊ Start with initial parameter values
- ➋ Execute action from **explore/exploit policy** giving  $s'$   
(should converge to greedy policy, i.e. GLIE)
- ➌ Perform TD update for each parameter:

$$\theta_i := \theta_i + \alpha \left( R(s) + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a) \right) f_i(s, a)$$

- ➍ Goto 2

- TD converges close to minimum error solution
- Q-learning can diverge!  
(Converges under some conditions.)

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Passive RL  
Active RL

Large state  
spaces

Linear  
Approximation

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Introduction to AI

Game Playing = Planning with Adversary

Erez Karpas  
Slides by Carmel Domshlak

IE&M — Technion

# Problem solving with and in face of multiple agents

Sometimes we are not alone ... *X Files...*

- So far we assumed that our agent is the only one affecting the environment  
... well, we assumed even more than that, right?
- This allows us to generate a plan and predict its outcome
- If other agents are around, they may also want/happen to affect the environment

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# From planning to playing

## Multi-agent settings

- Fully cooperative (not that different from single agent)
- Self-interested

- Cooperative but agents are self-interested ← "הקופט נאבקים"

- Non-cooperative ← "רשותה נבנאי".  
↳ גורמים נבדוקים  
↳ דוגמאות, ↳ קולטים

- Adversarial ↴  
↳ סיריות

הקופט נאבקים,  
↳ נאבקים.

רשותה נבנאי.  
↳ נבנאי סוגיות.  
↳ סכום מילוי.

"הקופט  
נאבקים"  
רשותה

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Relation to games and game theory

- Game theory addresses decision making in multi-agent setting: “Assuming that the other agents are rational, what do I have to do to achieve my goals?”
  - Game theory is related to **multi-agent planning**
- “Unpredictable” opponent  $\leadsto$  solution is a **strategy** specifying a move for every possible opponent reply
  - Example: Finding a **winning strategy** of a game like chess. In this case it is not necessary to distinguish between **intelligent opponent** and **randomly behaving opponent**.
- Game theory in general is about **optimal strategies** which do not necessarily guarantee winning. For example card games like poker do not have a winning strategy.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Relation to games and game theory

- Game theory addresses decision making in multi-agent setting: "Assuming that the other agents are rational, what do I have to do to achieve my goals?"
  - Game theory is related to **multi-agent planning**
- "Unpredictable" opponent  $\rightsquigarrow$  solution is a **strategy** specifying a move for every possible opponent reply
  - Example: Finding a **winning strategy** of a game like chess. In this case it is not necessary to distinguish between **intelligent opponent** and **randomly behaving opponent**.
- Game theory in general is about **optimal strategies** which do not necessarily guarantee winning. For example card games like poker do not have a winning strategy.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Relation to games and game theory

- Game theory addresses decision making in multi-agent setting: "Assuming that the other agents are rational, what do I have to do to achieve my goals?"
  - Game theory is related to **multi-agent planning**
- "Unpredictable" opponent  $\leadsto$  solution is a **strategy** specifying a move for every possible opponent reply
  - Example: Finding a **winning strategy** of a game like chess.  
In this case it is not necessary to distinguish between **intelligent opponent** and **randomly behaving opponent**.
- Game theory in general is about **optimal strategies** which do not necessarily guarantee winning. For example card games like poker do not have a winning strategy.

הנחות מינימלית ומקסימלית מושג הערך המוחלט של המשחק. מושג זה מוגדר כערך המוחלט של תוצאות השחקן המנצח במשחק. מושג זה מוגדר כערך המוחלט של תוצאות השחקן המנצח במשחק.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Types of games

|                       | deterministic                   | chance                                 |
|-----------------------|---------------------------------|----------------------------------------|
| perfect information   | chess, checkers,<br>go, othello | backgammon<br>monopoly                 |
| imperfect information | battleships,<br>blind tictactoe | bridge, poker, scrabble<br>nuclear war |

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

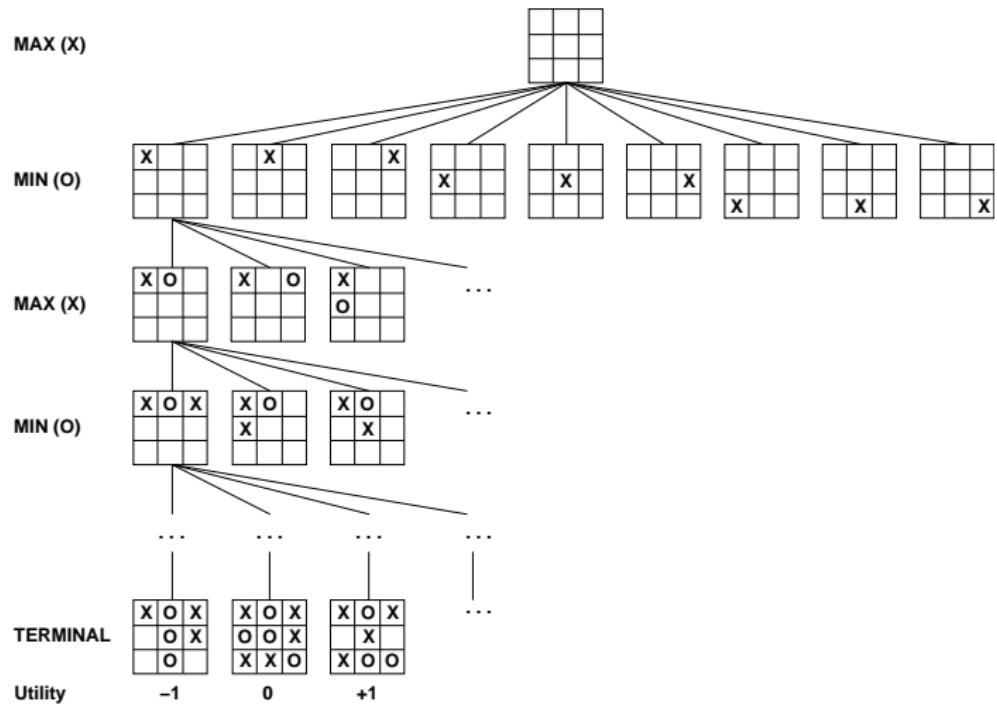
Introduction  
Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Game tree (2-player, deterministic, turns)



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Perfect play

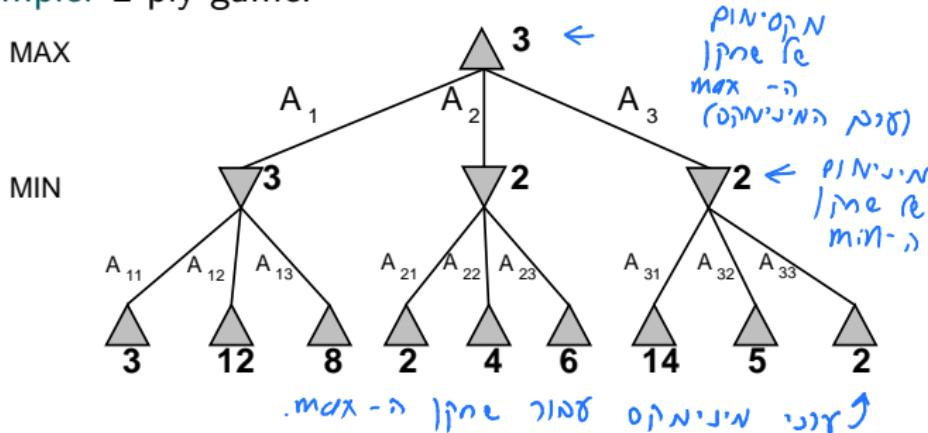
Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Minimax

- Perfect play for deterministic, perfect-information games
- Idea: choose move to position with highest **minimax value**  
= best achievable payoff against best play
- Example: 2-ply game:



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Minimax algorithm

## minimax-decision

```
value := -∞  
for each  $\langle o, s \rangle \in \text{succ}(\text{init}())$ :  
    value := max {value, min-value( $s$ )}  
return value
```

## max-value( $s$ )

```
if terminal-test( $s$ ): return utility( $s$ )  
value := -∞  
for each  $\langle o, s' \rangle \in \text{succ}(s)$ :  
    value := max {value, min-value( $s'$ )}  
return value
```

...

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Minimax algorithm

**max-value( $s$ )**

```
if terminal-test( $s$ ): return utility( $s$ )
value := -∞
for each  $\langle o, s' \rangle \in \text{succ}(s)$ :
    value := max {value, min-value( $s'$ )}
return value
```

**min-value( $s$ )**

```
if terminal-test( $s$ ): return utility( $s$ )
value := ∞
for each  $\langle o, s' \rangle \in \text{succ}(s)$ :
    value := min {value, max-value( $s'$ )}
return value
```

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Properties of Minimax

For finite  $m$ , fin resp : b

**Complete** Only if tree is finite (chess has specific rules for this).

**Optimal** Yes, against an optimal opponent. (*Otherwise?*)

**Time**  $O(b^m)$

**Space**  $O(bm)$  (depth-first exploration)

For chess,  $b \approx 35$ ;  $m \approx 100$  for “reasonable” games

~ exact solution completely infeasible

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Properties of Minimax

**Complete** Only if tree is finite (chess has specific rules for this).

**Optimal** Yes, against an optimal opponent. (*Otherwise?*)

**Time**  $O(b^m)$

**Space**  $O(bm)$  (depth-first exploration)

profon bn  
אלה גור תבונן      profon prz  
(אלה גור תבונן) תבונן

For chess,  $b \approx 35$ ;  $m \approx 100$  for “reasonable” games

~ exact solution completely infeasible

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

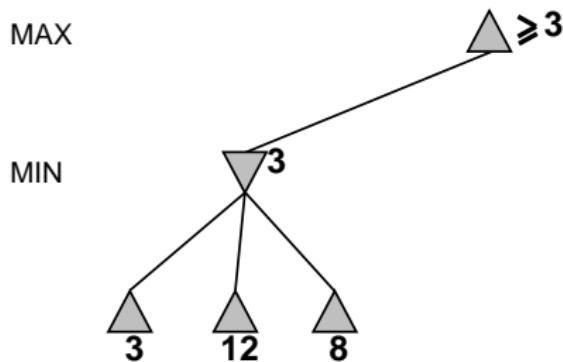
Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# $\alpha$ - $\beta$ pruning example



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

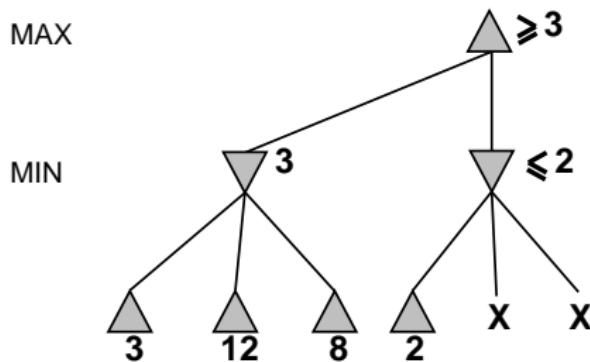
Introduction  
Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# $\alpha$ - $\beta$ pruning example



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

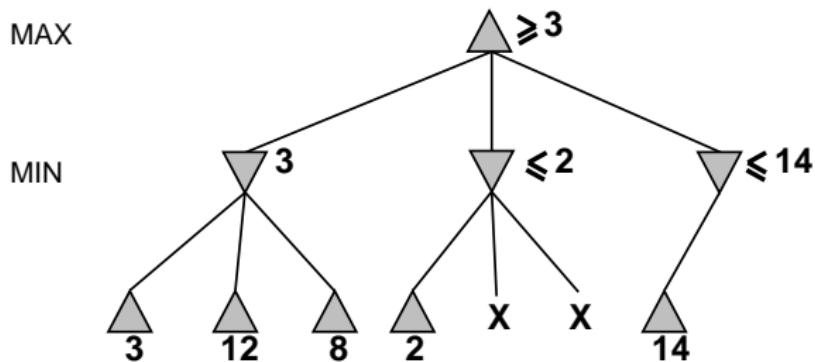
Introduction  
Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# $\alpha$ - $\beta$ pruning example



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

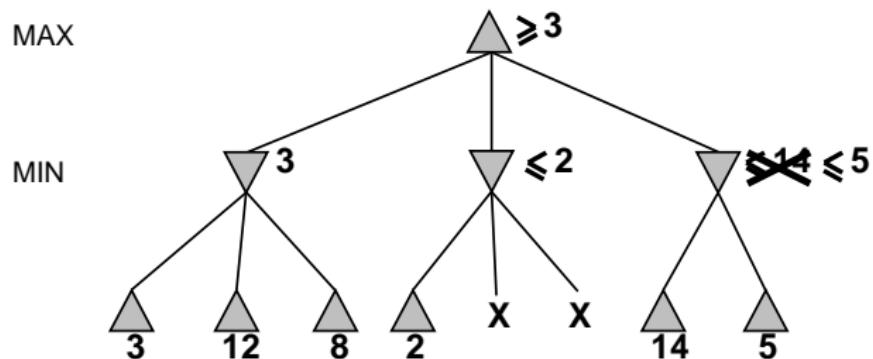
Introduction  
Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# $\alpha$ - $\beta$ pruning example



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

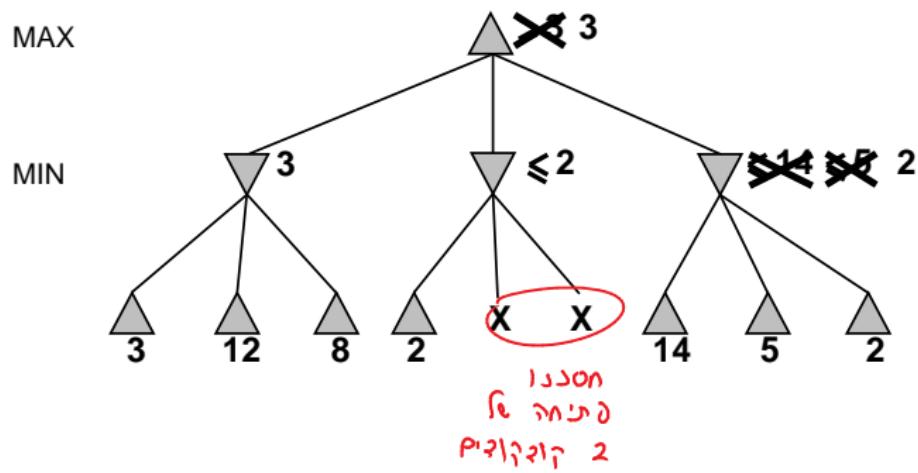
Introduction  
Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# $\alpha$ - $\beta$ pruning example



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

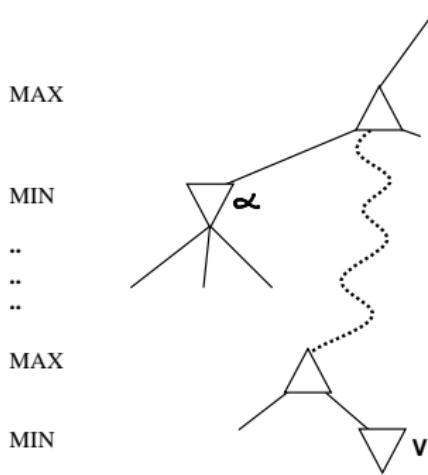
Introduction  
Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Why is it called $\alpha$ - $\beta$ ?



- $\alpha$  is the best value (to MAX) found so far off the current path
- if  $v$  is worse than  $\alpha$ , MAX will avoid it  $\leadsto$  prune that branch!
- define  $\beta$  similarly for MIN

# The $\alpha$ - $\beta$ algorithm

$\alpha$ -based pruning

$\text{min-value}(s, \alpha, \beta)$

```
if terminal-test( $s$ ): return utility( $s$ )
value :=  $\infty$ 
for each  $\langle o, s' \rangle \in \text{succ}(s)$ :
    value := min {value, max-value( $s'$ ,  $\alpha, \beta$ )}
    if value  $\leq \alpha$  then return value
     $\beta$  := min{ $\beta$ , value}
return value
```

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# The $\alpha$ - $\beta$ algorithm

$\beta$ -based pruning

**max-value**( $s, \alpha, \beta$ )

**if** terminal-test( $s$ ): **return** utility( $s$ )

$value := -\infty$

**for each**  $\langle o, s' \rangle \in \text{succ}(s)$ :

$value := \max \{ value, \text{min-value}(s', \alpha, \beta) \}$

**if**  $value \geq \beta$  **then return**  $value$

$\alpha := \max\{\alpha, value\}$

**return**  $value$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Properties of $\alpha$ - $\beta$

- Pruning **does not** affect final result
- Good move ordering improves effectiveness of pruning
- With “perfect ordering,” time complexity =  $O(b^{\frac{m}{2}})$   
~ **doubles solvable depth**
- A simple example of the value of reasoning about which computations are relevant (a form of **metareasoning**)
- Unfortunately,  $35^{50}$  is still impossible!

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Resource limits

## Standard approach

- Use cutoff-test instead of terminal-test
  - e.g., depth limit
- Use evaluation instead of utility
  - i.e., heuristic function that estimates desirability of state

Suppose we have 100 seconds, explore  $10^4$  nodes/second

$$\leadsto 10^6 \text{ nodes per move} \approx 35^{8/2}$$

$\leadsto \alpha\text{-}\beta$  reaches depth 8  $\Rightarrow$  pretty good chess program

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

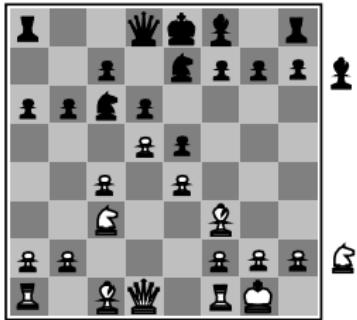
Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

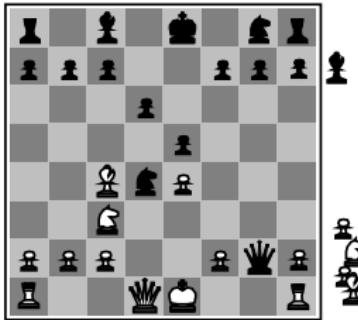
Games of  
imperfect  
information

# Evaluation functions



Black to move

White slightly better



White to move

Black winning

For chess, typically **linear** weighted sum of **features**

$$h(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

. material plan - pawns

e.g.,  $w_1 = 9$  with

$f_1(s) = (\# \text{ white queens}) - (\# \text{ black queens}), \text{ etc.}$

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

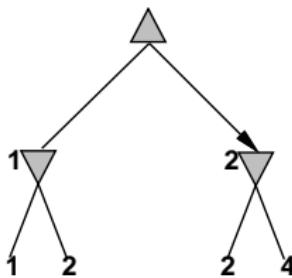
Games of  
chance

Games of  
imperfect  
information

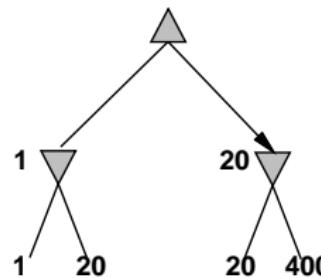
# Properties of evaluation function

Exact values don't matter

MAX



MIN



- Behavior is preserved under any **monotonic** transformation of  $h$
- Only the order matters: payoff in deterministic games acts as an **ordinal utility** function

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Deterministic games in practice

וְיַדְעָה  
↓

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
- Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997.
  - Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
  - Fritz (on PC!) against Kramnik in 2002 – 4-4 draw.
- Othello: Human champions refuse to compete against computers, who are too good.
- Go:
  - (Until 2014) Human champions refuse to compete against computers, who are too bad.
  - (2016) Computer program beat best human player
  - One of the key issues:  $b > 300$ .

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Deterministic games in practice

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
- **Chess:** Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997.
  - Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
  - Fritz (on PC!) against Kramnik in 2002 – 4-4 draw.
- **Othello:** Human champions refuse to compete against computers, who are too good.
- **Go:**
  - (Until 2014) Human champions refuse to compete against computers, who are too bad.
  - (2016) Computer program beat best human player
  - One of the key issues:  $b > 300$ .

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Deterministic games in practice

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
- **Chess:** Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997.
  - Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
  - Fritz (on PC!) against Kramnik in 2002 – 4-4 draw.
- **Othello:** Human champions refuse to compete against computers, who are too good.
- **Go:**
  - (Until 2014) Human champions refuse to compete against computers, who are too bad.
  - (2016) Computer program beat best human player
  - One of the key issues:  $b > 300$ .

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Deterministic games in practice

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
- **Chess:** Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997.
  - Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
  - Fritz (on PC!) against Kramnik in 2002 – 4-4 draw.
- **Othello:** Human champions refuse to compete against computers, who are too good.
- **Go:**
  - (Until 2014) Human champions refuse to compete against computers, who are too bad.
  - (2016) Computer program beat best human player
  - One of the key issues:  $b > 300$ .

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

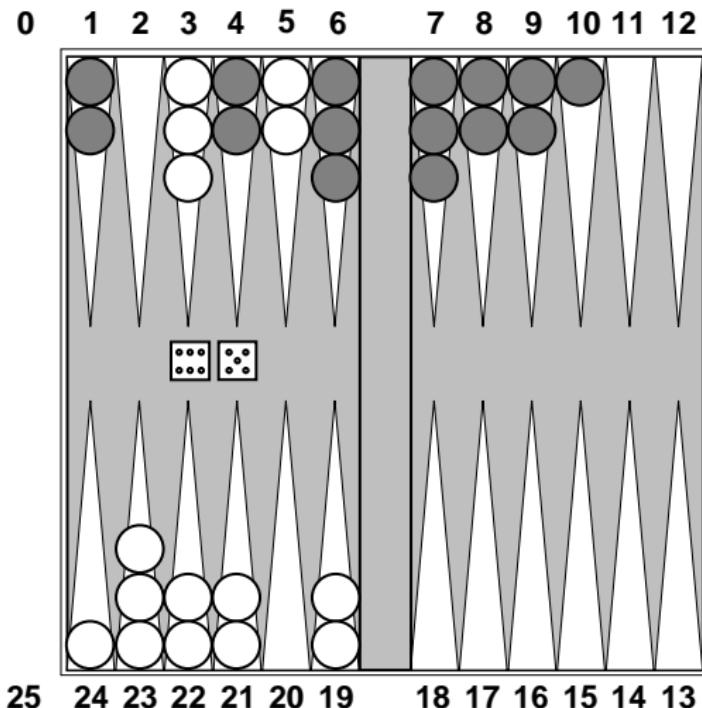
Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Nondeterministic games

## Backgammon



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

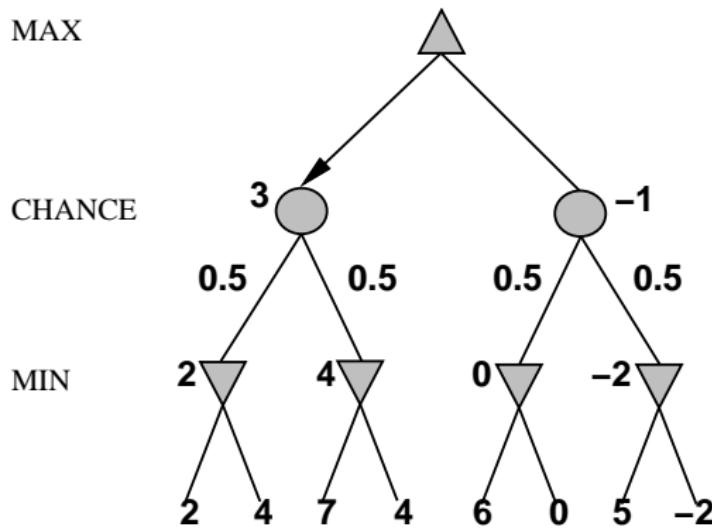
Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Nondeterministic games in general

- In nondeterministic games, chance introduced by dice, card-shuffling
- Simplified example with coin-flipping:



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Algorithm for nondeterministic games

- expecti-minimax gives perfect play
- Just like minimax, except we must also handle chance nodes:

```
...  
if  $\sigma$  is a MAX node then  
    return the highest expecti-minimax-value of succ(state( $\sigma$ ))  
if  $\sigma$  is a MIN node then  
    return the lowest expecti-minimax-value of succ(state( $\sigma$ ))  
if  $\sigma$  is a CHANCE node then  
    return average of expecti-minimax-value of succ(state( $\sigma$ ))  
...  
...
```

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Nondeterministic games in practice

- Dice rolls increase  $b$ : 21 possible rolls with 2 dice
- Backgammon  $\approx 20$  legal moves  
(can be 6,000 with 1-1 roll)  
$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$
- As depth increases, probability of reaching a given node shrinks
  - ~ value of lookahead is diminished
- TDGAMMON uses depth-2 search + very good  $h$   
 $\approx$  world-champion level

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

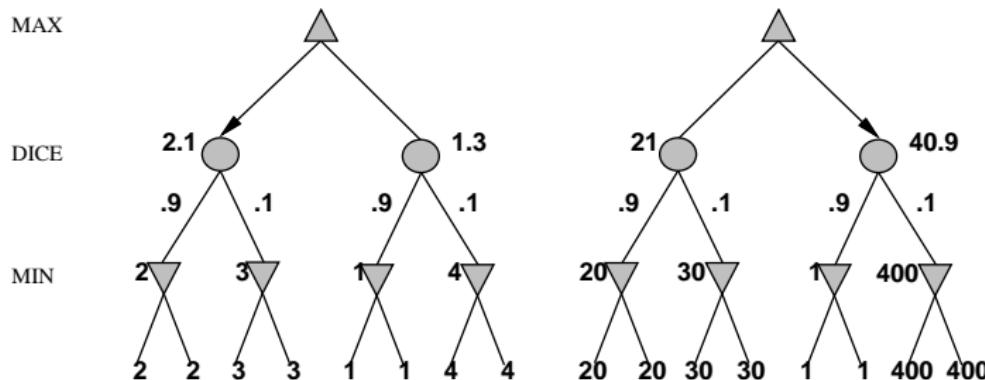
Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Properties of evaluation function

Exact values DO matter



- Behaviour is preserved only by **positive linear** transformation of  $h$
- Hence  $h$  should be proportional to the expected payoff

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction  
Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Games of imperfect information

- Example: card games, where opponent's initial cards are unknown
- Typically we can calculate a probability for each possible deal
- Seems just like having one big dice roll at the beginning of the game
- Idea: compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals
  - Special case:  
if an action is optimal for all deals, it's optimal.
- GIB, one of the best bridge programs, approximates this idea by
  - ① generating 100 deals consistent with bidding information
  - ② picking the action that wins most tricks on average

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Games of imperfect information

- Example: card games, where opponent's initial cards are unknown
- Typically we can calculate a probability for each possible deal
- Seems just like having one big dice roll at the beginning of the game
- Idea: compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals
  - Special case:  
if an action is optimal for all deals, it's optimal.
- GIB, one of the best bridge programs, approximates this idea by
  - ① generating 100 deals consistent with bidding information
  - ② picking the action that wins most tricks on average

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

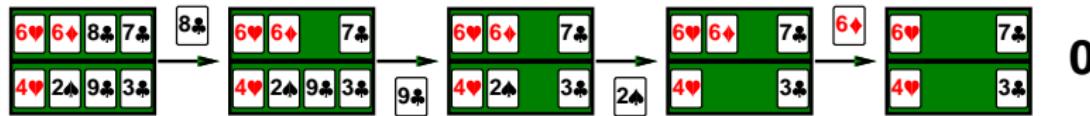
Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Example

Four-card bridge/whist/hearts hand, MAX to play first



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Example

Four-card bridge/whist/hearts hand, MAX to play first



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

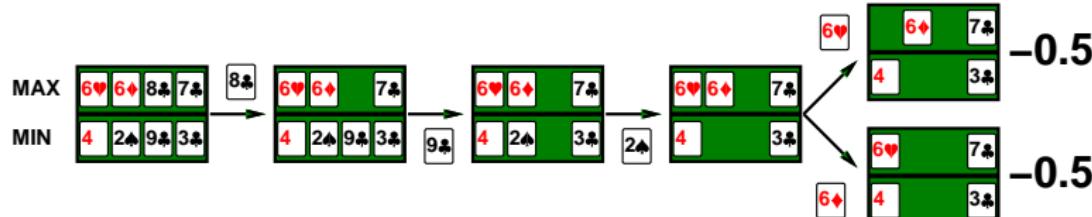
Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Example

Four-card bridge/whist/hearts hand, MAX to play first



Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Commonsense example

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - take the left fork and you'll find a mound of jewels;
  - take the right fork and you'll be run over by a bus.
  
- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - take the left fork and you'll be run over by a bus;
  - take the right fork and you'll find a mound of jewels.
  
- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - guess correctly and you'll find a mound of jewels;
  - guess incorrectly and you'll be run over by a bus.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Commonsense example

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - take the left fork and you'll find a mound of jewels;
  - take the right fork and you'll be run over by a bus.
- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - take the left fork and you'll be run over by a bus;
  - take the right fork and you'll find a mound of jewels.
- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - guess correctly and you'll find a mound of jewels;
  - guess incorrectly and you'll be run over by a bus.

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Proper analysis

- Intuition that the value of an action is the average of its values in all actual states is **WRONG**
- With partial observability, value of an action depends on the **information state** or **belief state** the agent is in
- Can generate and search a tree of information states
- Leads to rational behaviors such as
  - Acting to obtain information
  - Signalling to one's partner
  - Acting randomly to minimize information disclosure

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information

# Summary

Games are to AI as grand prix racing is to automobile design

- They are fun to work on and dangerous

Games illustrate several important points about AI

- perfection is unattainable  $\Rightarrow$  must approximate
- good idea to think about what to think about
- uncertainty constrains the assignment of values to states
- optimal decisions depend on information state, not real state

Introduction  
to AI

Erez Karpas  
Slides by  
Carmel  
Domshlak

Introduction

Perfect play

Resource  
limits and  
approximate  
evaluation

Games of  
chance

Games of  
imperfect  
information