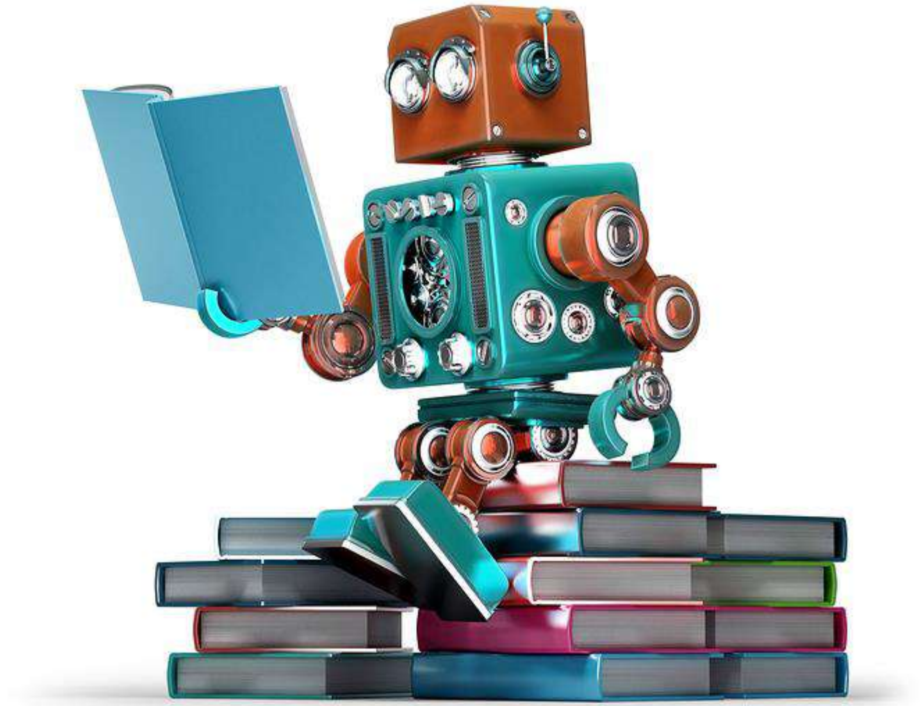
REASONING SYSTEMS
DAY 1



https://robohub.org/wp-content/uploads/2016/11/bigstock-Retro-Robot-Reading-A-Book-Is-110707406.jpg

Google

who is sam gu zhan

All | Images | News | Maps | Videos | More | Settings | Tools

About 335,000 results (0.61 seconds)

[Analytics & Intelligent Systems - NUS-ISS](https://www.iss.nus.edu.sg/about-us/iss-team/teaching.../analytics-intelligent-systems)
https://www.iss.nus.edu.sg/about-us/iss-team/teaching.../analytics-intelligent-systems ▾
Ms. FAN **Zhen Zhen**. Senior Lecturer & Consultant, Analytics & Intelligent ... **GU Zhan**. Mr. **GU Zhan**. Lecturer & Consultant, Analytics & Intelligent Systems ...

[GU Zhan - NUS-ISS](https://www.iss.nus.edu.sg/about-us/staff/detail/201/GU%20Zhan)
https://www.iss.nus.edu.sg/about-us/staff/detail/201/GU%20Zhan ▾
GU Zhan (**Sam**) lectures Master of Technology programme in the areas of data science, machine intelligence, soft computing, and applied deep learning. Prior to ...

[Zhan GU | LinkedIn](https://sg.linkedin.com/in/zhan-gu-27a82823)
https://sg.linkedin.com/in/zhan-gu-27a82823
As a lecturer and consultant in applied machine intelligence, **Zhan GU** (**Sam**) engages communities and schools to help organizations making sense of their ...

[sam gu - Principal Manager - Qualcomm | LinkedIn](https://www.linkedin.com/in/sam-gu-97b65b102)
https://www.linkedin.com/in/sam-gu-97b65b102
San Diego, California - Qualcomm
View **sam gu's** profile on LinkedIn, the world's largest professional community. ... See the complete profile on LinkedIn and discover sam's connections and jobs at similar companies. ... Li **Zhang**. Vice President Engineering at Qualcomm Inc ...
Missing: ~~zhan~~ | Must include: zhan

[GitHub - telescopeuser/GCP-SamGu](https://github.com/telescopeuser/GCP-SamGu)
https://github.com/telescopeuser/GCP-SamGu ▾
Contribute to telescopeuser/GCP-**SamGu** development by creating an account on GitHub. ... Gu **Zhan** support TensorFlow-v1.6. Latest commit 1dd2d5e on Apr ...

Images for who is sam gu zhan

→ More images for who is sam gu zhan | Report images

Google

how to pass exam

how to pass exam**s**
how to pass exam**s without studying**
how to pass exam**s pdf**
how to pass exam**s easily**
how to pass exam**s dominic o'brien pdf**
how to pass exam **p**
how to pass exam**s with top grades**
how to pass exam **in one night**
how to pass exam **fm**
how to pass exam**s without reading**

Report inappropriate predictions

4. Use all of your senses while **studying**:
5. Partner up with someone in your class:
6. Wear a watch:
7. Teach a class of stuffed animals:
8. Drink water and eat fruit:

More items...

[10 Study Hacks That Will Help You Ace Your Final Exams - Business ...](https://www.businessinsider.com/study-hacks-for-final-exams-2013-12)
https://www.businessinsider.com/study-hacks-for-final-exams-2013-12

About this result | Feedback

People also ask

- How do you get good grades without studying?
- Can you pass GED without studying?
- How can I pass the exam?
- How can I study effectively at the last minute?

Feedback

[4 Ways to Pass a Class Without Really Studying - wikiHow](https://www.wikihow.com/Pass-a-Class-Without-Really-Studying)
https://www.wikihow.com/Pass-a-Class-Without-Really-Studying ▾
★★★☆☆ Rating: 56% - 121 votes
Studying may not be your forte, but that shouldn't prevent you from **passing** your class! ... most out of your class time, you may be able to **pass** your class **without studying**. ... Sitting in the same seat may help trigger your memory on **exam** days.

DAY 1 AGENDA

1.1 Reasoning Systems Overview

1.2 Uninformed Search Techniques

1.3 Informed Search Techniques (part 1/2)

1.4 Search Representation **Workshop**

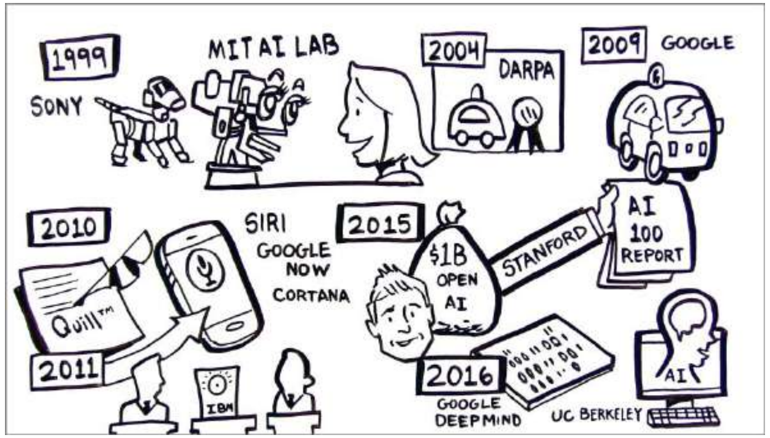
DAY 1 TIMETABLE

No	Time	Topic	By Whom	Where
1	9 am	Welcome and Introduction	GU Zhan (Sam)	Class
2	9.30 am	1.1 Reasoning Systems Overview	GU Zhan (Sam)	Class
3	10.10 am	Morning Break		
4	10.30 am	1.2 Uninformed Search Techniques	GU Zhan (Sam)	Class
5	12.10 pm	Lunch Break		
6	1.30 pm	1.3 Informed Search Techniques (part 1/2)	All	Class
7	3.10 pm	Afternoon Break		
8	3.30 pm	1.4 Search Representation Workshop	All	Class
9	4.50 pm	Summary and Review	All	Class
10	5 pm	End		

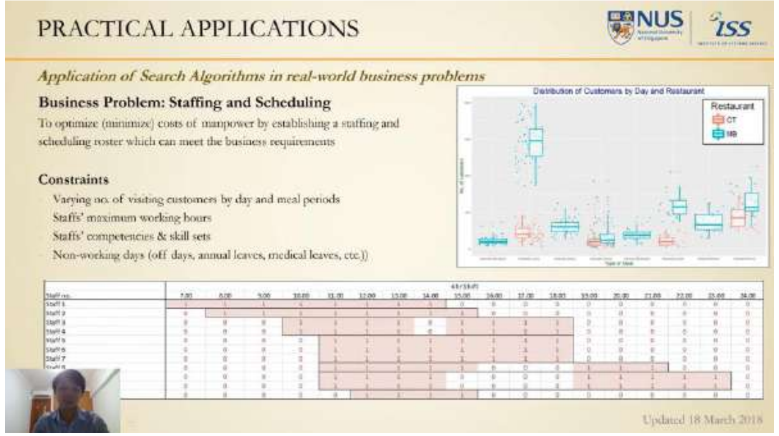
1.1
REASONING SYSTEMS OVERVIEW

- ## AI History

THE
AI
EFFECT



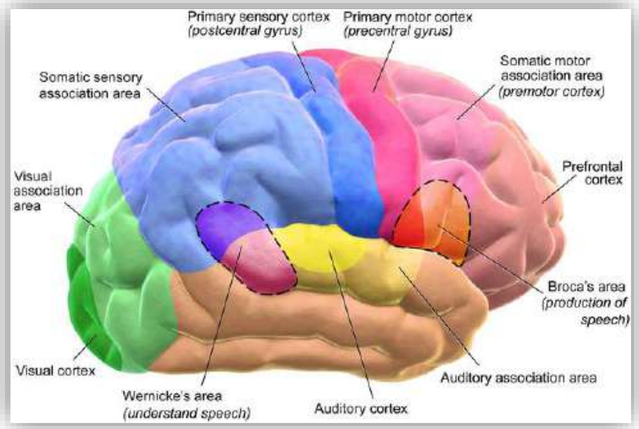
1999
SONY
MIT AI LAB
2004
DARPA
2009
GOOGLE
2010
Quill™
2011
IBM
SIRI
GOOGLE
NOW
CORTANA
2015
$1B
OPEN
AI
STANFORD
AI
100
REPORT
2016
GOOGLE
DEEPMIND
UC BERKELEY
AI

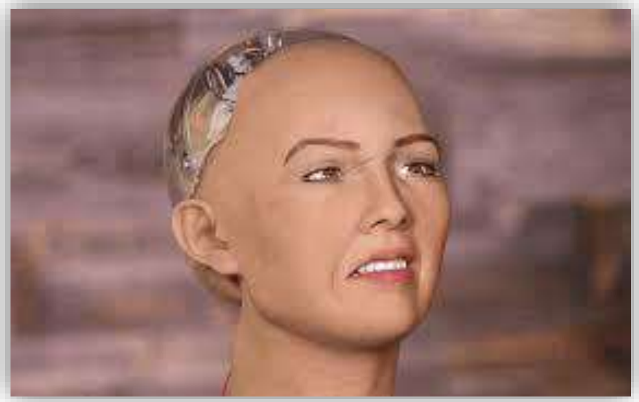


PRACTICAL APPLICATIONS
Application of Search Algorithms in real-world business problems
Business Problem: Staffing and Scheduling
To optimize (minimize) costs of manpower by establishing a staffing and scheduling roster which can meet the business requirements
Constraints
Varying no. of visiting customers by day and meal periods
Staffs' maximum working hours
Staffs' competencies & skill sets
Non-working days (off days, annual leaves, medical leaves, etc.)
Distribution of Customers by Day and Restaurant
Restaurant
CT
Updated 18 March 2018

PRODUCTION
DEVELOPMENT

1.1 REASONING SYSTEMS OVERVIEW

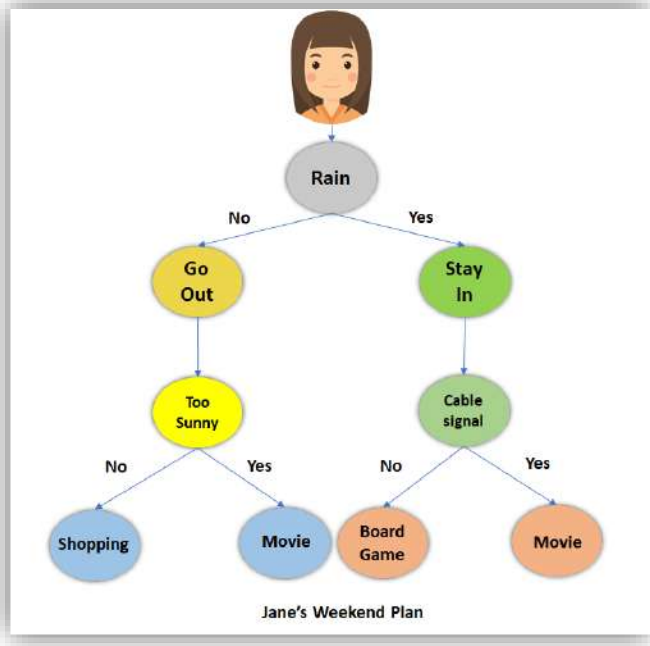




Thinking vs Acting (acting = behaviour)

Human vs Rational (rationality = doing the right thing)

Systems that think like humans (cognitive science)	Systems that think rationally (logic/laws of thought)
Systems that act like humans (c.f. Turing test)	Systems that act rationally (agents)



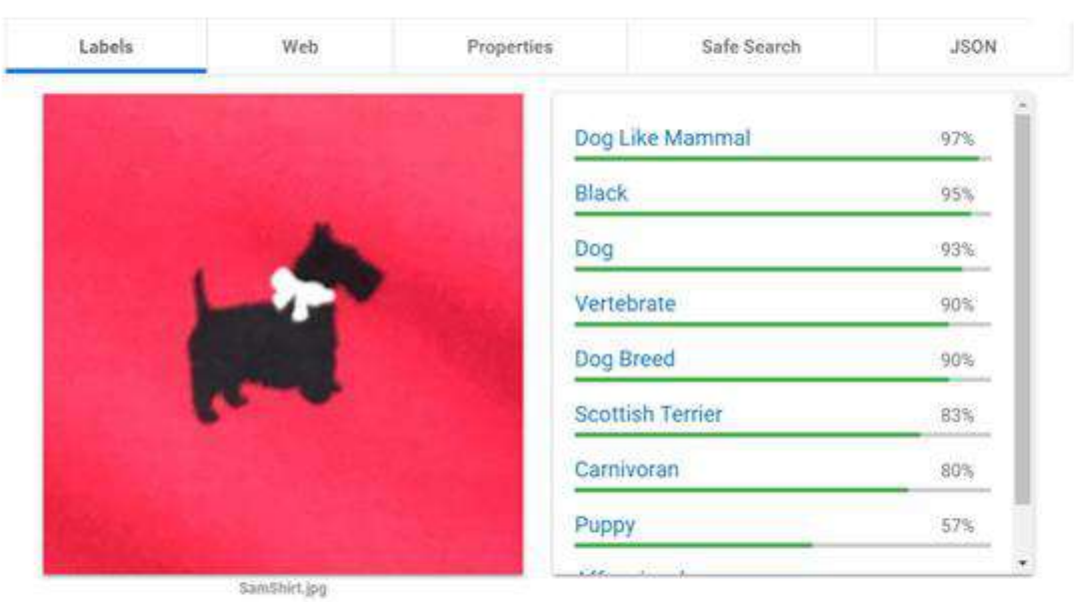
Source https://slideplayer.com/slide/4644026/15/images/20/Systems+that+think+like+humans+%28cognitive+science%29.jpg

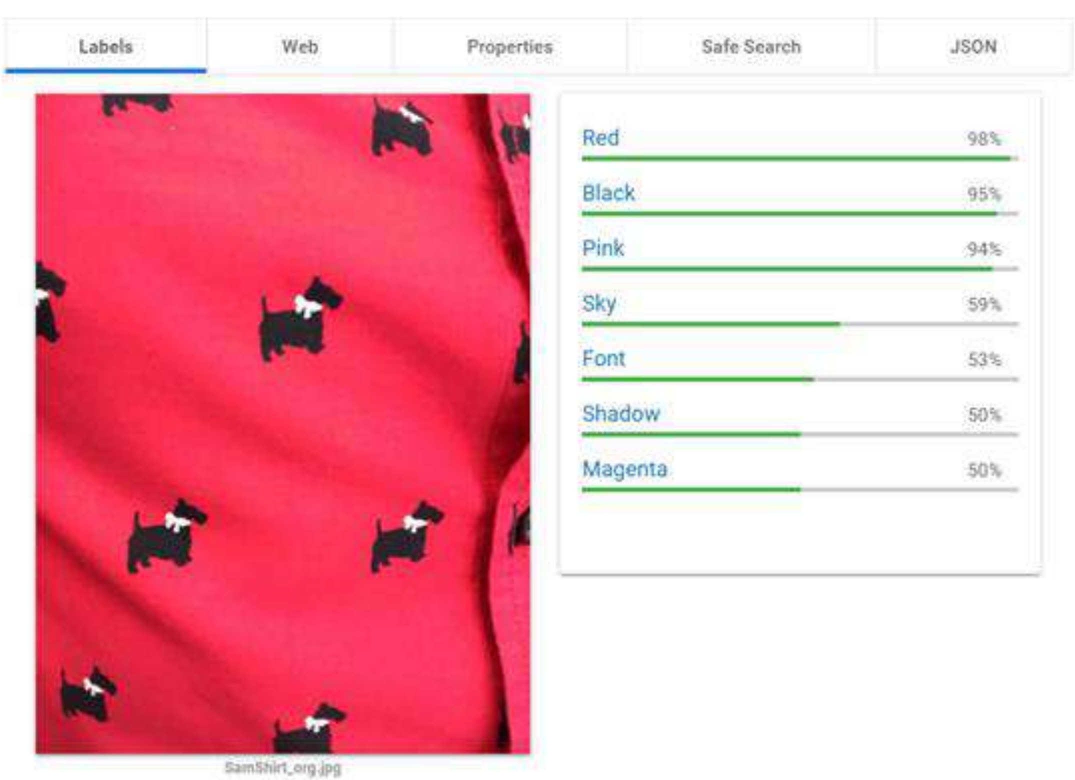
1.1 REASONING SYSTEMS OVERVIEW

- **Question Answering System: IBM Watson**

IBM Watson

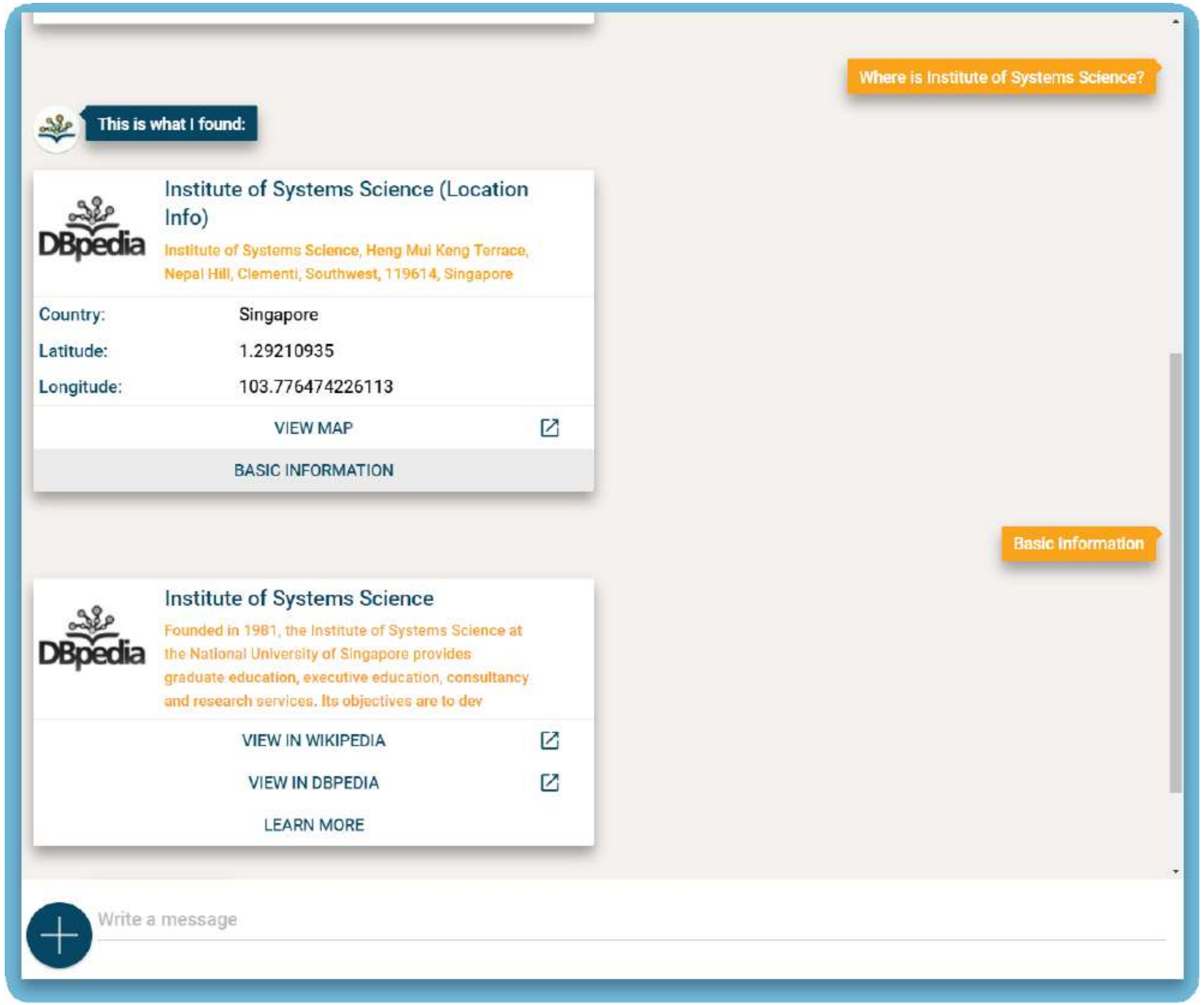
- **Image Object Recognition: Google Vision**





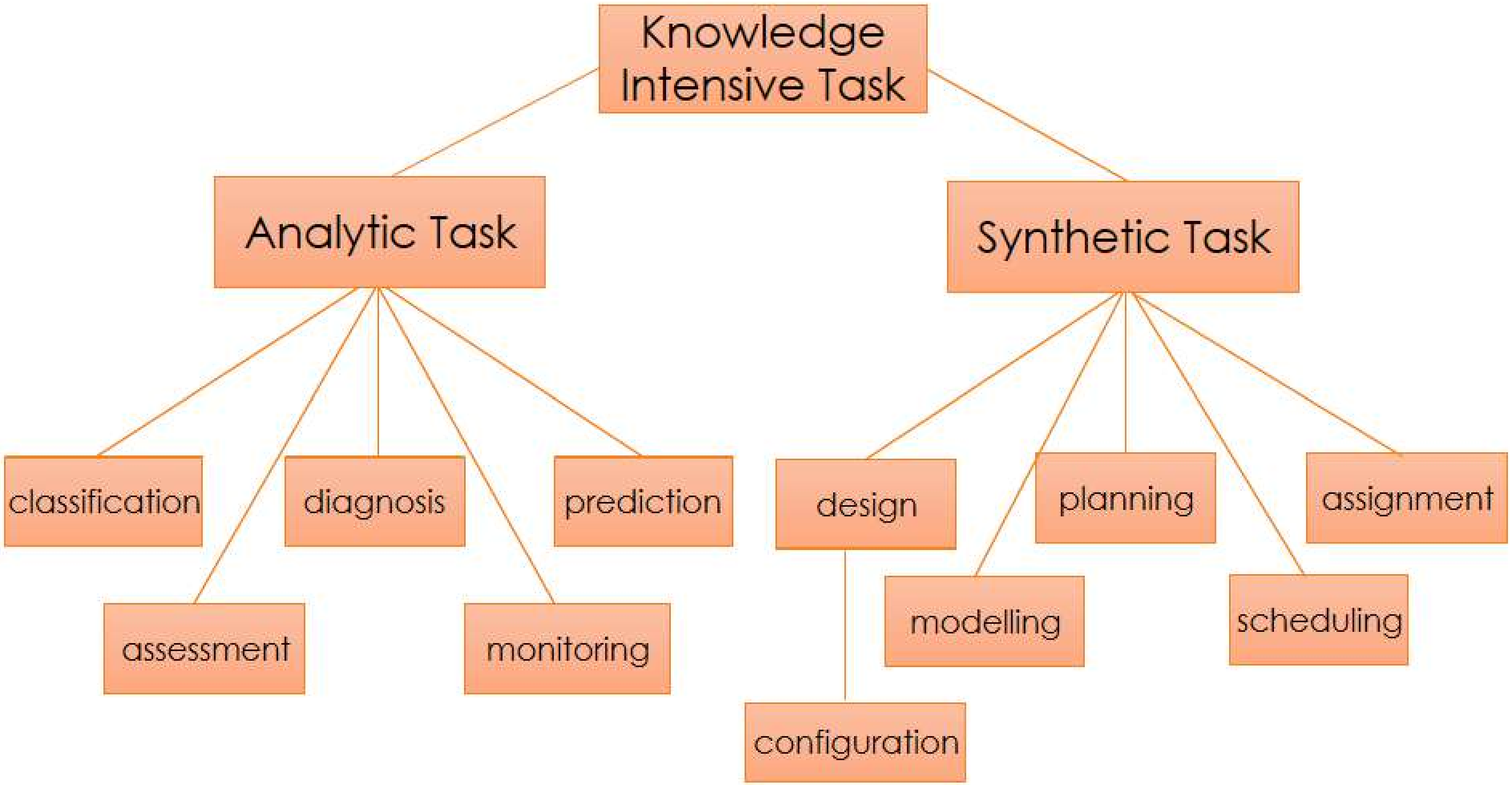
1.1 REASONING SYSTEMS OVERVIEW

- **Chat-Bot: DBpedia**



1.1 REASONING SYSTEMS OVERVIEW

Problem Solving Task Hierarchy



1.1 REASONING SYSTEMS OVERVIEW

Problem Solving Task Types

- **Analytic Tasks**
  - System/Solution to be analysed pre-exists, but usually not completely "known“.
  - Input: some data to trigger the system (e.g. patient symptoms)
  - Output: some characterization or behaviours about the system (e.g. cause of illness)
- **Synthetic Tasks**
  - System/Solution does not yet exist.
  - Input: requirements about system to be constructed
  - Output: constructed system description

1.1 REASONING SYSTEMS OVERVIEW

Problem Solving of Analytic Tasks

- **Analytic Tasks**

  Identification, Classification, Prediction, Clustering/Grouping, …

- **Techniques (Machine Reasoning)**

  Heuristic Business Rules

  Decision Trees

  Case Based Reasoning

  Fuzzy Logic

  Rule Induction

  Machine Learning

  …

1.1 REASONING SYSTEMS OVERVIEW

Problem Solving of Synthetic Tasks

- **Synthetic Tasks**

  Planning, Scheduling, Optimisation, Design, …

- **Techniques (Reasoning Systems)**

  Uninformed (brute force / blind) Search

  Informed (heuristic) Search

  Simulations

  Genetic Algorithms

  Reinforcement Learning

  Data Mining

  …

1.1 REASONING SYSTEMS OVERVIEW

- **Exercise: How to pass this course?**
  - Analytic Tasks?
  - Synthetic Tasks?
  - Your proposed solution (sub tasks)?

1.2
UNINFORMED SEARCH TECHNIQUES

1.2 UNINFORMED SEARCH TECHNIQUES

- Solving Problem by Search
- Search Tree Representation
- Depth First Search (DFS)
- Breadth First Search (BFS)

https://modernmarketingtoday.com/wp-content/uploads/2013/02/search-marketing.jpg

1.2 UNINFORMED SEARCH TECHNIQUES

Solving Problem by Search

- Synthesis of a new valid solution is performed by searching through the (search/solution) space, which contains all possible solutions
- Each possible solution is evaluated to see whether it is valid and/or the optimum (best solution found by now), e.g. a valid employee schedule, a valid vehicle delivery route, an optimal (shortest) vehicle delivery route,
- Validity of solution involves satisfaction of a set of constraints on the solution variables
- Optimality is measured by a user-defined function which measures the "goodness" of the solution, e.g. the shorter delivery route the better.

1.2 UNINFORMED SEARCH TECHNIQUES

Solving Problem by Search

(1) Create a pool of solution candidates (search space)

(2) Pick up one candidate solution from pool

(3) Check whether this candidate is valid (constraints satisfied?)

- (3)=True — If valid, continue
- (3)=False — If not valid, go to (2)

(4) Check whether this candidate is the best till now (optimal solution?)

- (4)=True — If best, save this solution as the best then continue
- (4)=False — If not best, discard this solution then continue

(5) Go to (2). Repeat the cycle until a stopping criteria is met.

1.2 UNINFORMED SEARCH TECHNIQUES

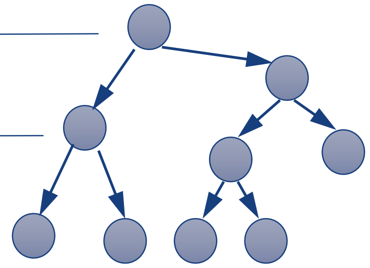
Search Tree Representation

- **Search is illustrated using a search space with a particular restricted structure**
- **Solutions (search space) can be represented as a Tree**
  - Nodes in tree represent

    an initial state

    an intermediate state

    a final state (feasible solution, or failure)



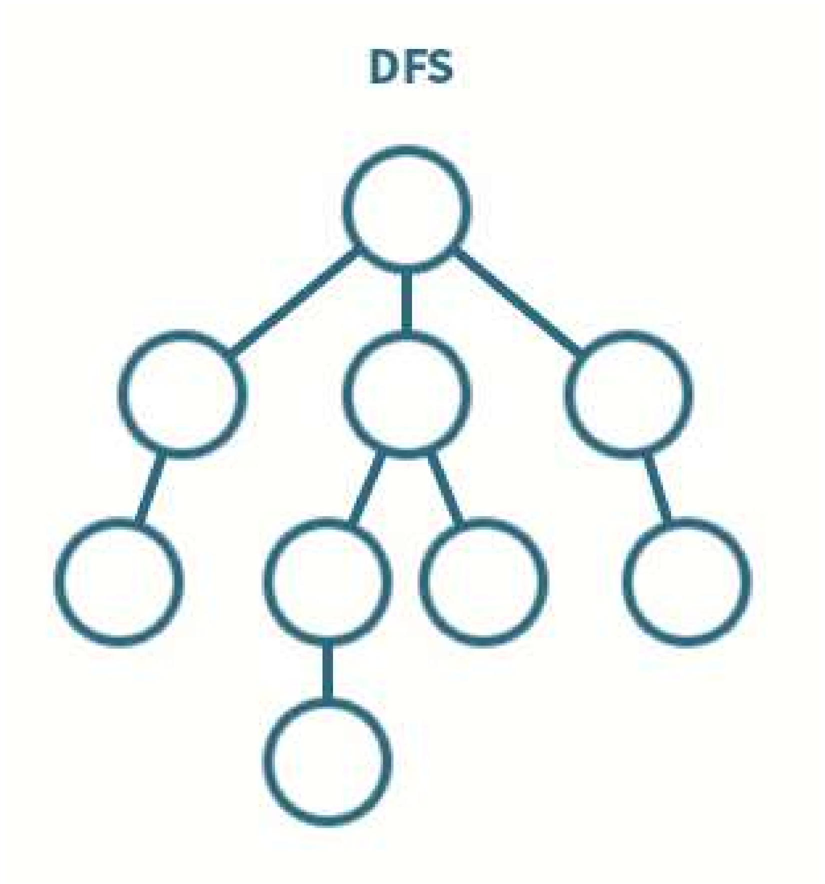
  - Connection between nodes represents a search step

Depth First Search (DFS)

- Always prefers to search deeper in the search tree rather than wider.

1.2 UNINFORMED SEARCH TECHNIQUES
Depth First Search (DFS)

- **Visit order**



https://medium.com/@kenny.hom27/breadth-first-vs-depth-first-tree-traversal-in-javascript-48df2ebfc6d1

1.2 UNINFORMED SEARCH TECHNIQUES

Depth First Search (DFS)

Algorithm Pseudo Code

(1) Set N to be a list of initial nodes

(2) If N is empty, then exit and signal failure

(3) Set n to be the first node in N, and remove n from N

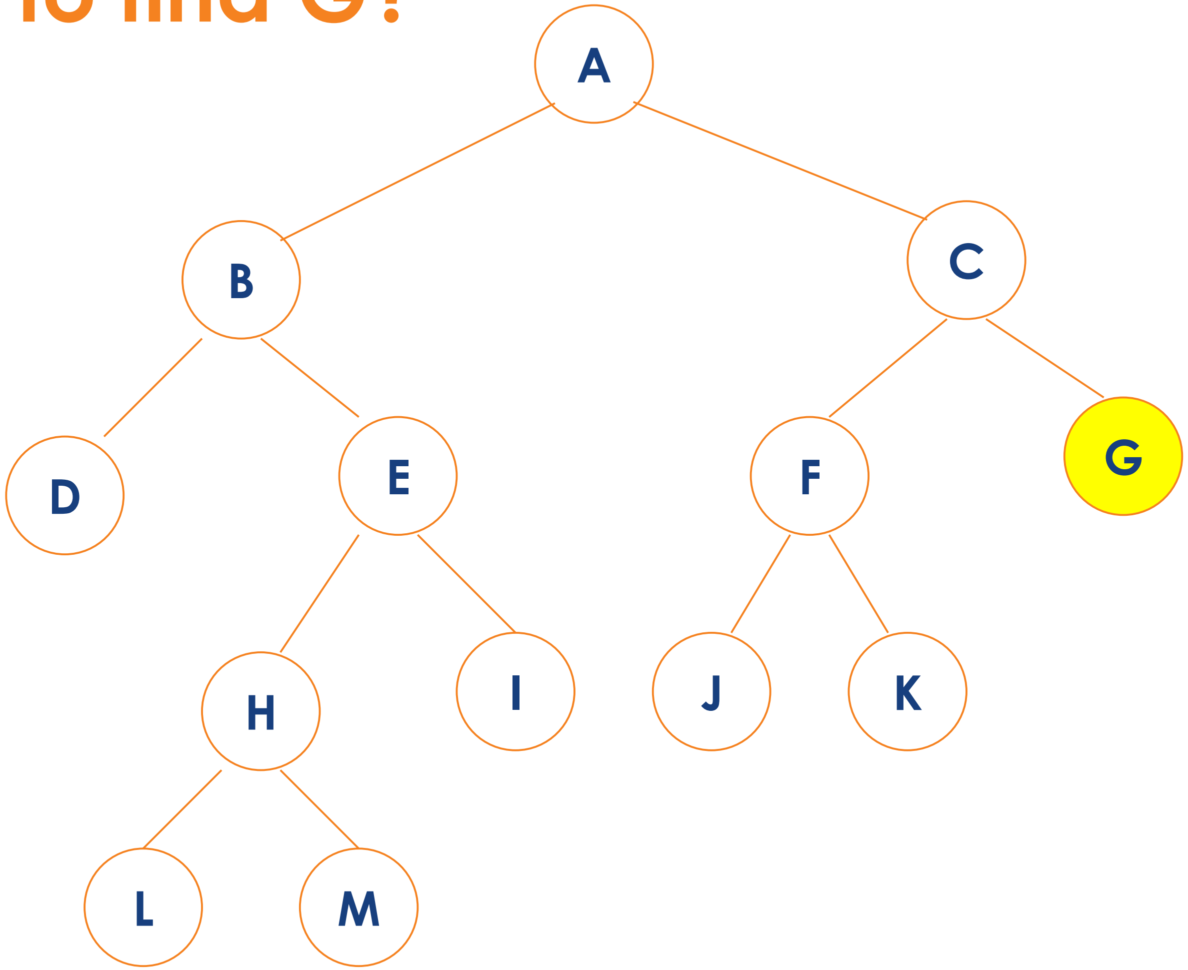
(4) Check n:

- **(4.1)** If **n** is a goal node, then exit and signal success
- **(4.2)** Otherwise, add the children of **n** to the **front** of **N** then go to step **(2)**

1.2 UNINFORMED SEARCH TECHNIQUES

Depth First Search (DFS)

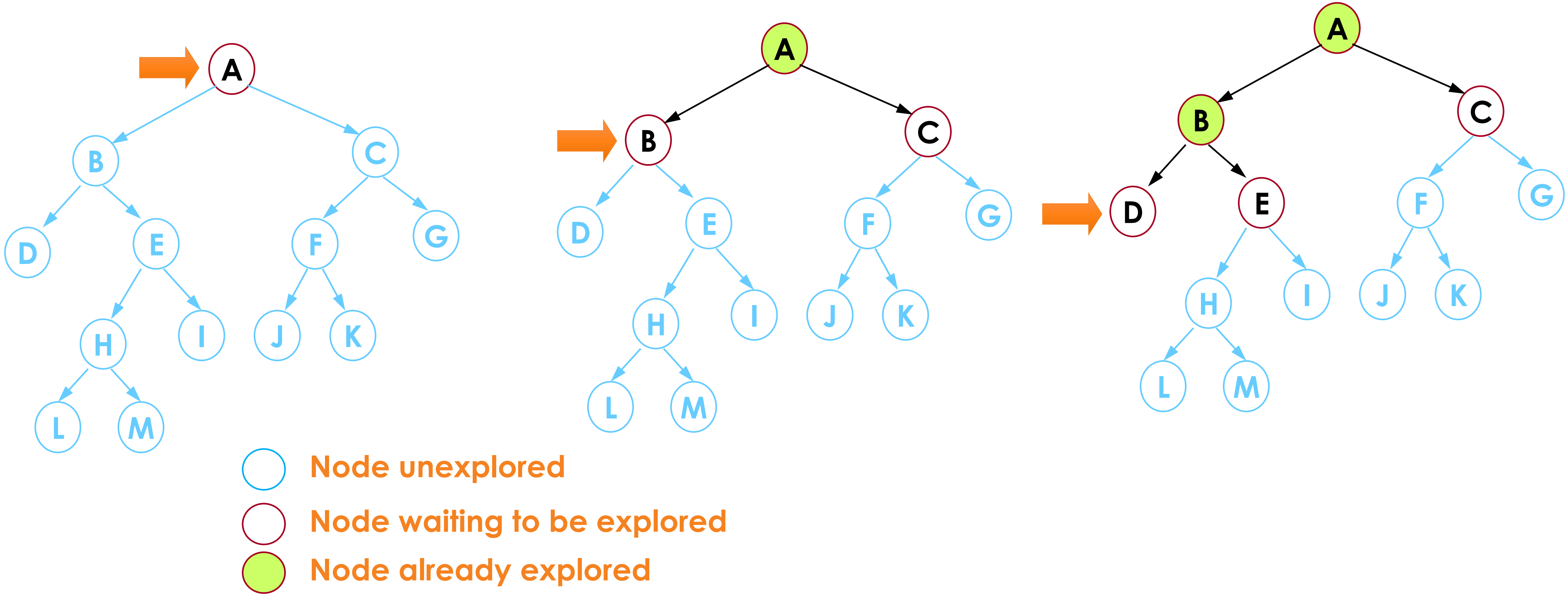
- Visit order to find G?



1.2 UNINFORMED SEARCH TECHNIQUES

Depth First Search (DFS)

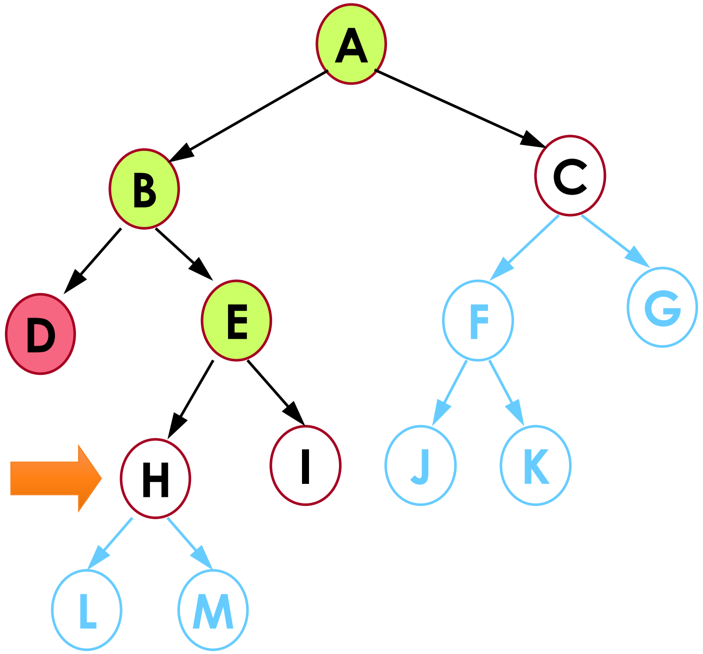
- Keep track of nodes

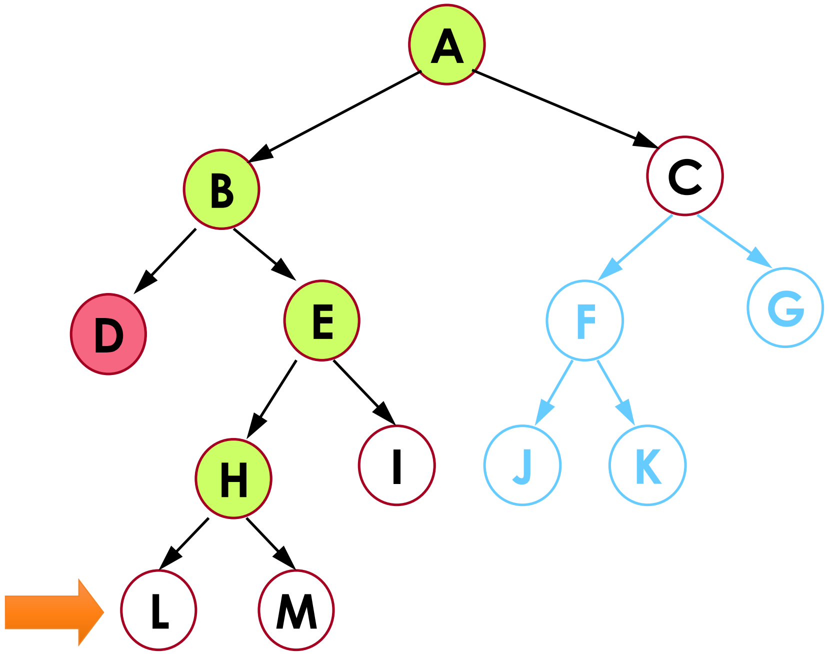


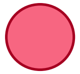
1.2 UNINFORMED SEARCH TECHNIQUES

Depth First Search (DFS)

- Keep track of nodes



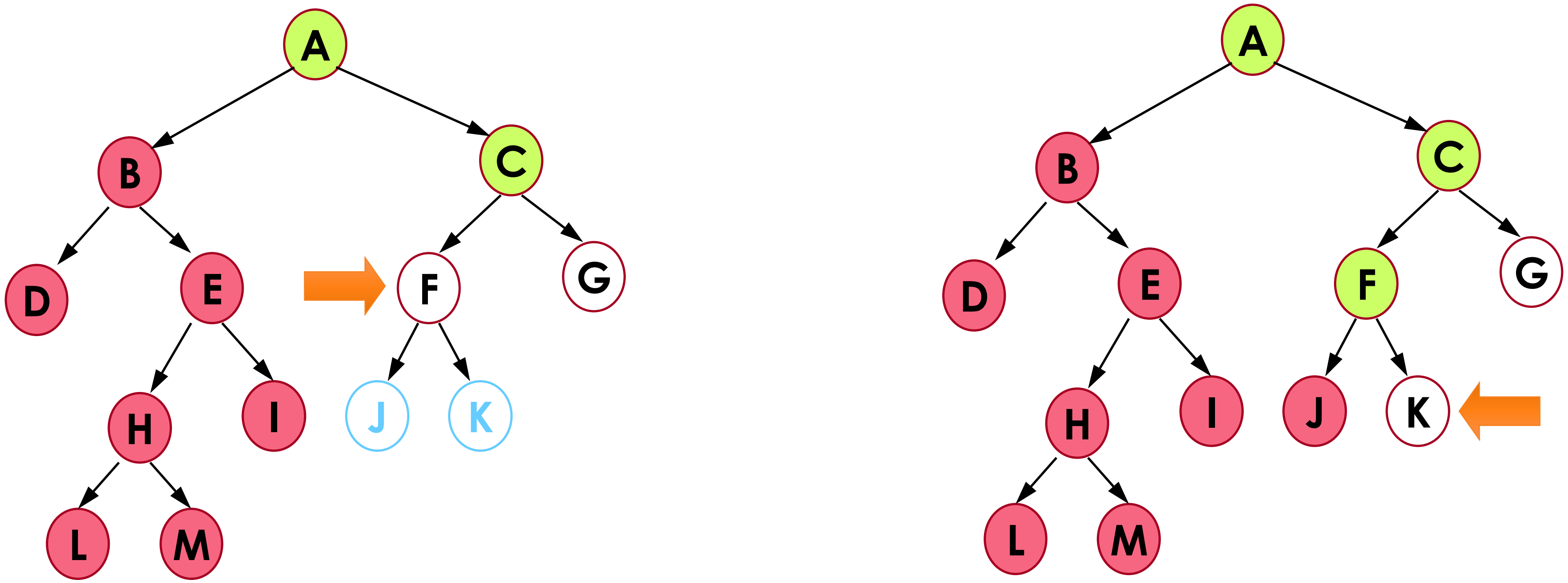



Explored non-solution node can be removed

1.2 UNINFORMED SEARCH TECHNIQUES

Depth First Search (DFS)

- Keep track of nodes

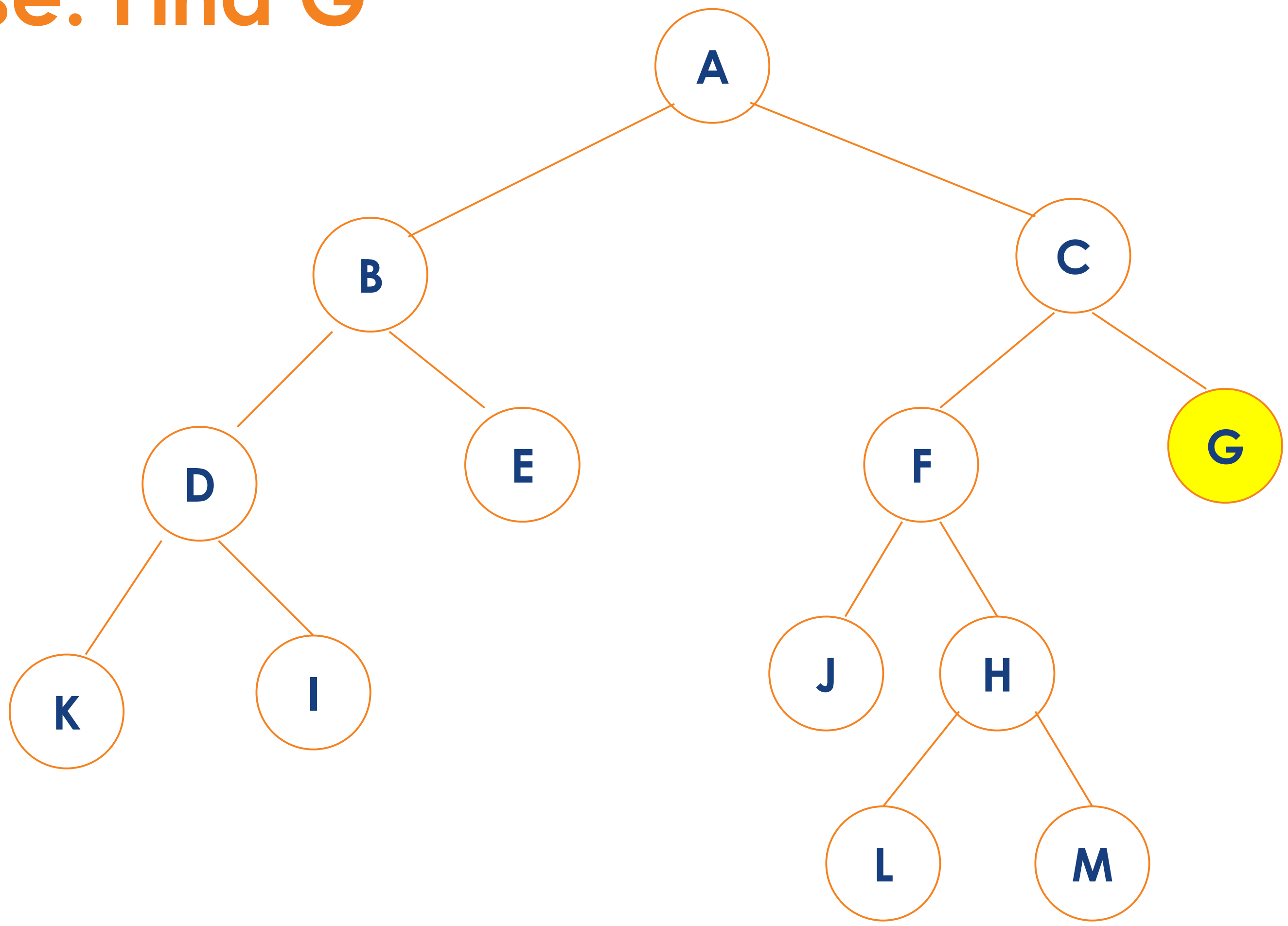


Explored non-solution node/branch can be removed

1.2 UNINFORMED SEARCH TECHNIQUES

Depth First Search (DFS)

- Exercise: Find G



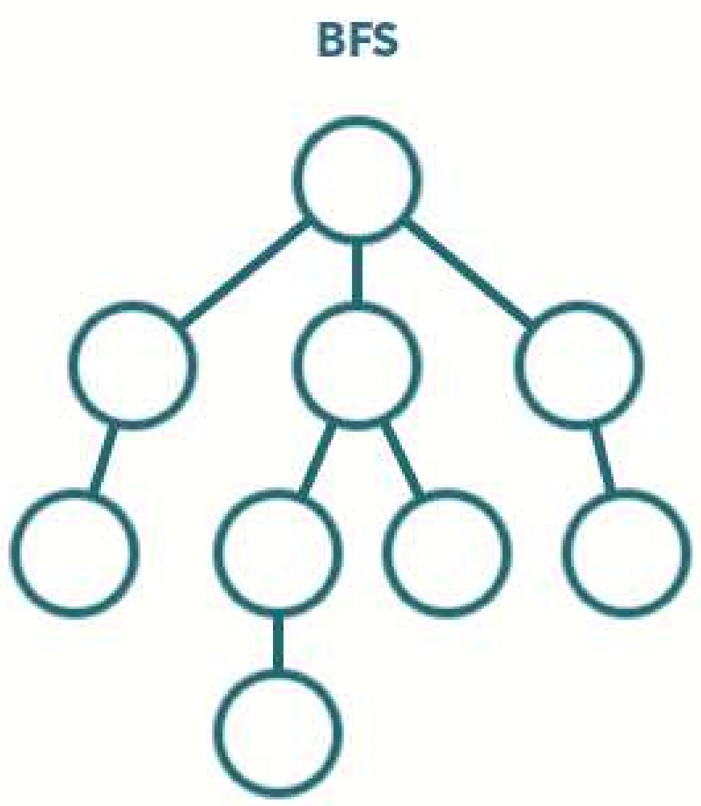
Breadth First Search (BFS)

- Explores all the nodes at a given depth before processing deeper in the search tree.

1.2 UNINFORMED SEARCH TECHNIQUES

Breadth First Search (BFS)

- **Visit order**



https://medium.com/@kenny.hom27/breadth-first-vs-depth-first-tree-traversal-in-javascript-48df2ebfc6d1

Algorithm Pseudo Code

(1) Set N to be a list of initial nodes

(2) If N is empty, then exit and signal failure

(3) Set n to be the first node in N, and remove n from N

(4) Check n:

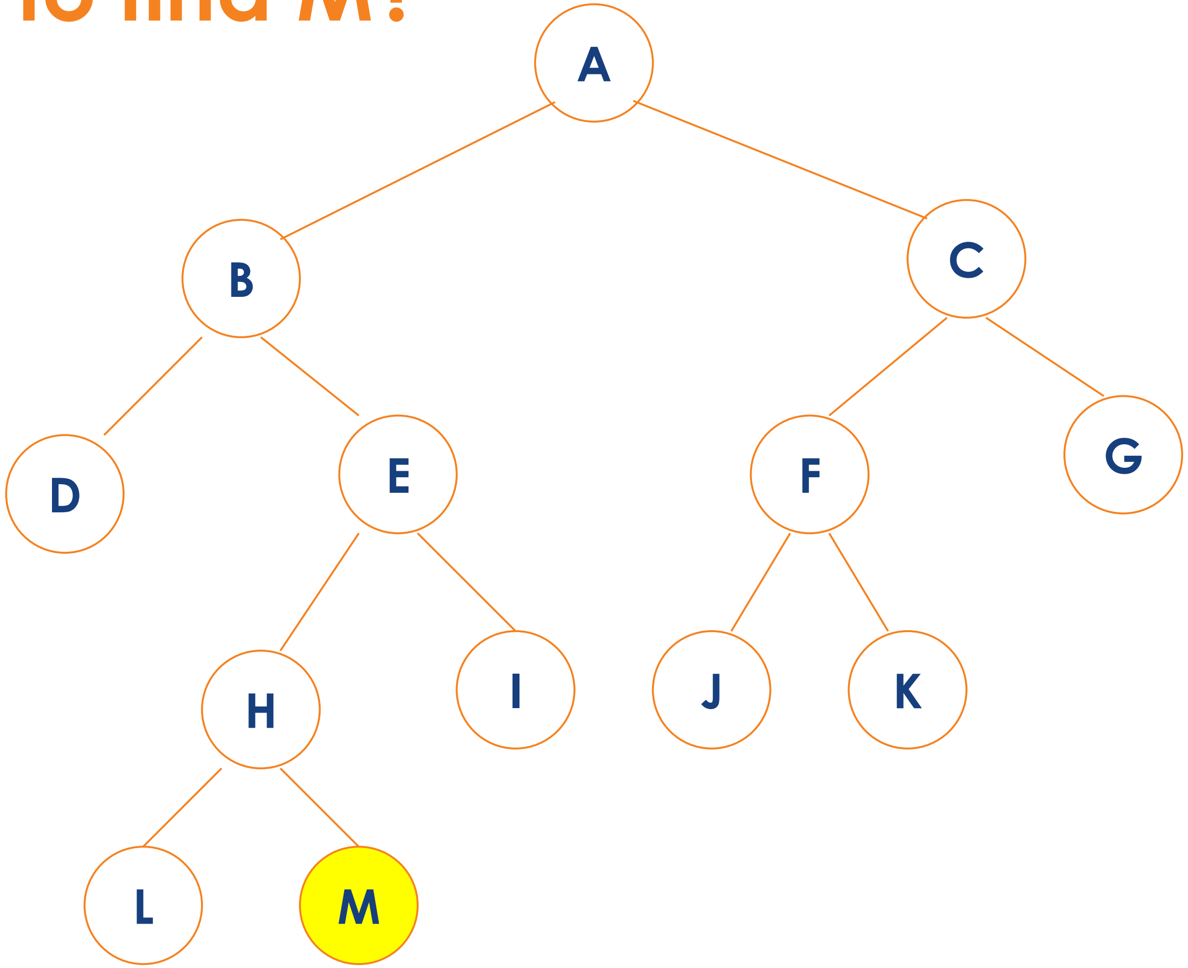
(4.1) If n is a goal node, then exit and signal success

(4.2) Otherwise, add the children of n to the end of N then go to step (2)

1.2 UNINFORMED SEARCH TECHNIQUES

Breadth First Search (BFS)

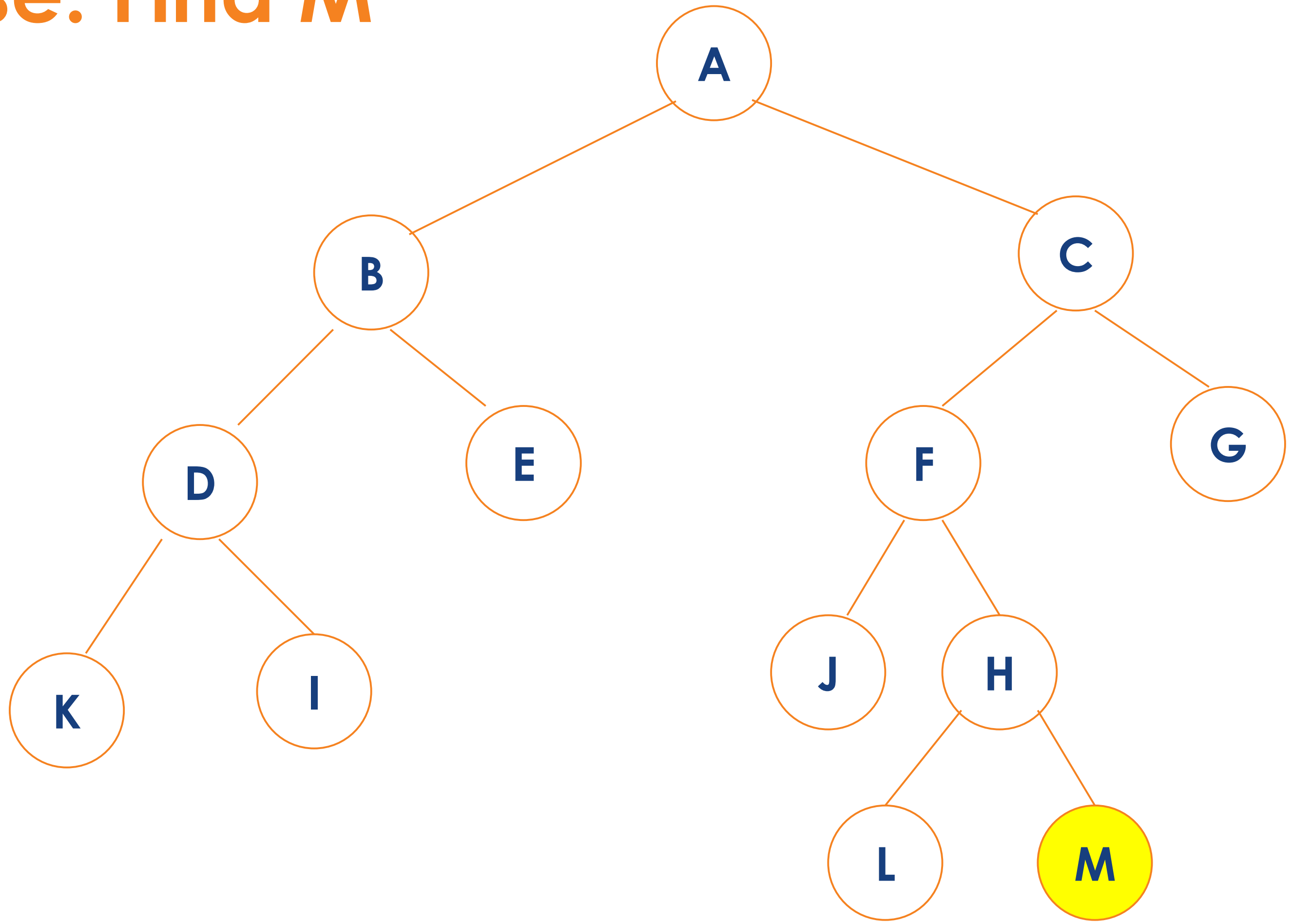
- Visit order to find M?



1.2 UNINFORMED SEARCH TECHNIQUES

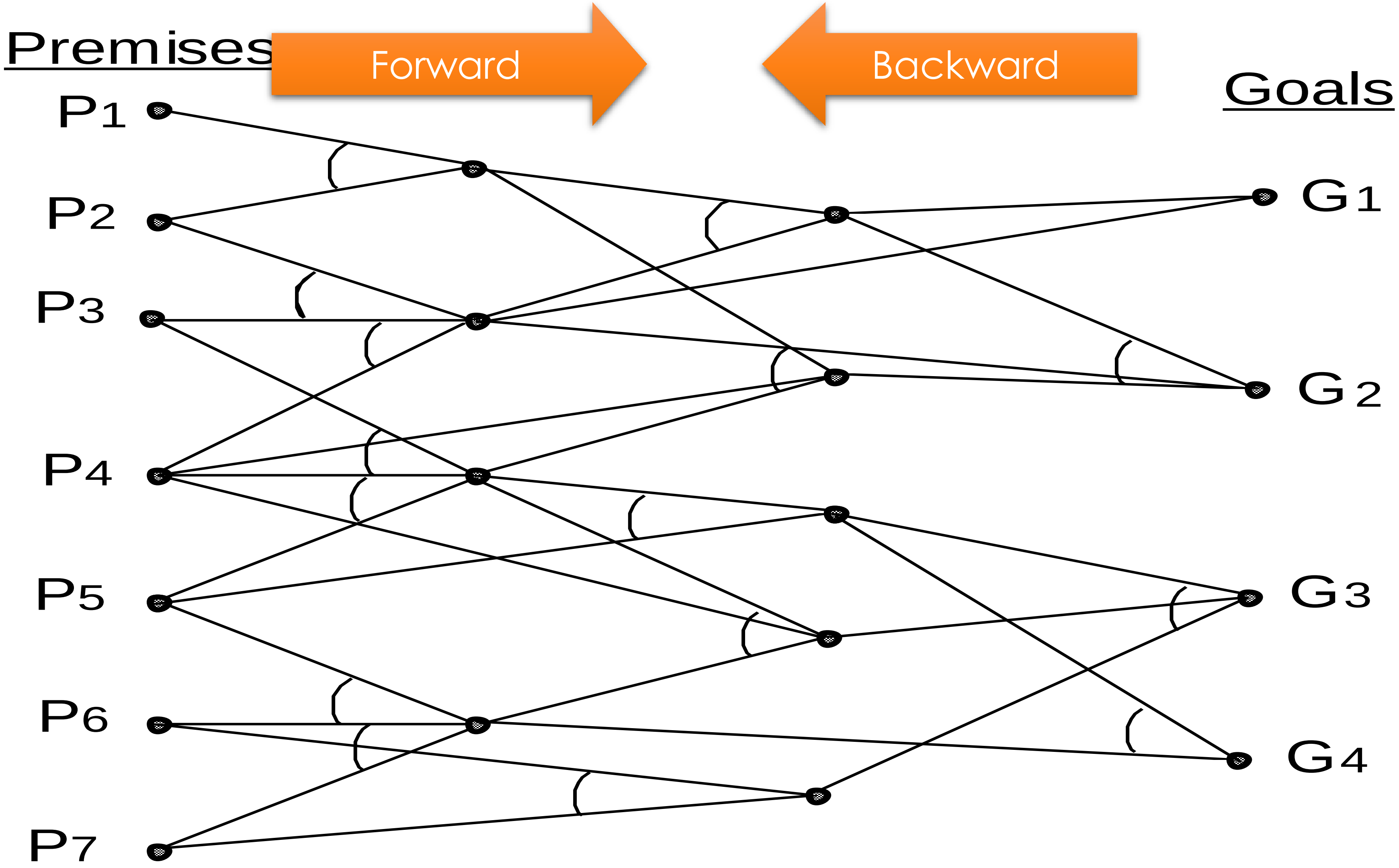
Breadth First Search (BFS)

- Exercise: Find M



1.2 UNINFORMED SEARCH TECHNIQUES

Forward Chaining (BFS) vs. Backward Chaining (DFS)



3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

X is an (instance of) animal (class).

1. HasHair(X) → Mammal(X)
2. FeedMilk(X) → Mammal(X)
3. HasFeather(X) → Bird(X)
4. CanFly(X) ∧ LayEgg(X) → Bird(X)
5. Mammal(X) ∧ EatMeat(X) → Carnivore(X)
6. Mammal(X) ∧ HasPointedTeeth(X) ∧ HasClaws(X) ∧ HasForwardEyes(X) → Carnivore(X)
7. Mammal(X) ∧ EatGrass(X) → Herbivore(X)
8. Mammal(X) ∧ HasHooves(X) → Herbivore(X)
9. Carnivore(X) ∧ HasColorTawny(X) ∧ HasDarkSpots(X) → Cheetah(X)
10. Carnivore(X) ∧ HasColorTawny(X) ∧ HasDarkStripes(X) → Tiger(X)
11. Herbivore(X) ∧ HasColorTawny(X) ∧ HasDarkSpots(X) ∧ HasLongNeck(X) → Giraffe(X)
12. Herbivore(X) ∧ HasColorBlackWhite(X) → Zebra(X)
13. Bird(X) ∧ CanWalk(X) ∧ HasColorBlackWhite(X) ∧ HasLongNeck(X) → Ostrich(X)
14. Bird(X) ∧ CanSwim(X) ∧ HasColorBlackWhite(X) → Penguin(X)
15. Bird(X) ∧ CanFly(X) ∧ HasColorBlackWhite(X) → Albatross(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

- **Knowledge Base (KB)**
  - Rule sets plus below Facts:
  - HasHair(X)
  - HasClaws(X)
  - HasPointedTeeth(X)
  - HasForwardEyes(X)
  - HasColorTawny(X)
  - HasDarkSpots(X)
- **Clouse form conversion :** $p \rightarrow q \equiv \neg p \vee q$
- **Hypothesis to prove is :** a = Cheetah(X)
- **Refutation of hypothesis is :** $\neg$a = $\neg$Cheetah(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. HasHair(X) → Mammal(X)
2. FeedMilk(X) → Mammal(X)
3. HasFeather(X) → Bird(X)
4. CanFly(X) ∧ LayEgg(X) → Bird(X)
5. Mammal(X) ∧ EatMeat(X) → Carnivore(X)
6. Mammal(X) ∧ HasPointedTeeth(X) ∧ HasClaws(X) ∧ HasForwardEyes(X) → Carnivore(X)
7. Mammal(X) ∧ EatGrass(X) → Herbivore(X)
8. Mammal(X) ∧ HasHooves(X) → Herbivore(X)
9. Carnivore(X) ∧ HasColorTawny(X) ∧ HasDarkSpots(X) → Cheetah(X)
10. Carnivore(X) ∧ HasColorTawny(X) ∧ HasDarkStripes(X) → Tiger(X)
11. Herbivore(X) ∧ HasColorTawny(X) ∧ HasDarkSpots(X) ∧ HasLongNeck(X) → Giraffe(X)
12. Herbivore(X) ∧ HasColorBlackWhite(X) → Zebra(X)
13. Bird(X) ∧ CanWalk(X) ∧ HasColorBlackWhite(X) ∧ HasLongNeck(X) → Ostrich(X)
14. Bird(X) ∧ CanSwim(X) ∧ HasColorBlackWhite(X) → Penguin(X)
15. Bird(X) ∧ CanFly(X) ∧ HasColorBlackWhite(X) → Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v Cheetah(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v Cheetah(X) ^ ¬ Cheetah(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v {}
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) ^ HasDarkSpots(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) v {}
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v ¬ HasColorTawny(X) ^ HasColorTawny(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X) v {}
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v Carnivore(X) ʌ ¬ Carnivore(X)
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v Carnivore(X) ʌ ¬ Carnivore(X)
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v {} ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) v ¬ HasPointedTeeth(X) v ¬ HasClaws(X) v ¬ HasForwardEyes(X) v {} ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) v {} ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Which branch to search/check next? Use conflict resolution strategy, ***e.g. Less No. Sub-goals***

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X)
2. ¬ FeedMilk(X) v Mammal(X)
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v Mammal(X) ∧ ¬ Mammal(X) ← branch 1.1
2. ¬ FeedMilk(X) v Mammal(X) ∧ ¬ Mammal(X) ← branch 1.2
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) v {} ← branch 1.1
2. ¬ FeedMilk(X) v {} ← branch 1.2
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. ¬ HasHair(X) ^ HasHair(X) ← branch 1.1
2. ¬ FeedMilk(X) v {} ← branch 1.2
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. {} ← branch 1.1
2. ¬ FeedMilk(X) v {} ← branch 1.2
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

3.1 MACHINE INFERENCE

Exercise 3.1 – A possible solution: first order logic

1. {} ← a = Cheetah(X) proved by refutation ← branch 1.1
2. ¬ FeedMilk(X) v {} ← branch 1.2
3. ¬ HasFeather(X) v Bird(X)
4. ¬ CanFly(X) v ¬ LayEgg(X) v Bird(X)
5. ¬ Mammal(X) v ¬ EatMeat(X) v {} ← branch 2
6. ¬ Mammal(X) ← branch 1
7. ¬ Mammal(X) v ¬ EatGrass(X) v Herbivore(X)
8. ¬ Mammal(X) v ¬ HasHooves(X) v Herbivore(X)
9. ¬ Carnivore(X)
10. ¬ Carnivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkStripes(X) v Tiger(X)
11. ¬ Herbivore(X) v ¬ HasColorTawny(X) v ¬ HasDarkSpots(X) v ¬ HasLongNeck(X) v Giraffe(X)
12. ¬ Herbivore(X) v ¬ HasColorBlackWhite(X) v Zebra(X)
13. ¬ Bird(X) v ¬ CanWalk(X) v ¬ HasColorBlackWhite(X) v ¬ HasLongNeck(X) v Ostrich(X)
14. ¬ Bird(X) v ¬ CanSwim(X) v ¬ HasColorBlackWhite(X) v Penguin(X)
15. ¬ Bird(X) v ¬ CanFly(X) v ¬ HasColorBlackWhite(X) v Albatross(X)

Facts:
HasHair(X)
HasClaws(X)
HasPointedTeeth(X)
HasForwardEyes(X)
HasColorTawny(X)
HasDarkSpots(X)
¬ Cheetah(X)
¬ Carnivore(X)
¬ Mammal(X)

1.3
INFORMED SEARCH TECHNIQUES
(PART 1/2)

1.3 INFORMED SEARCH TECHNIQUES (1/2)

- Use Heuristics
- Hill Climbing Search (HC)
- A Star Search (A*)
- Tabu Search (TS)
- Simulated Annealing (SA)
- Informed Search Use Case

https://modernmarketingtoday.com/wp-content/uploads/2013/02/search-marketing.jpg

1.3 INFORMED SEARCH TECHNIQUES (1/2)

Use Heuristics

- **Basic Idea**
  - It works by firstly sorting the list of nodes, then explore them orderly, according to their **optimality** (best score) determined by an evaluation function ***f*(*n*)**
- **Typical Best-first Strategies**
  - Use heuristics only : Hill Climbing, Tabu search
  - Use heuristics and past cost : A*, Late Acceptance
  - Use heuristics and randomness : Simulated Annealing, GA

1.3 INFORMED SEARCH TECHNIQUES (1/2)

Use Heuristics

- A key component of evaluation function *f(n)* is a heuristic function: *h(n)* = estimated cost of the cheapest path from node n to a goal node. Or estimated degree of difference between the current states/solutions and ultimate goal state.
- By convention, the lower the heuristic value the more promising the node: better to check first. h(n) = 0 when n is goal
- ☺ When there is no other information available but only the heuristics, *f(n)* = *h(n)*

1.3 INFORMED SEARCH TECHNIQUES (1/2)

Use Heuristics

- **Heuristic Function (Knowledge)**
  - ***h(n)*** = Correct or Incorrect positions

Initial=

1	3	2
8		4
5	6	7

Goal=

1	2	3
8		4
7	6	5

  - ***h_1(n)*** = number of tiles in their **correct** goal state positions

1	3	2
8		4
5	6	7

= 4

  - ***h_2(n)*** = number of tiles in their **incorrect** goal state positions

1	3	2
8		4
5	6	7

= 4

1.3 INFORMED SEARCH TECHNIQUES (1/2)

Use Heuristics

- **Heuristic Function (Knowledge)**
  - ***h(n)*** = Manhattan distance (sum of the horizontal & vertical distance each tile is away from its goal state position)

Goal=

1	2	3
8		4
7	6	5

1		2
8	3	4
5	6	7

$\boxed{3}$ $\boxed{2}$ $\boxed{7}$ $\boxed{5}$

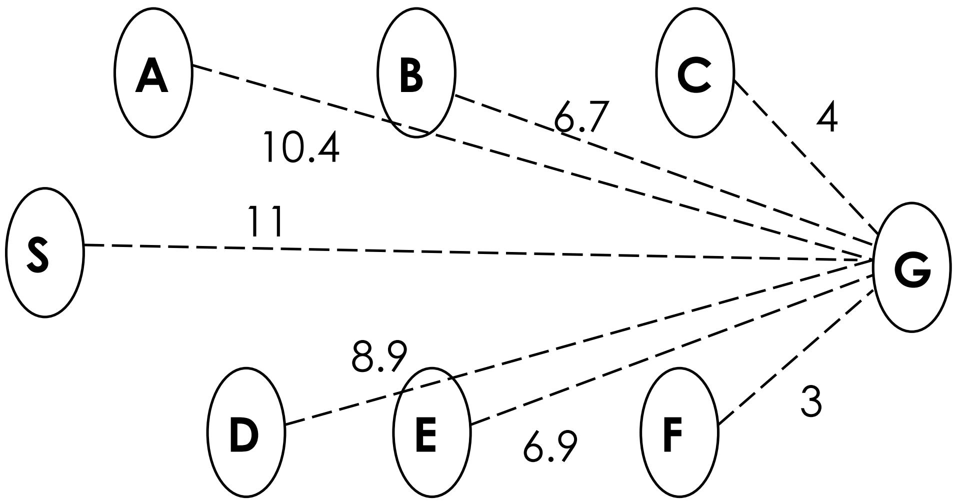
$= (1+1) + 1 + 2 + 2 = 7$

☺ **Manhattan distance gives a better estimate of the distance to the Goal state**

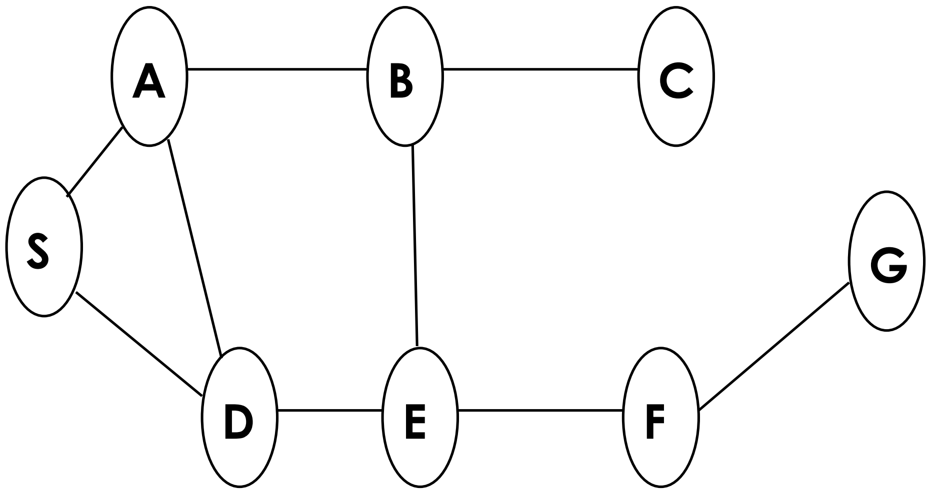
1.3 INFORMED SEARCH TECHNIQUES (1/2)

Use Heuristics

- **Heuristic Function (Knowledge)**
  - *h(n)* = “straight-line” distance between each city & the goal (This is useful estimation or heuristic.)



Heuristic Distances



Actual Roads

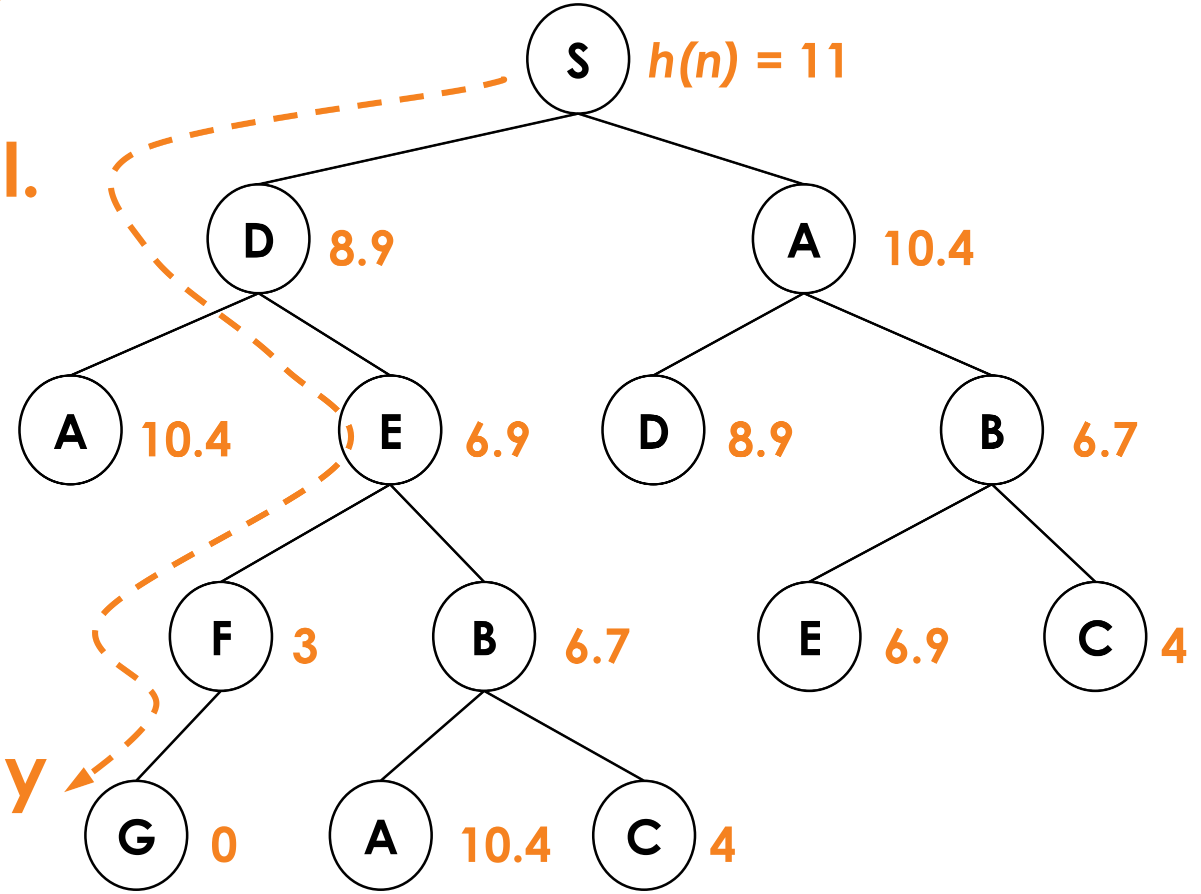
1.3 INFORMED SEARCH TECHNIQUES (1/2)

Hill Climbing (Greedy Best First Search)

- Minimize the cost to reach the goal state by expanding the node that is closest to the goal.
- Using only the heuristic values for evaluation function:

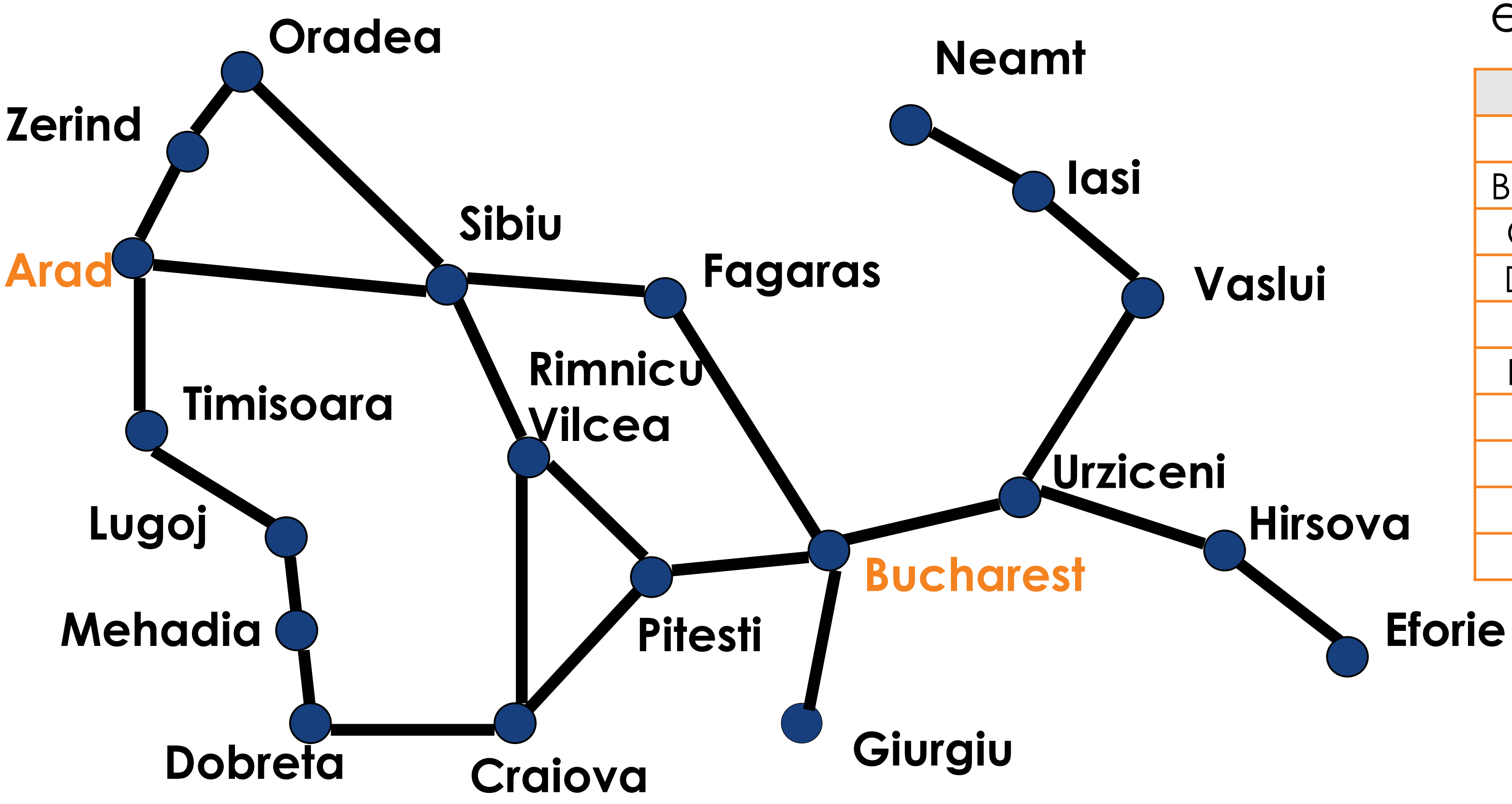
$$f(n) = h(n)$$

- Select search node with min($f(n)$) at each step.
- Follow a single path all the way to a goal, but can back track when it hits a dead end.



Hill Climbing (Greedy Best First Search)

- **Arad → Bucharest**

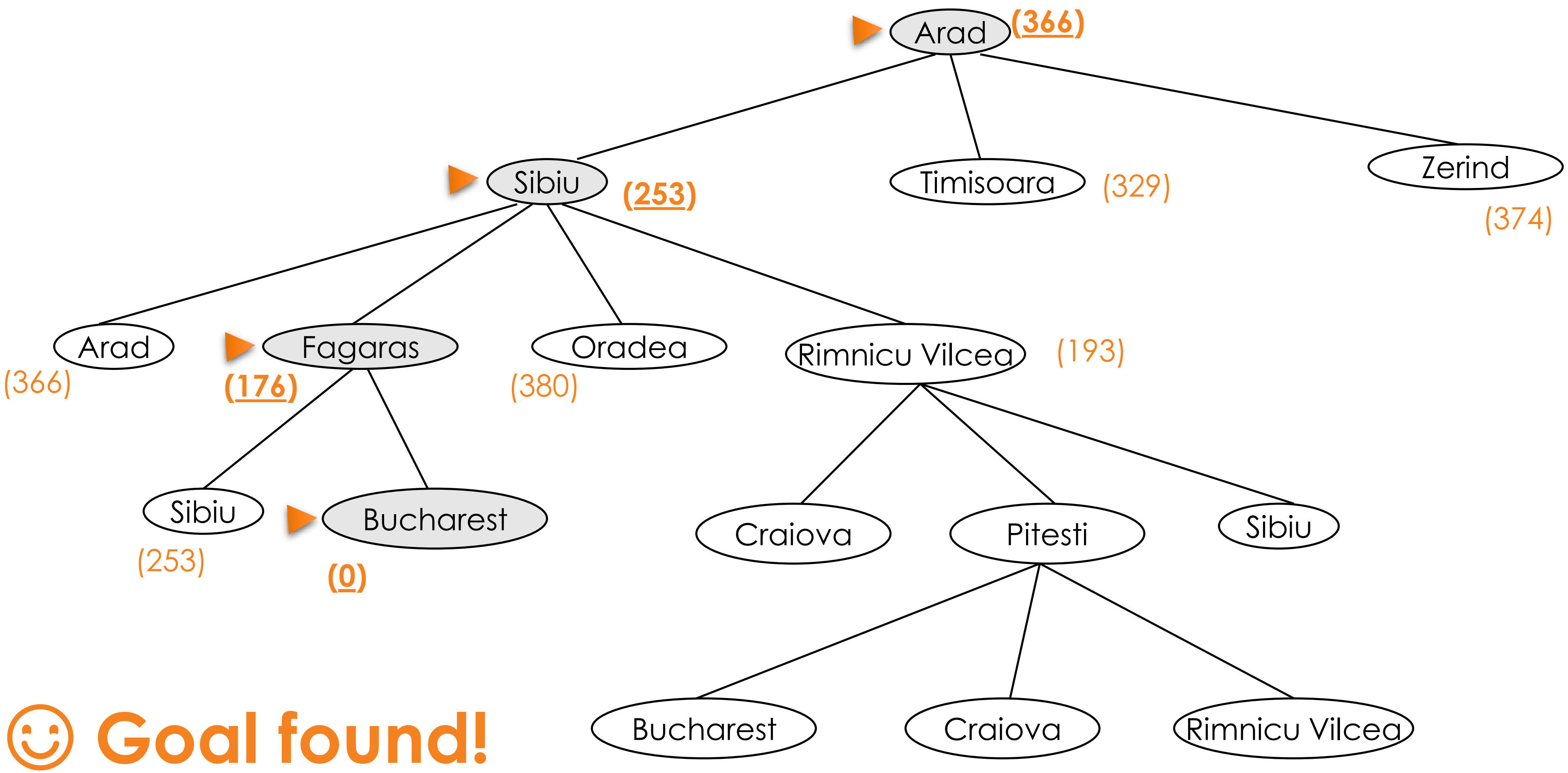


h(n) =Straight-line distance from each city to **Bucharest**

City	Distance	City	Distance
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

1.3 INFORMED SEARCH TECHNIQUES (1/2)

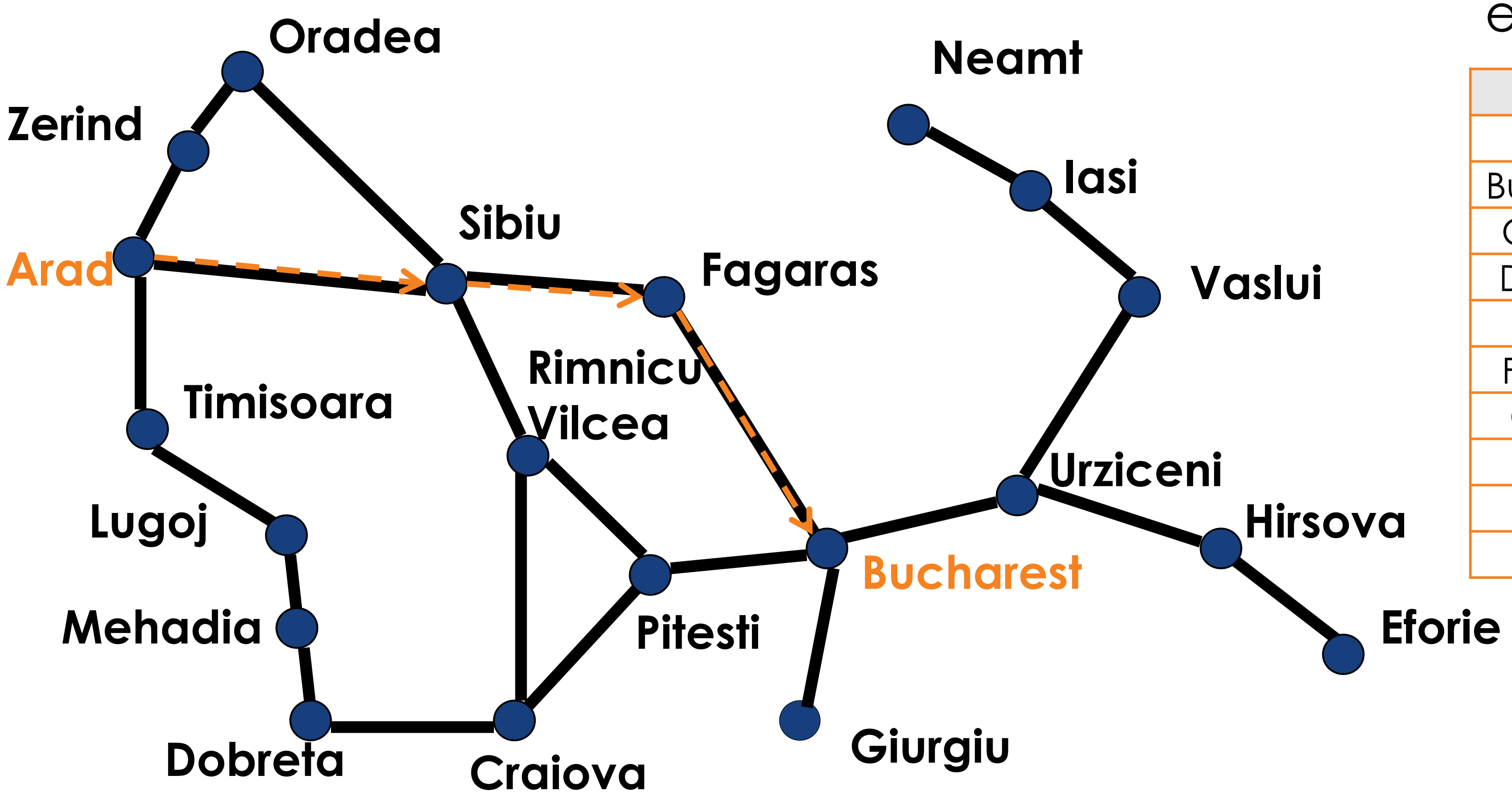
Hill Climbing (Greedy Best First Search)



☺ **Goal found!**

Hill Climbing (Greedy Best First Search)

- **Arad → Bucharest**



h(n) =Straight-line distance from each city to **Bucharest**

City	Distance	City	Distance
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search

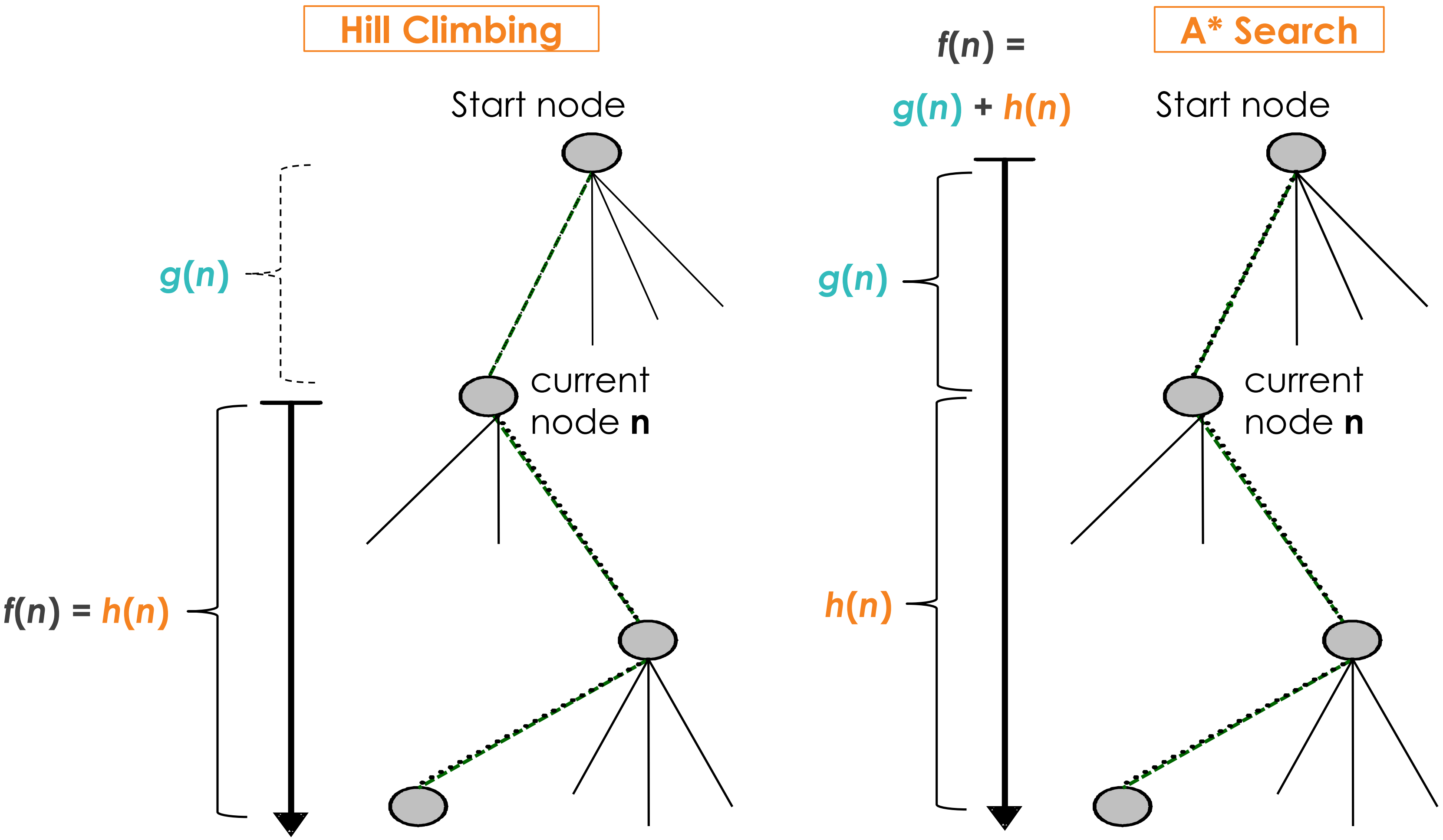
- **A* search is the most widely-known form of best-first search**
  - This strategy evaluates each search node by combining ***g(n)***, the past (path) cost from the start node to current node ***n***, and ***h(n)***, the estimated future (path) cost: the cheapest path/cost from current node ***n*** to a goal node
  - Estimated total cost of the cheapest solution through ***n***

$$f(n) = g(n) + h(n)$$

☺ ***g(n)*** **is exactly known, but *h(n)* is only an estimation with possible error.**

1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search vs. Hill Climbing

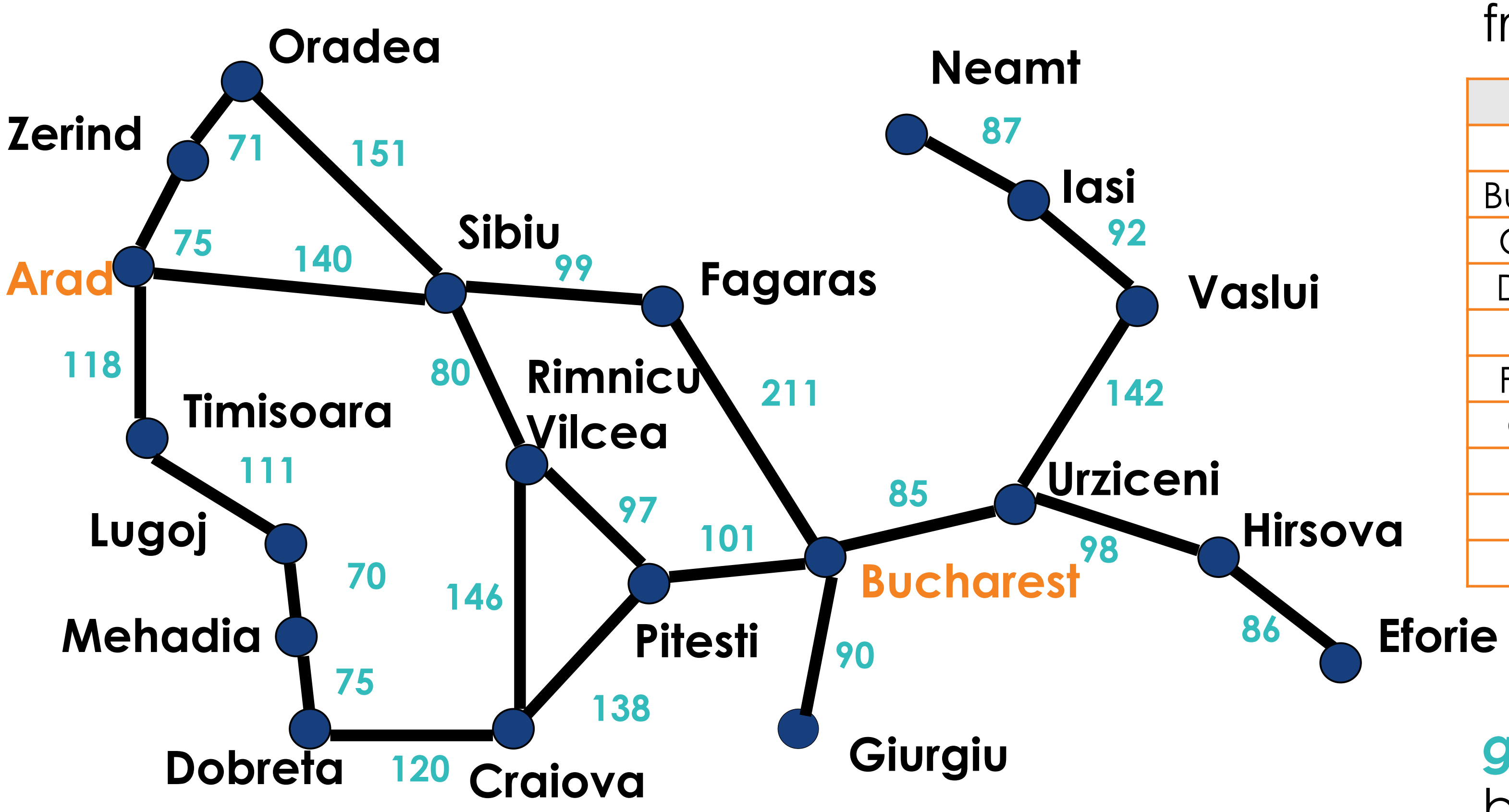


Select then expand "best-path-**from-n**-to-goal" child-node at each layer

Select then expand "best-path-**from-start**-to-goal" child-node at each layer

A* Search

- # Arad → Bucharest



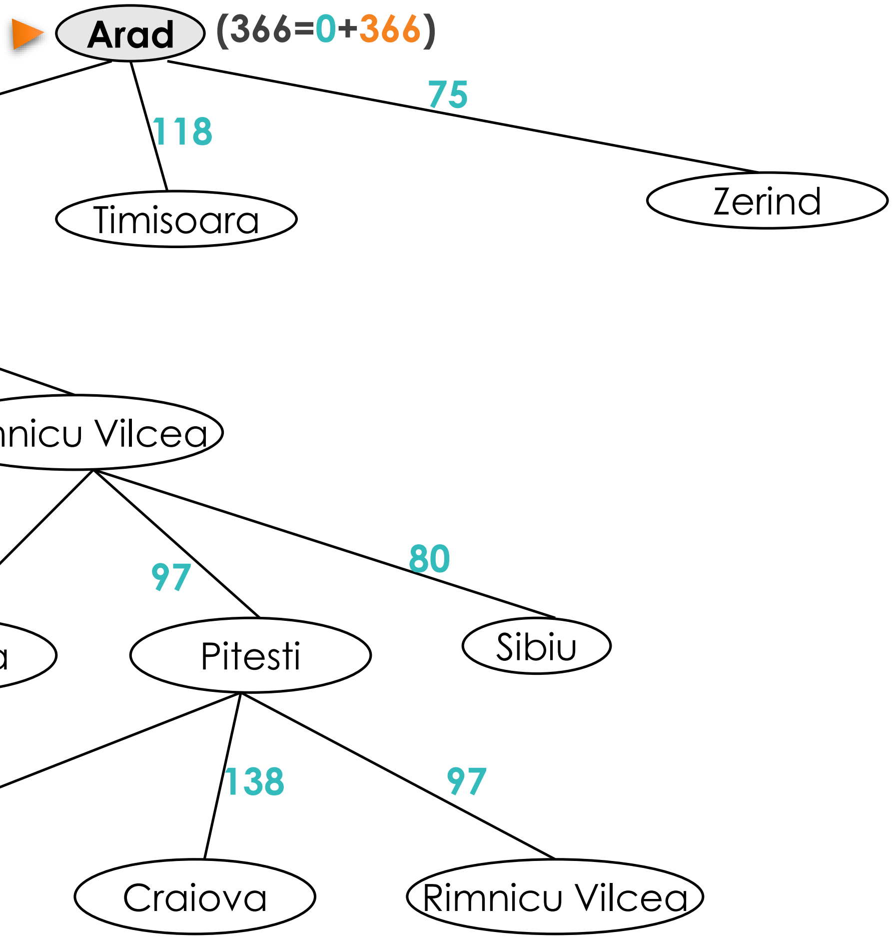
h(n) = Straight-line distance from each city to **Bucharest**

City	Distance	City	Distance
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

g(n) = Actual path distance between different cities

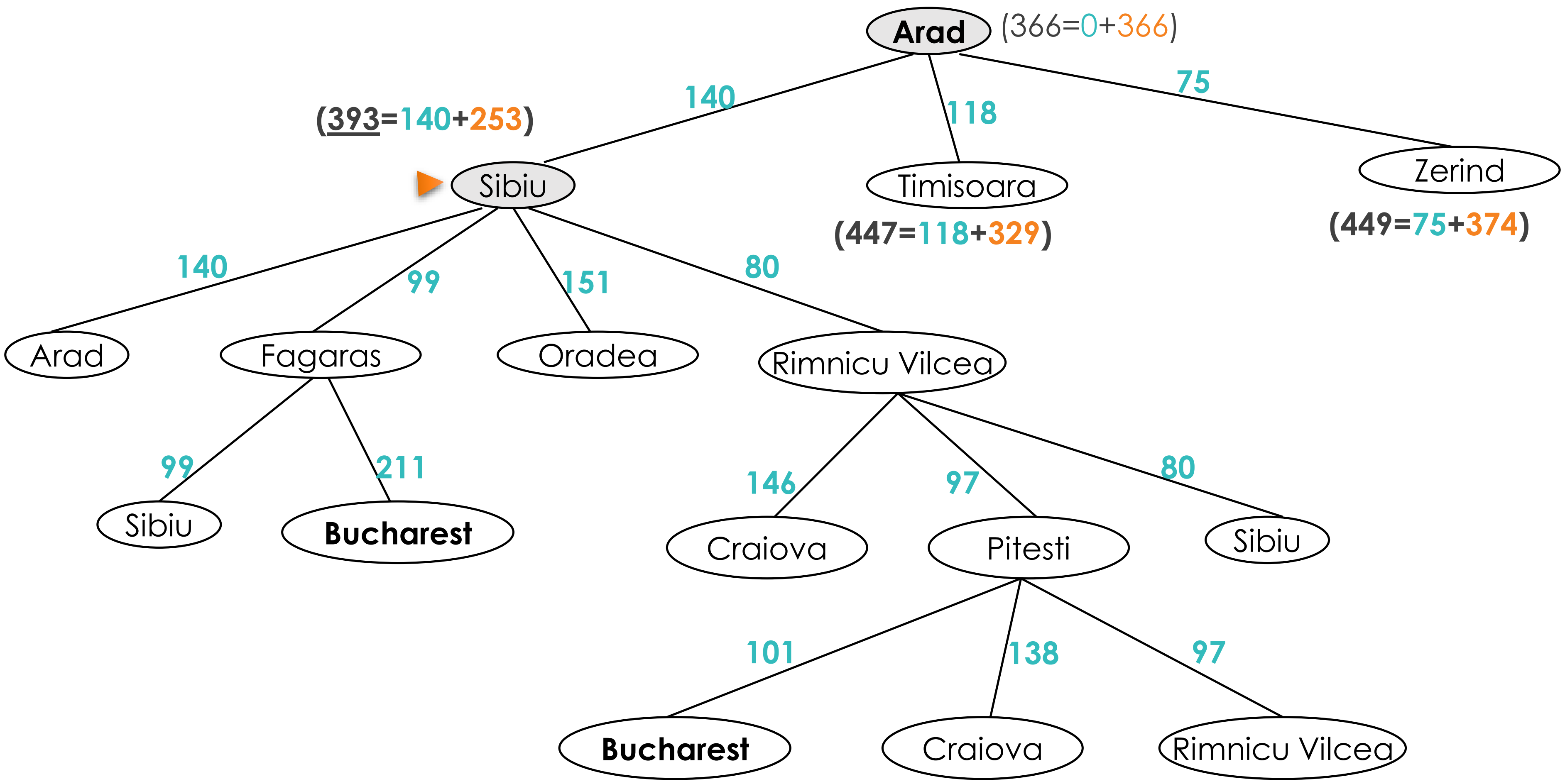
1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search



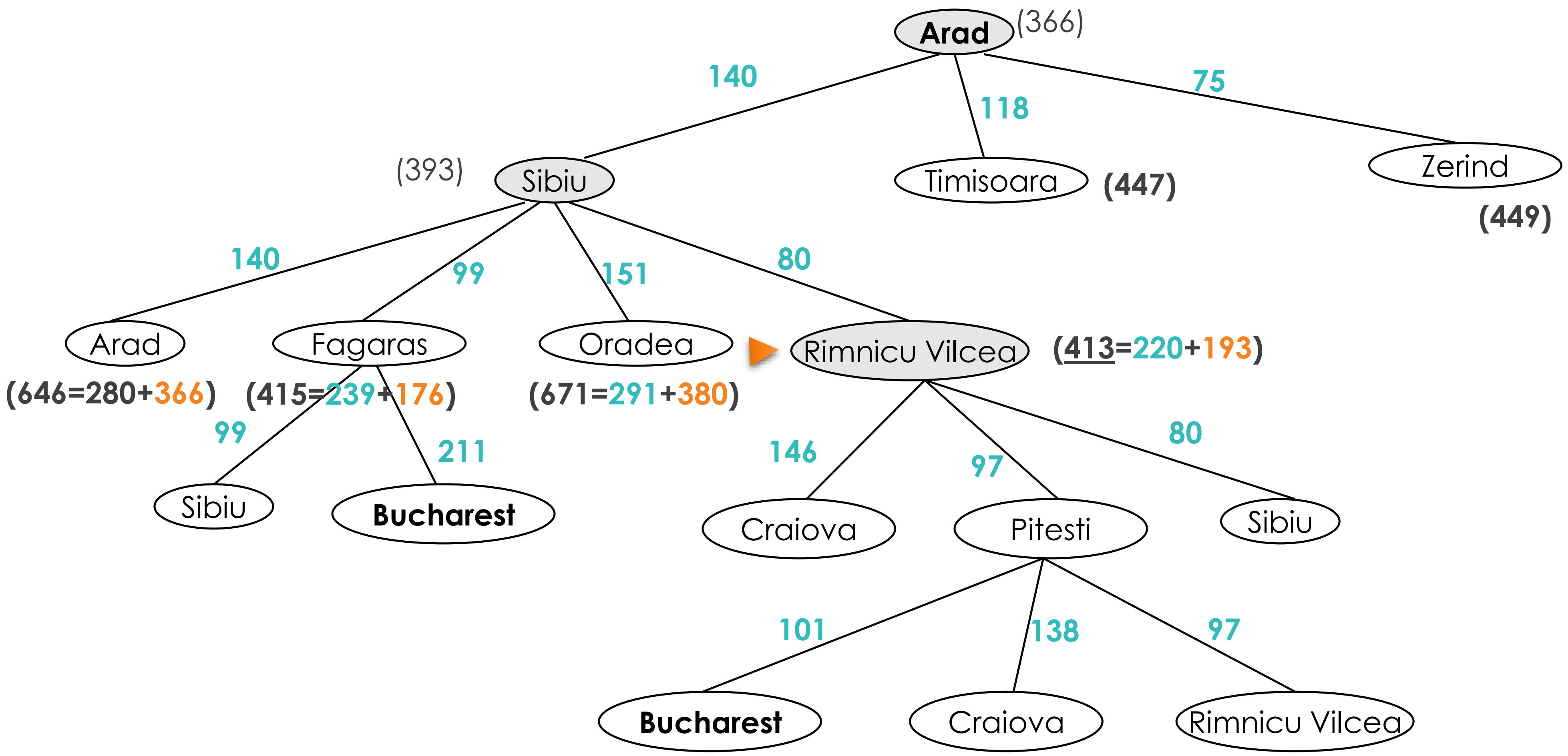
1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search



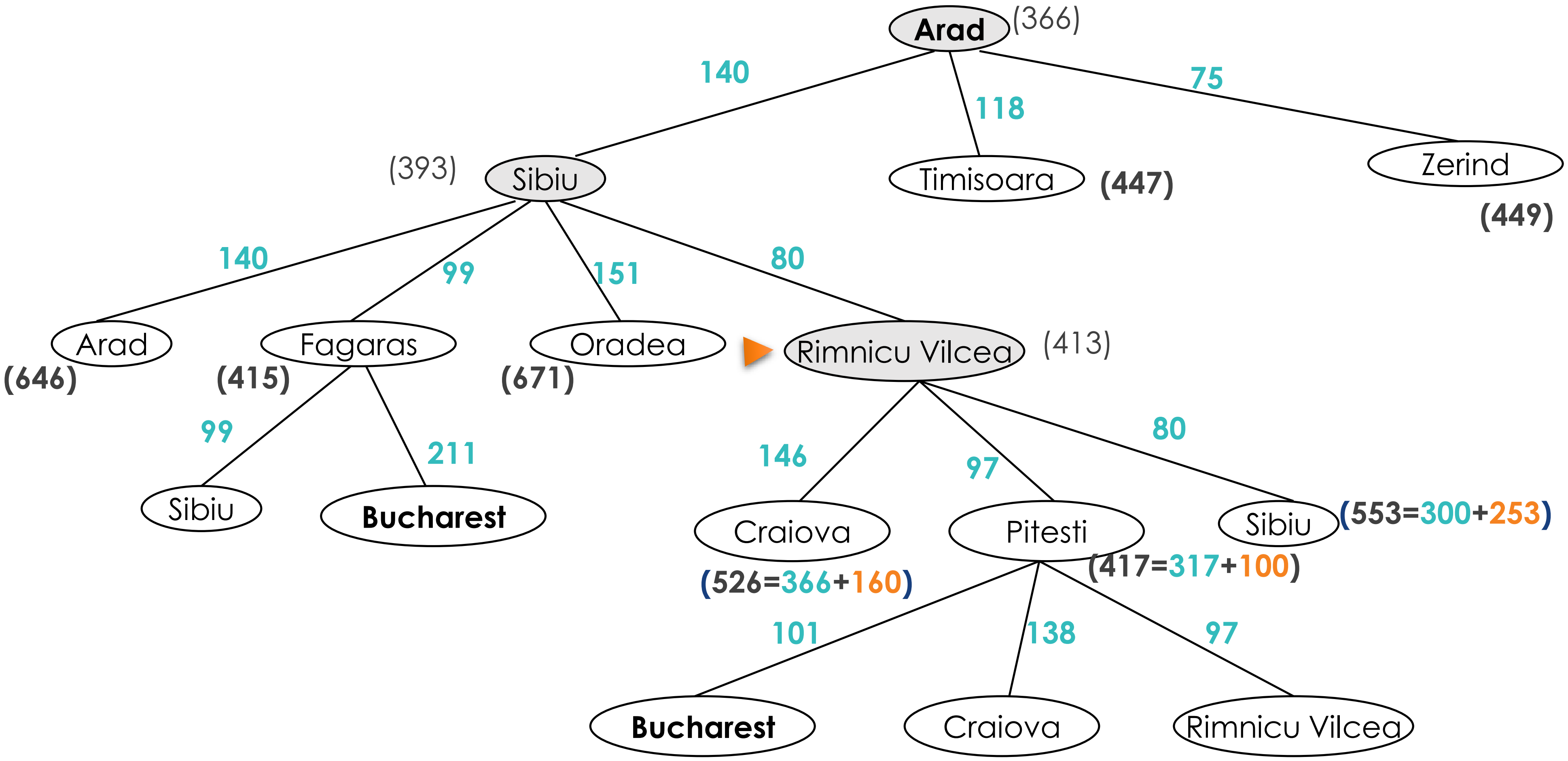
1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search



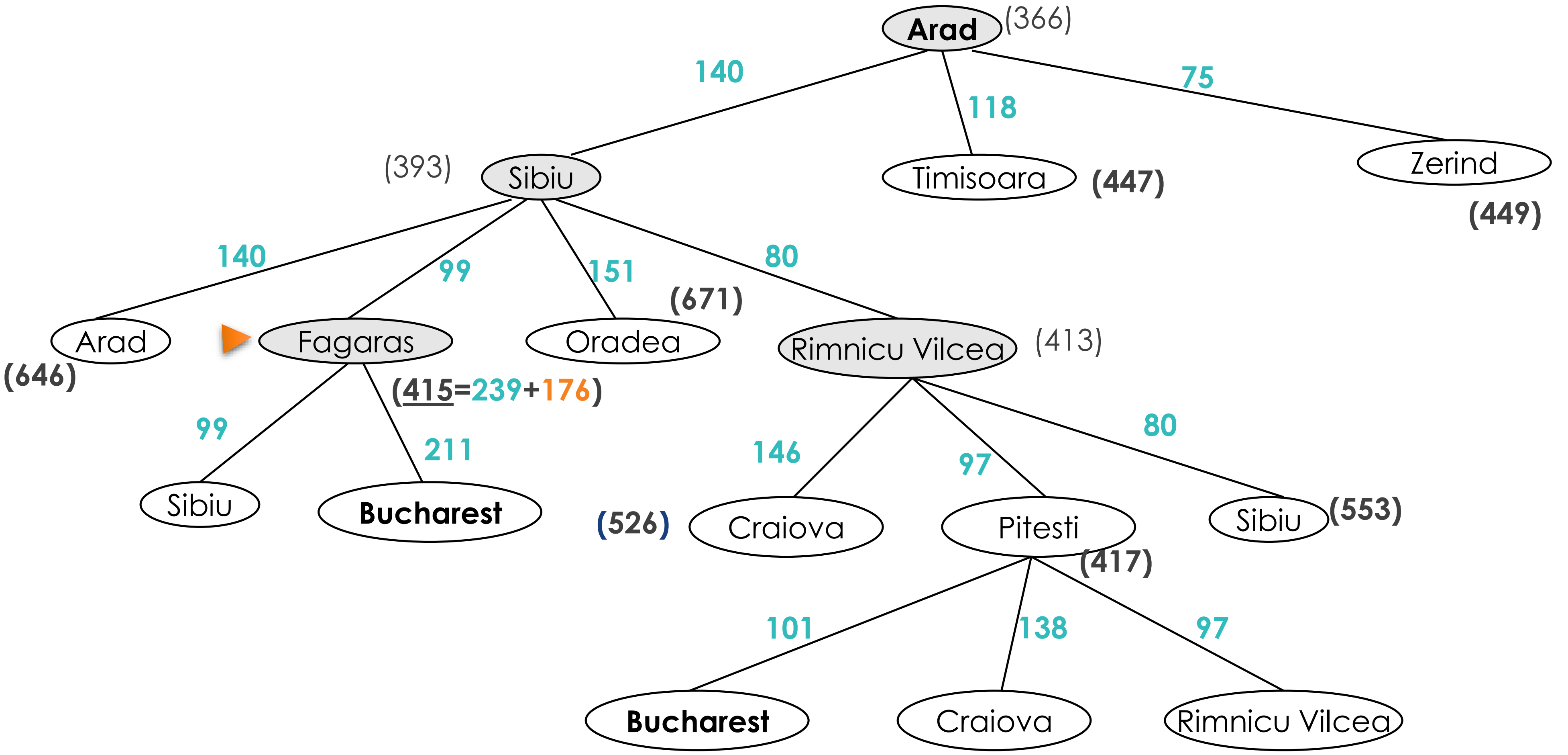
1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search



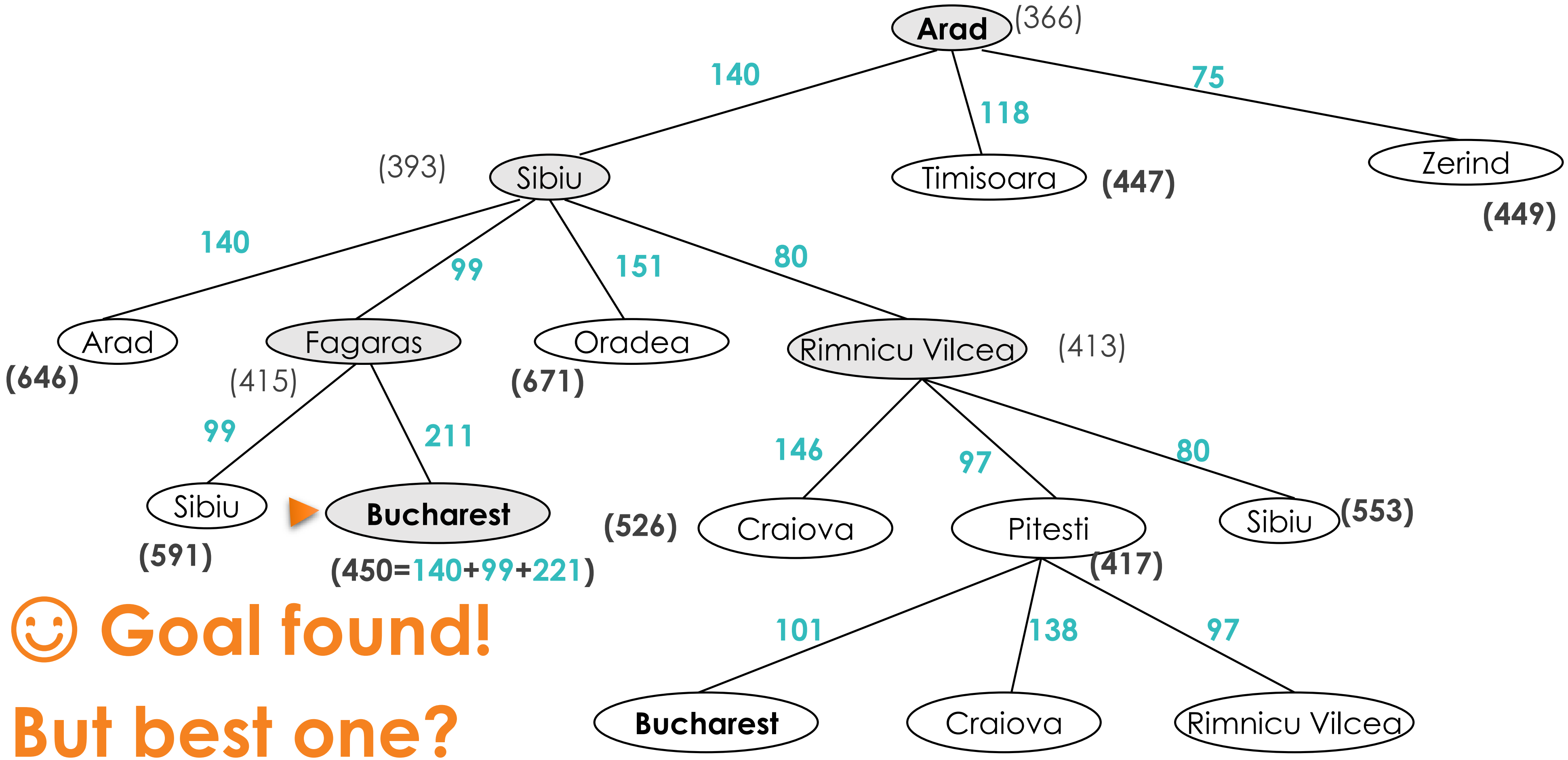
1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search



1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search

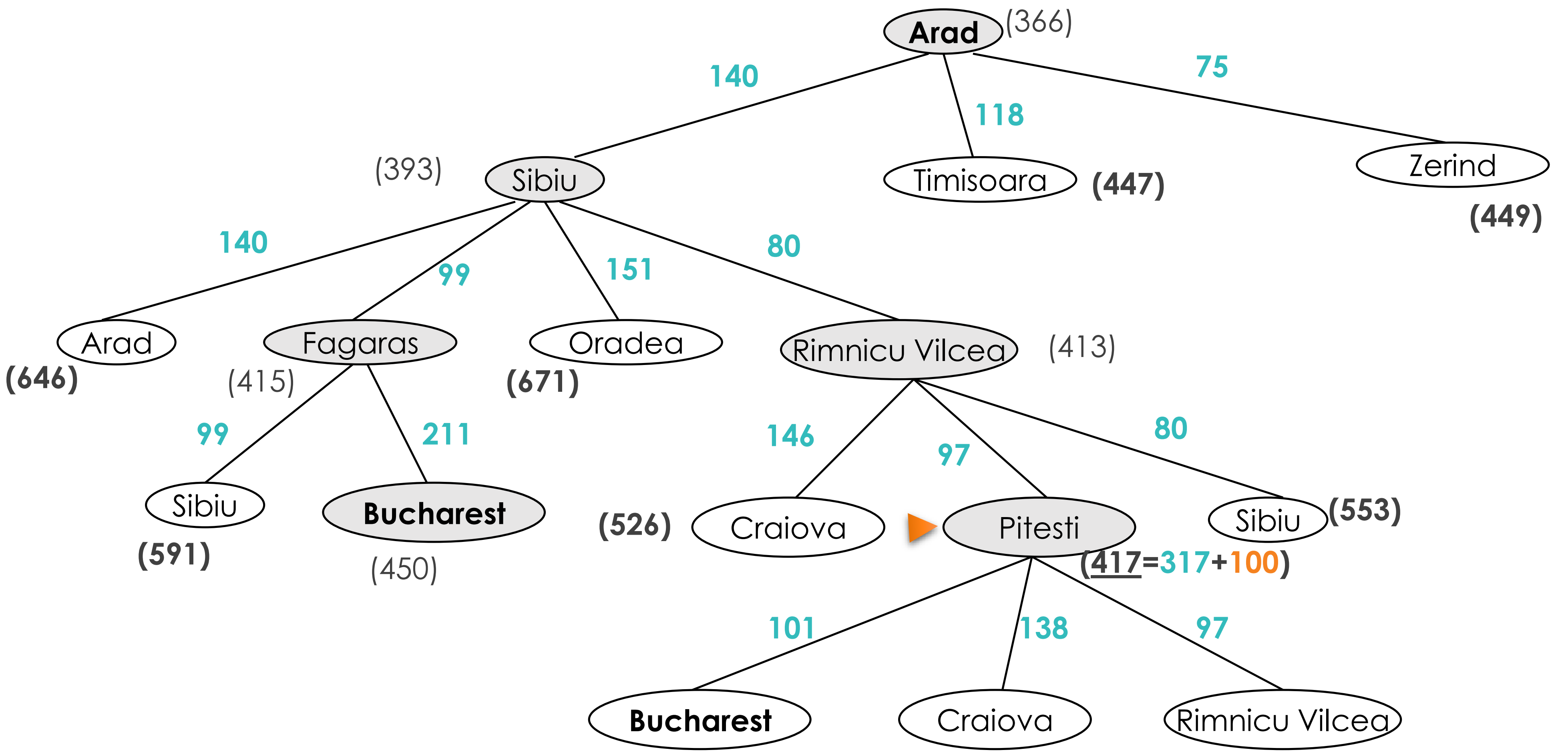


☺ Goal found!

But best one?

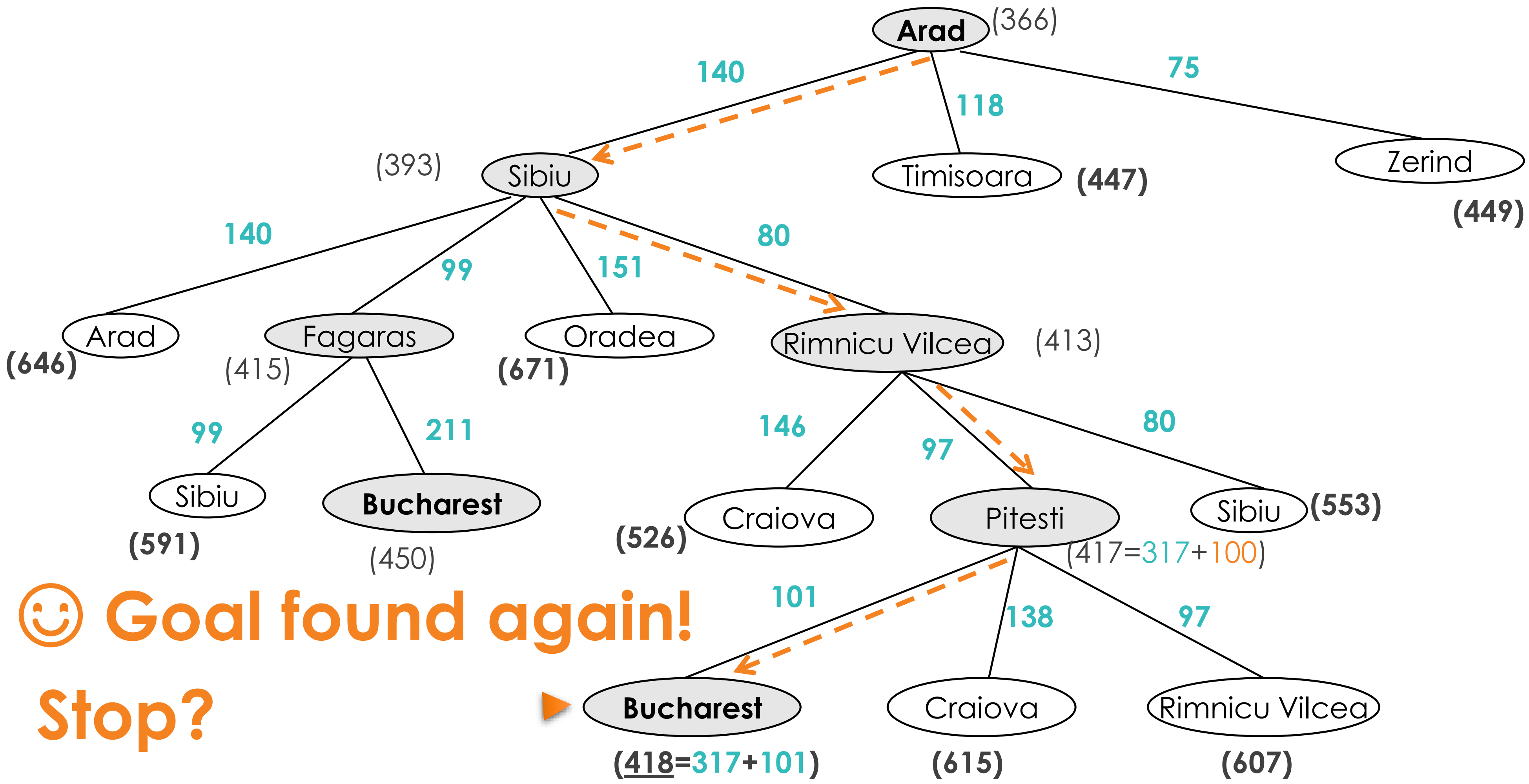
1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search



1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search



☺ Goal found again!

Stop?

1.3 INFORMED SEARCH TECHNIQUES (1/2)

A* Search vs. Hill Climbing

Exercise

- Compare the results between A* Search & Hill Climbing.

- Are they the same? If not, why?

1.4 WORKSHOP

SEARCH REPRESENTATION

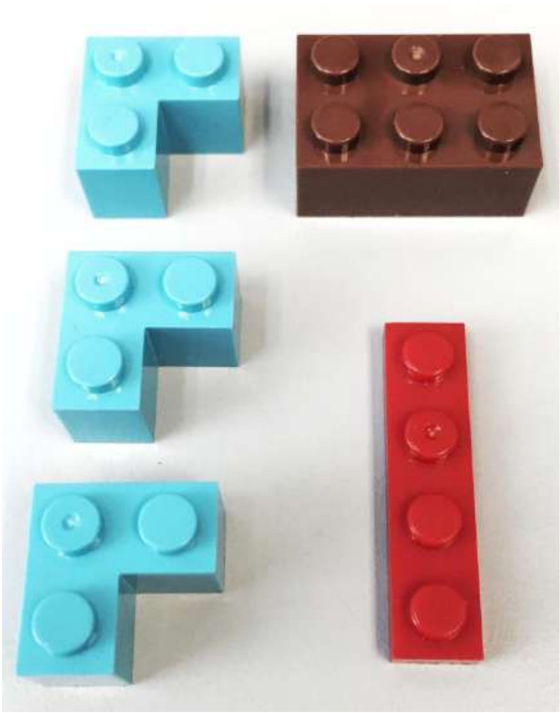
1.4 WORKSHOP SEARCH REPRESENTATION

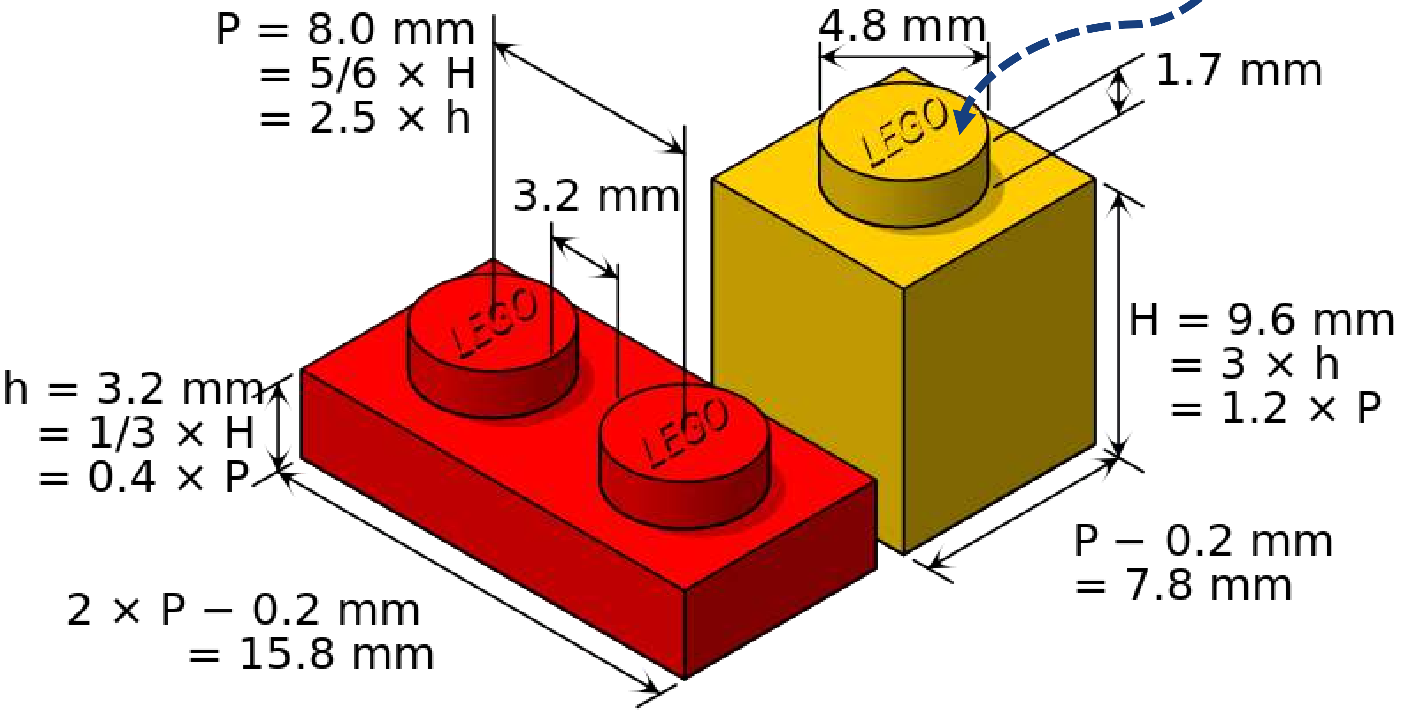
- **Search Modelling & Representation**
  - Pen & Paper Planning
  - Robot Navigation
- **KIE OptaPlanner Tutorial**
  - Optimizing Vehicle Route Planning (VRP)
  - Optimizing Europe Travelling Sales Person (TSP)

1.4 WORKSHOP SEARCH REPRESENTATION

Search Modelling & Representation

- Goal: Find a configuration with minimal total studs: little round bumps.
- Enhanced Goal: Lower block height without increasing total studs.





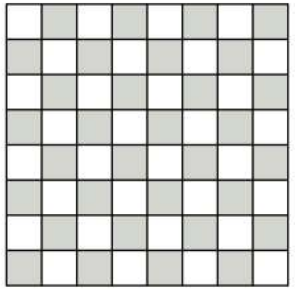
https://www.quora.com/Are-the-dimensions-of-LEGO-bricks-functional-for-recreating-a-Minecraft-environment

1.4 WORKSHOP SEARCH REPRESENTATION

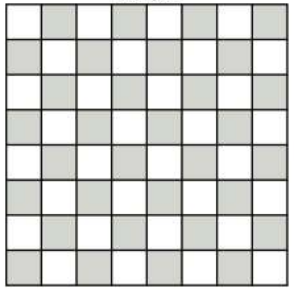
Search Modelling & Representation

- **Pen & Paper Planning**

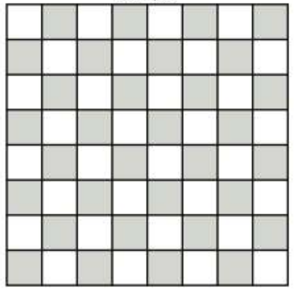
1) Place 8 queens on this chessboard so no 2 queens can attack each other.

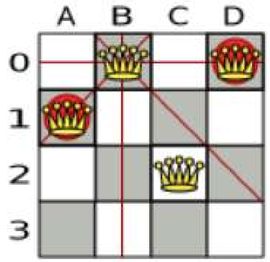


attempt 2

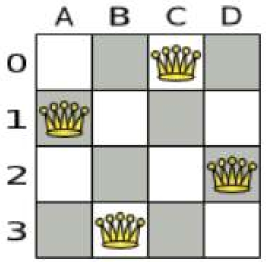


attempt 3



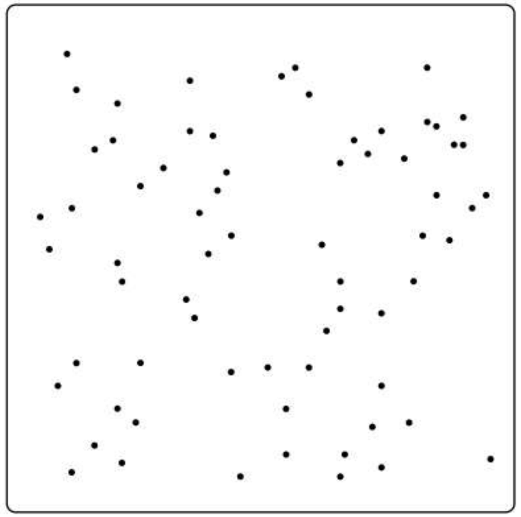


Bad

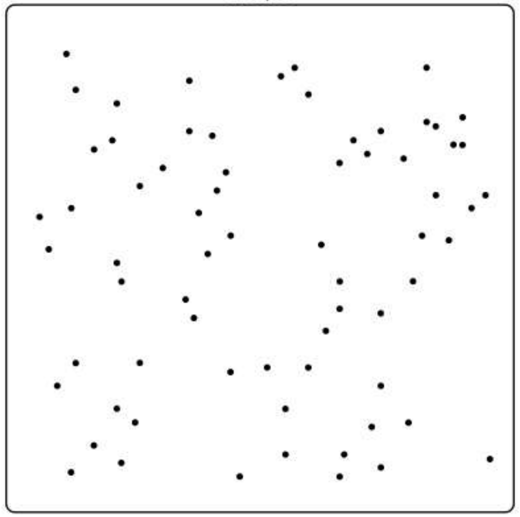


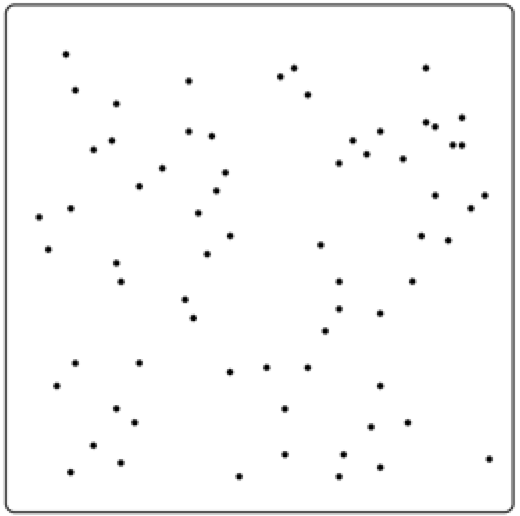
Good

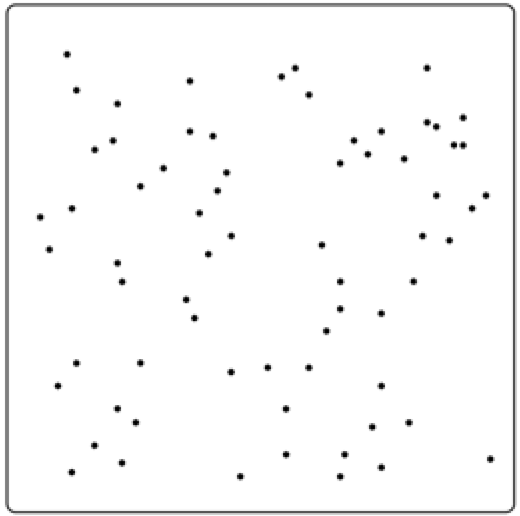
2) Draw the shortest line that connects all dots and returns to its origin.



attempt 2







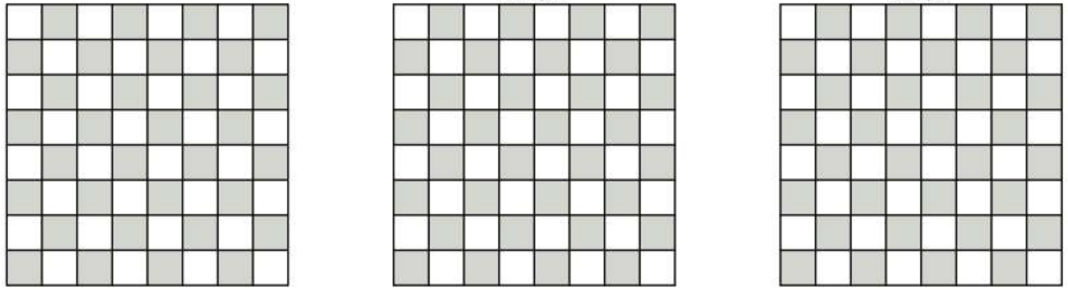
1.4 WORKSHOP SEARCH REPRESENTATION

Search Modelling & Representation

- **Pen & Paper Planning**

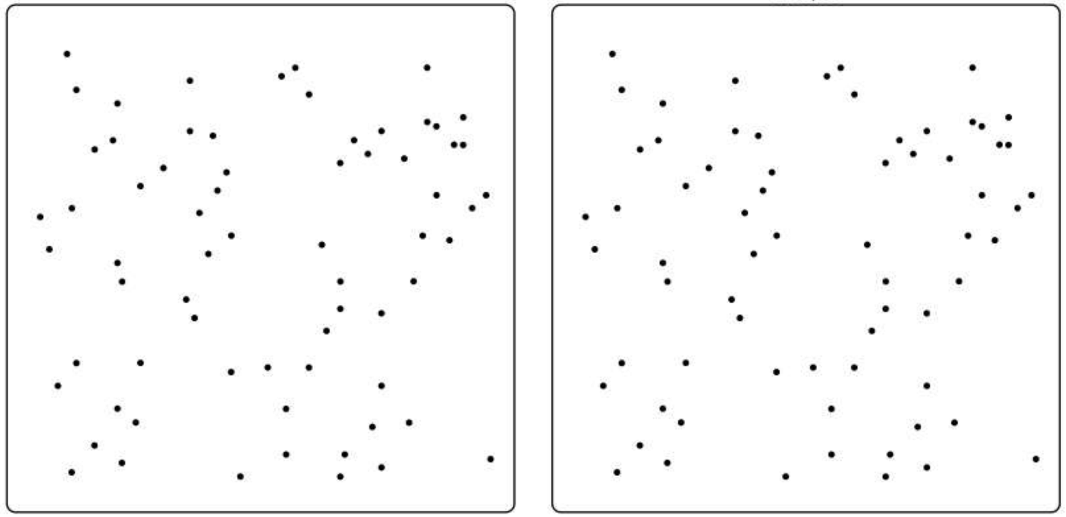
1) Place 8 queens on this chessboard so no 2 queens can attack each other.

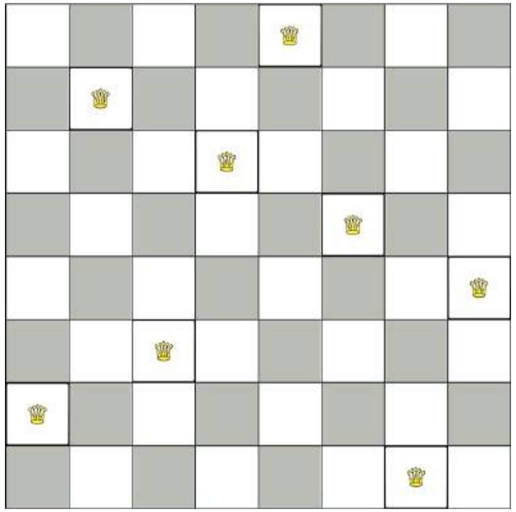
attempt 2 attempt 3

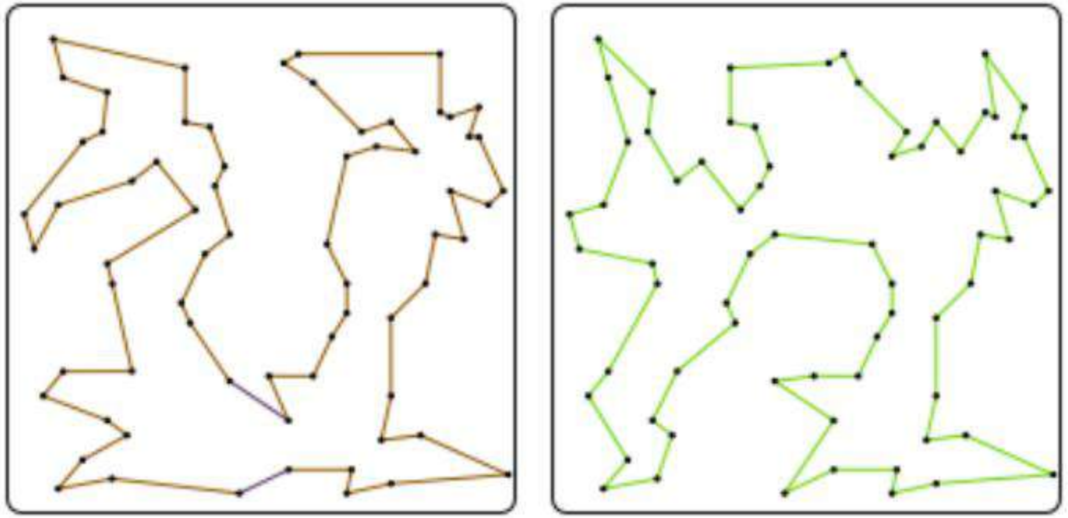


2) Draw the shortest line that connects all dots and returns to its origin.

attempt 2

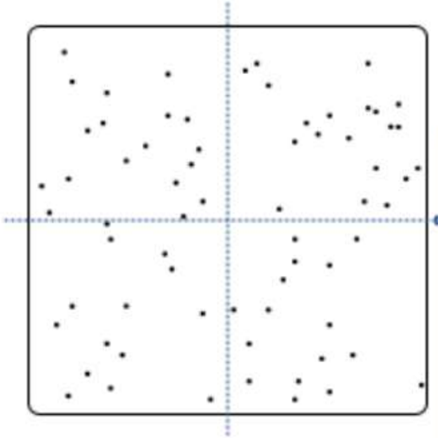






MapReduce is terrible for TSP

Why do MapReduce, Divide&Conquer and partitioning perform badly on NP-hard problems?

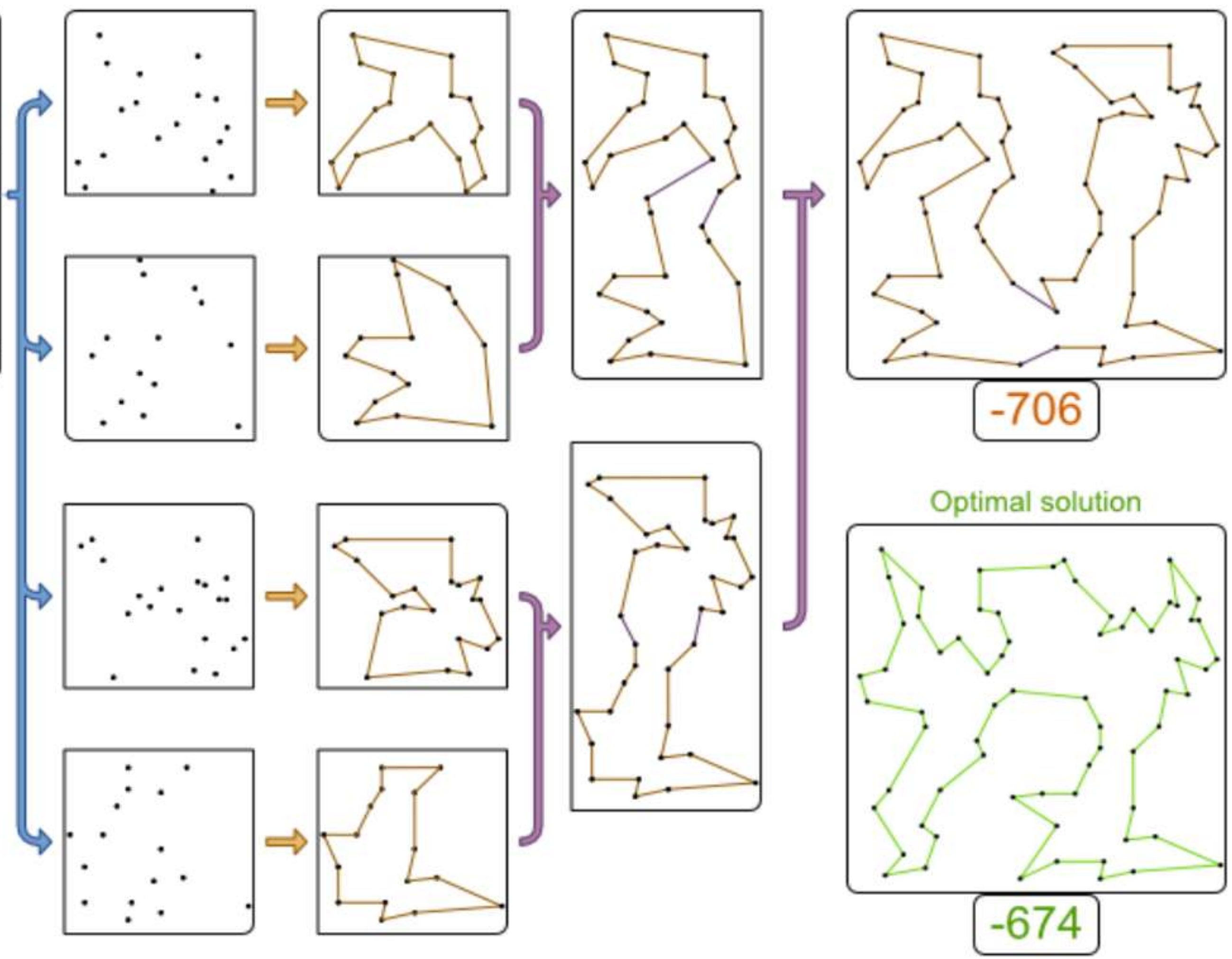


1) Partition
Divide into n pieces

2) Map
Solve each piece

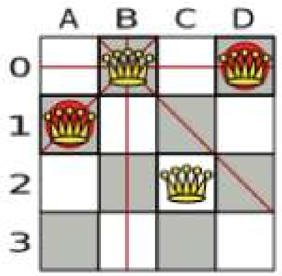
3) Reduce
Merge piece solutions
(not always possible)

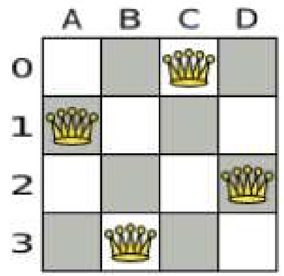
4) Result
Not (near) optimal
Worse when scaling



N Queens

- Hard constraints:
  - -1 for every pair of conflicting queens
- Soft constraints:
  - None



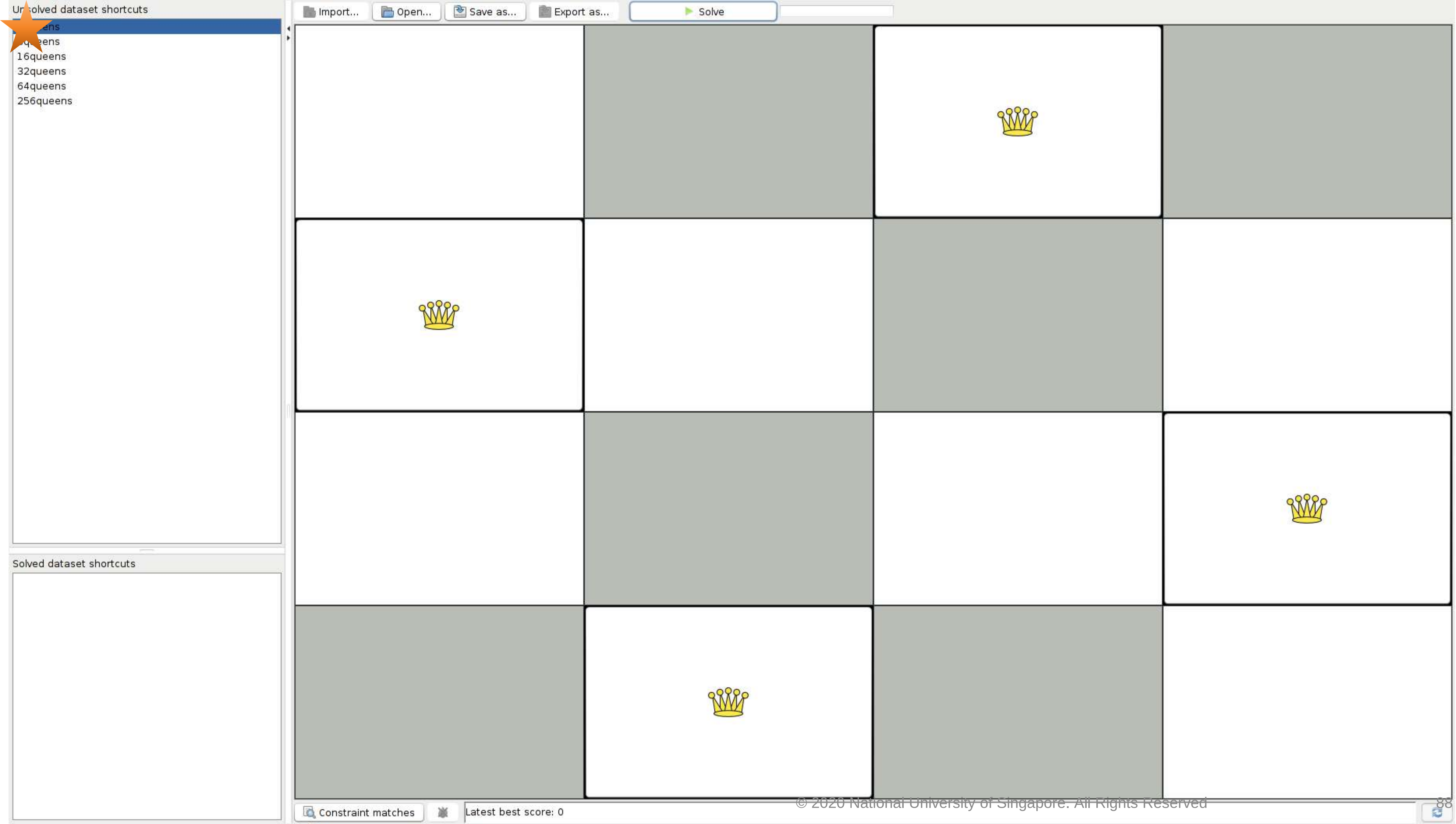


Score = -2

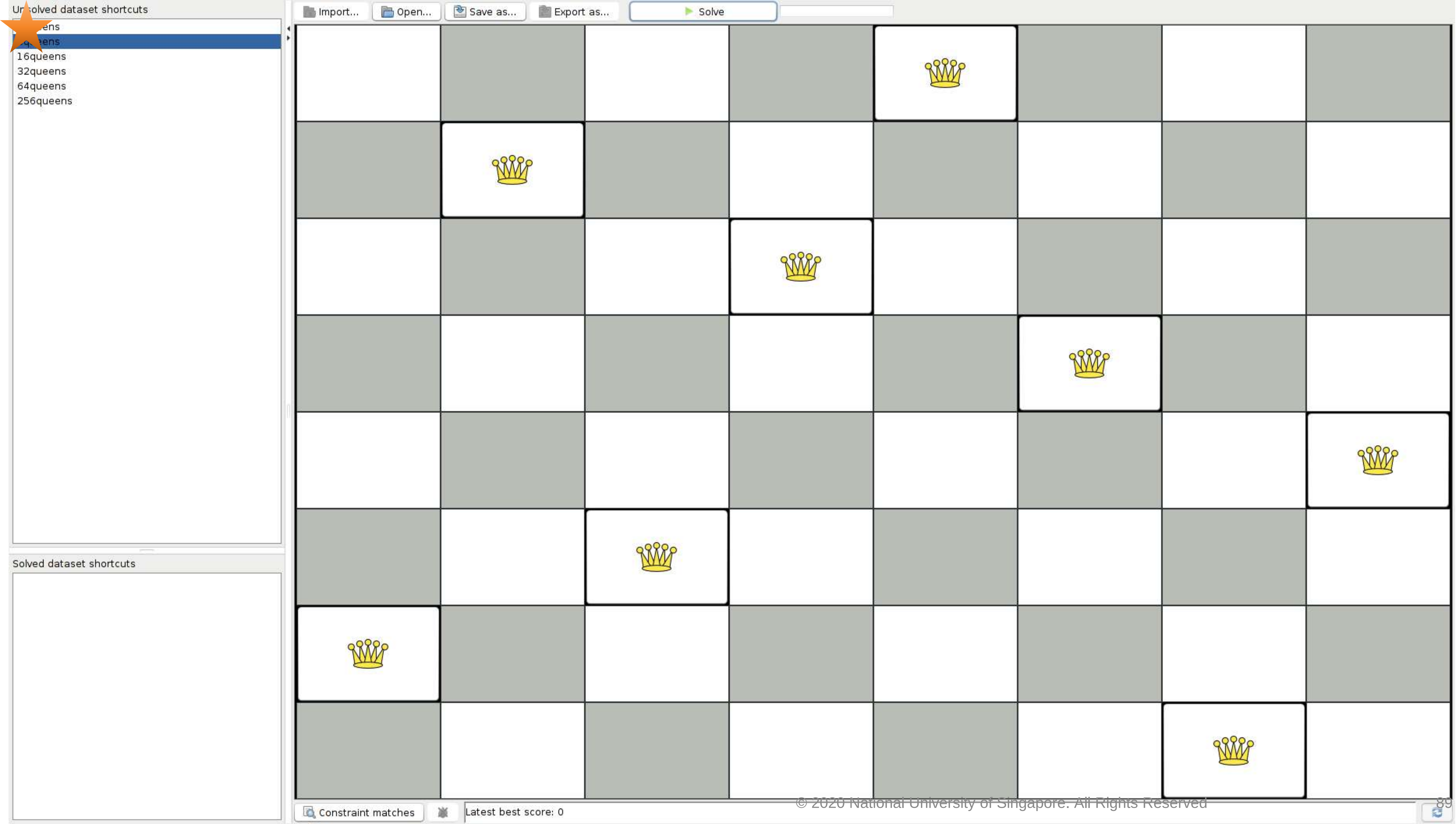
Conflicts: A-B, B-D

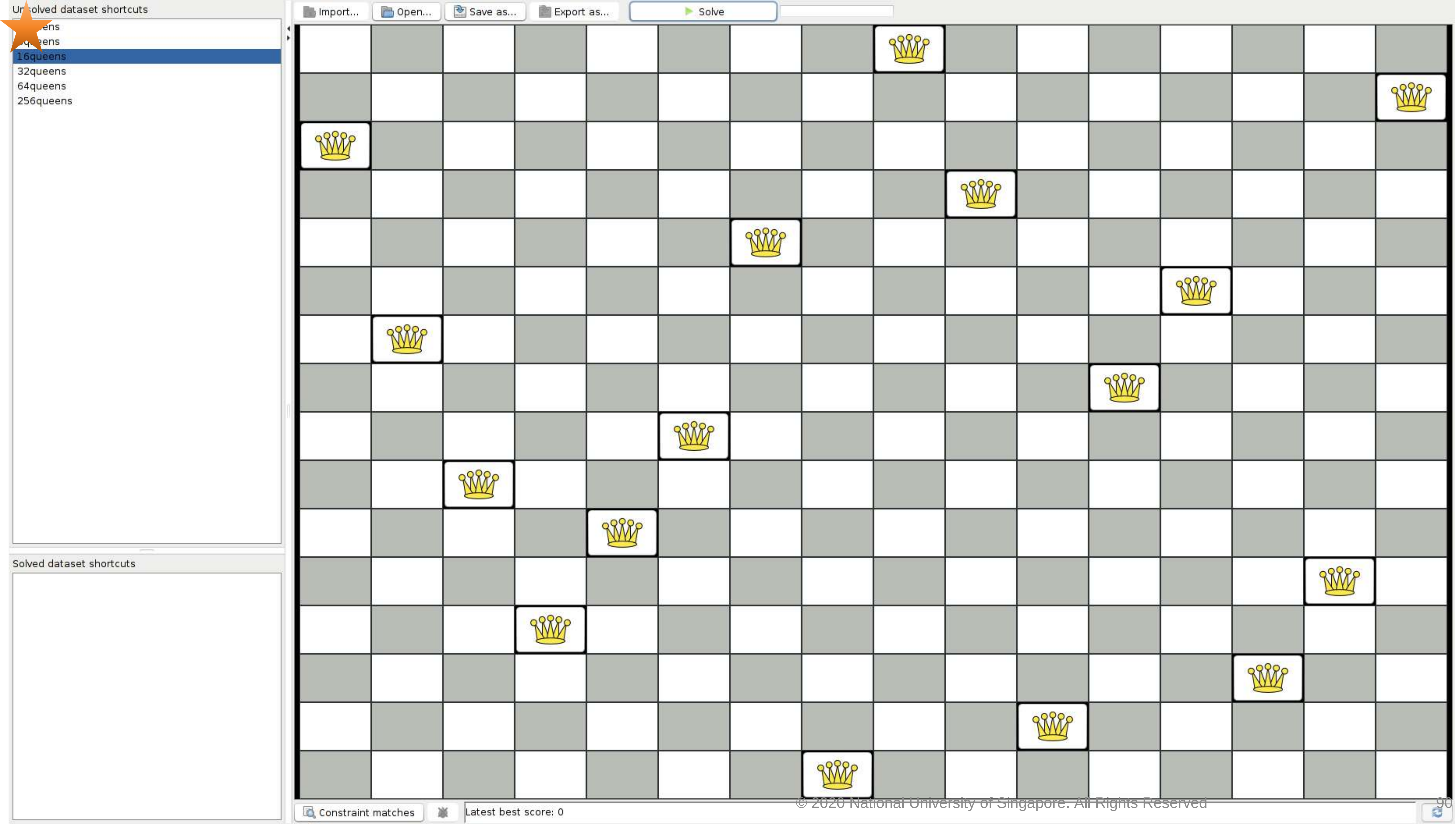
Score = 0

No conflicts



Import...
Open...
Save as...
Export as...
Solve
16queens
32queens
64queens
256queens
Solved dataset shortcuts
Constraint matches
Latest best score: 0


16queens
32queens
64queens
256queens
Solved dataset shortcuts
Import...
Open...
Save as...
Export as...
Solve
Constraint matches
Latest best score: 0

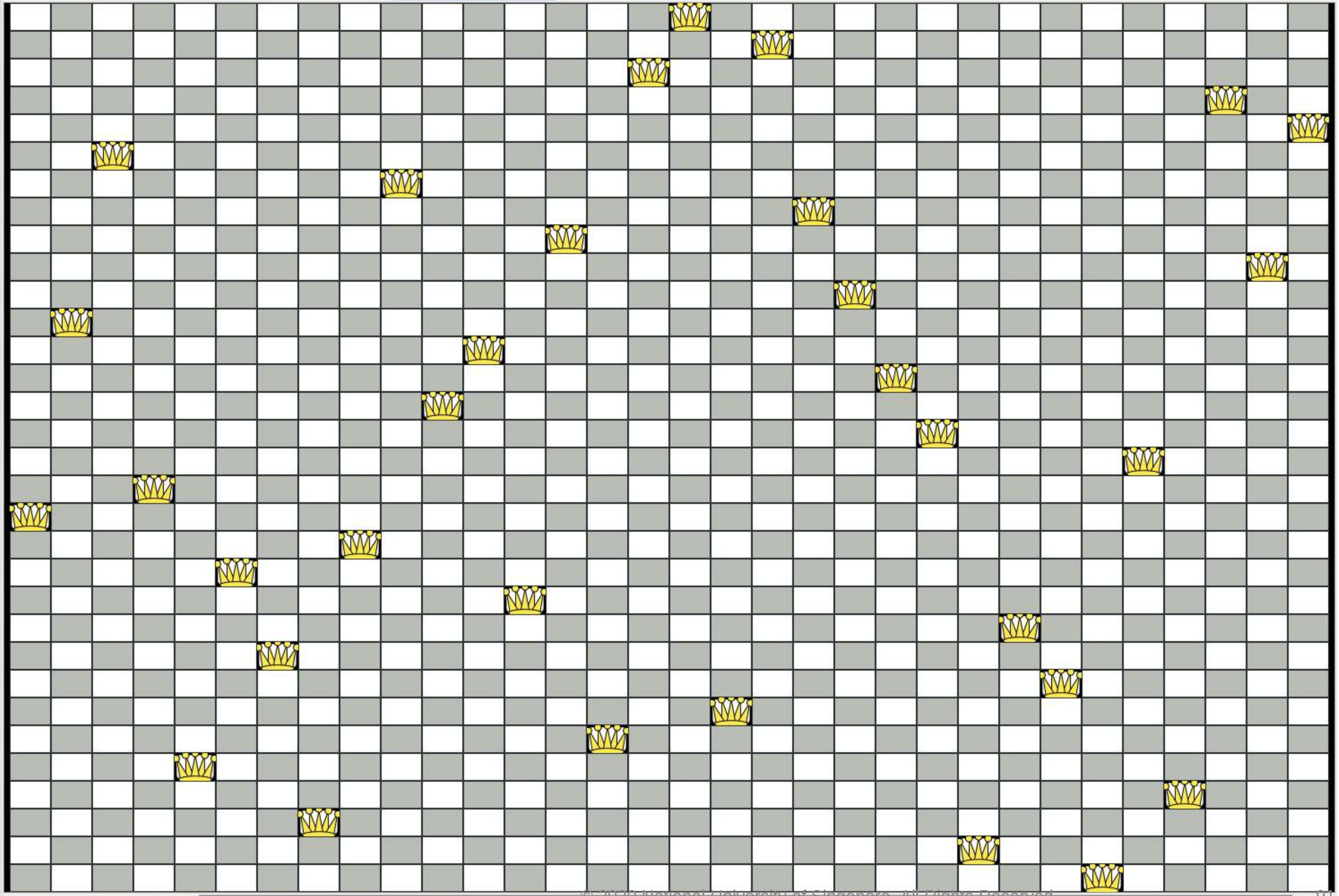

Unsolved dataset shortcuts
16queens
32queens
64queens
256queens
Import...
Open...
Save as...
Export as...
Solve
Solved dataset shortcuts
Constraint matches
Latest best score: 0

Unsolved dataset shortcuts

- ens
- queens
- 16queens
- 32queens
- 64queens
- 256queens

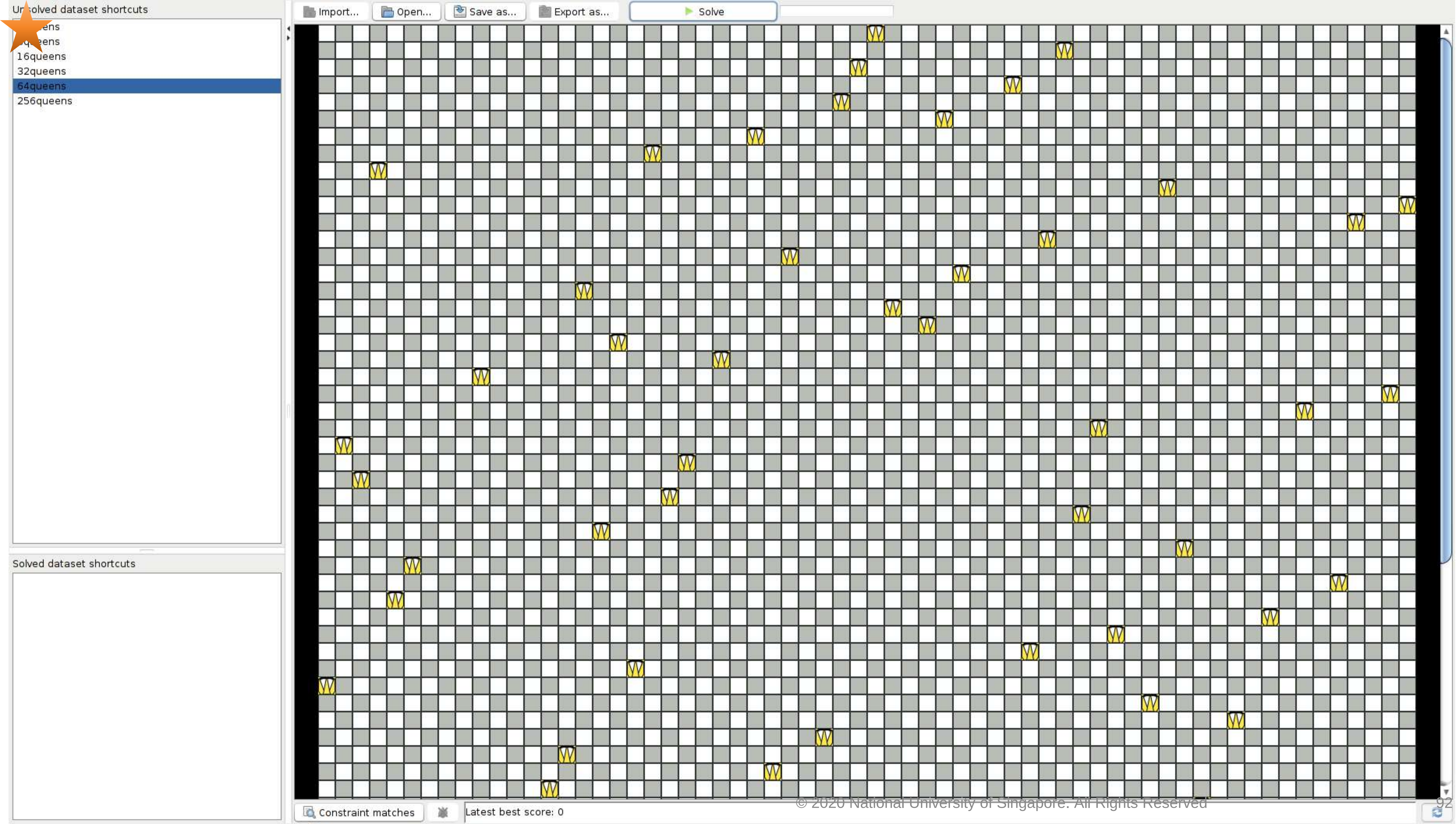
Solved dataset shortcuts

Import... Open... Save as... Export as... Solve

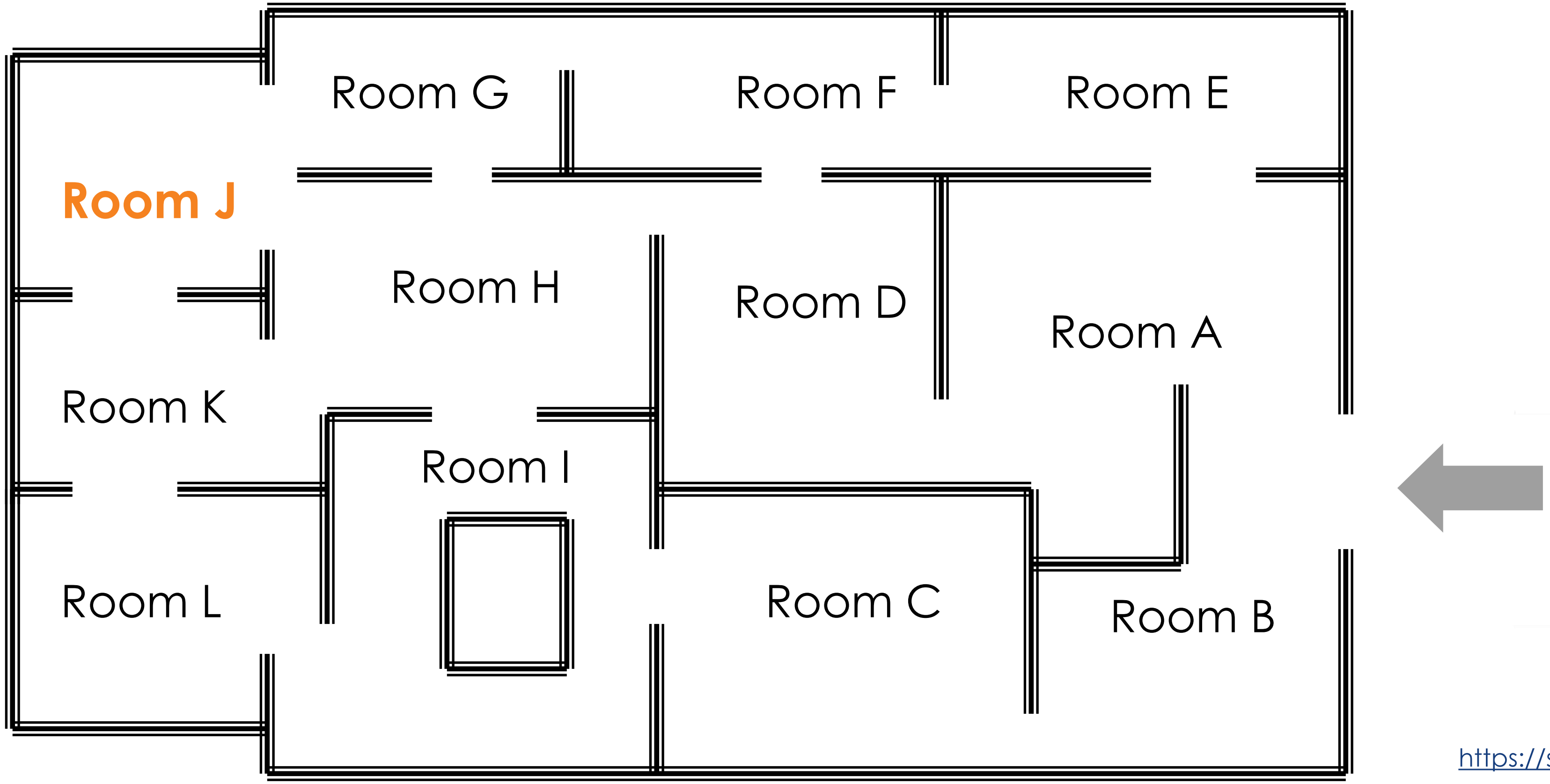


Constraint matches

Latest best score: 0


Unsolved dataset shortcuts
16queens
32queens
64queens
256queens
Solved dataset shortcuts
Import...
Open...
Save as...
Export as...
Solve
Constraint matches
Latest best score: 0

- Robotics: How to rapidly navigate to Room J ?

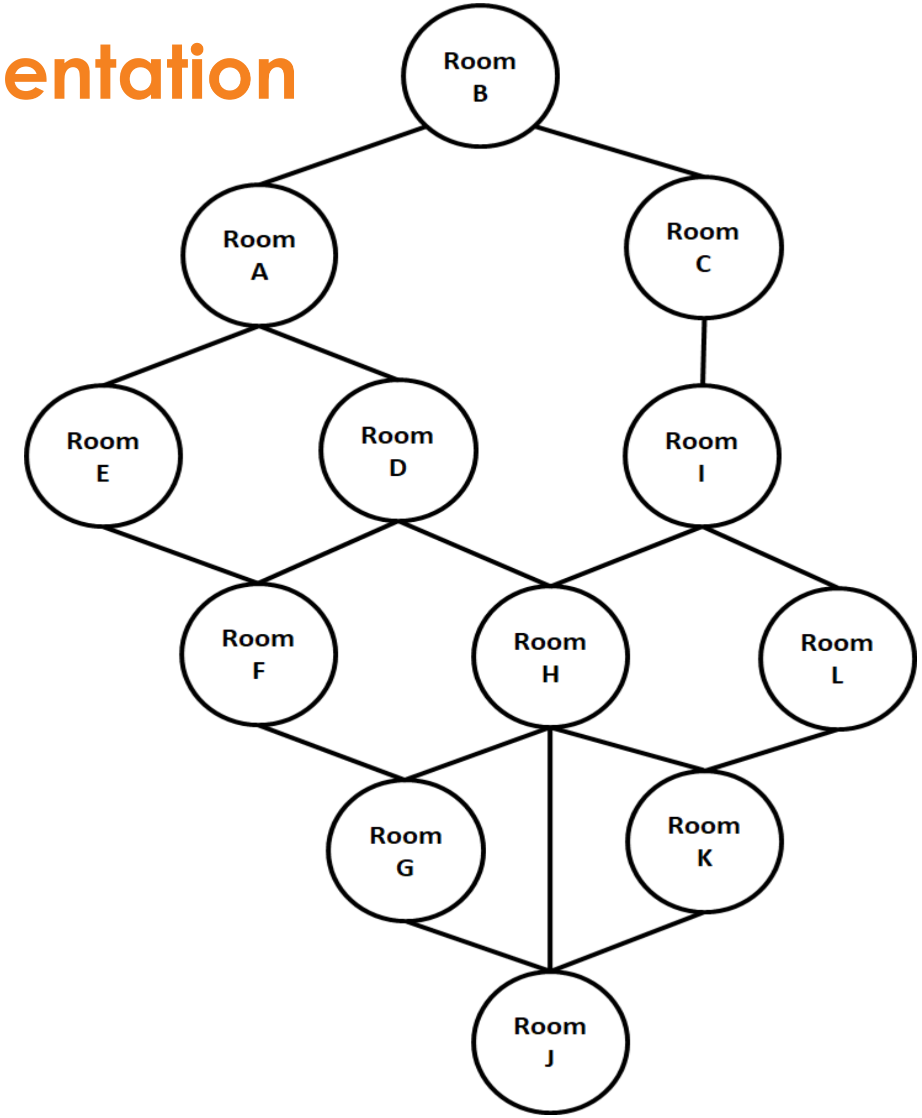


https://static1.squarespace.com/static/57c8a68a20099ef23fb19e90/t/5a32a9804192022be97f9d10/1513269668629/Atlas.png

1.4 WORKSHOP SEARCH REPRESENTATION

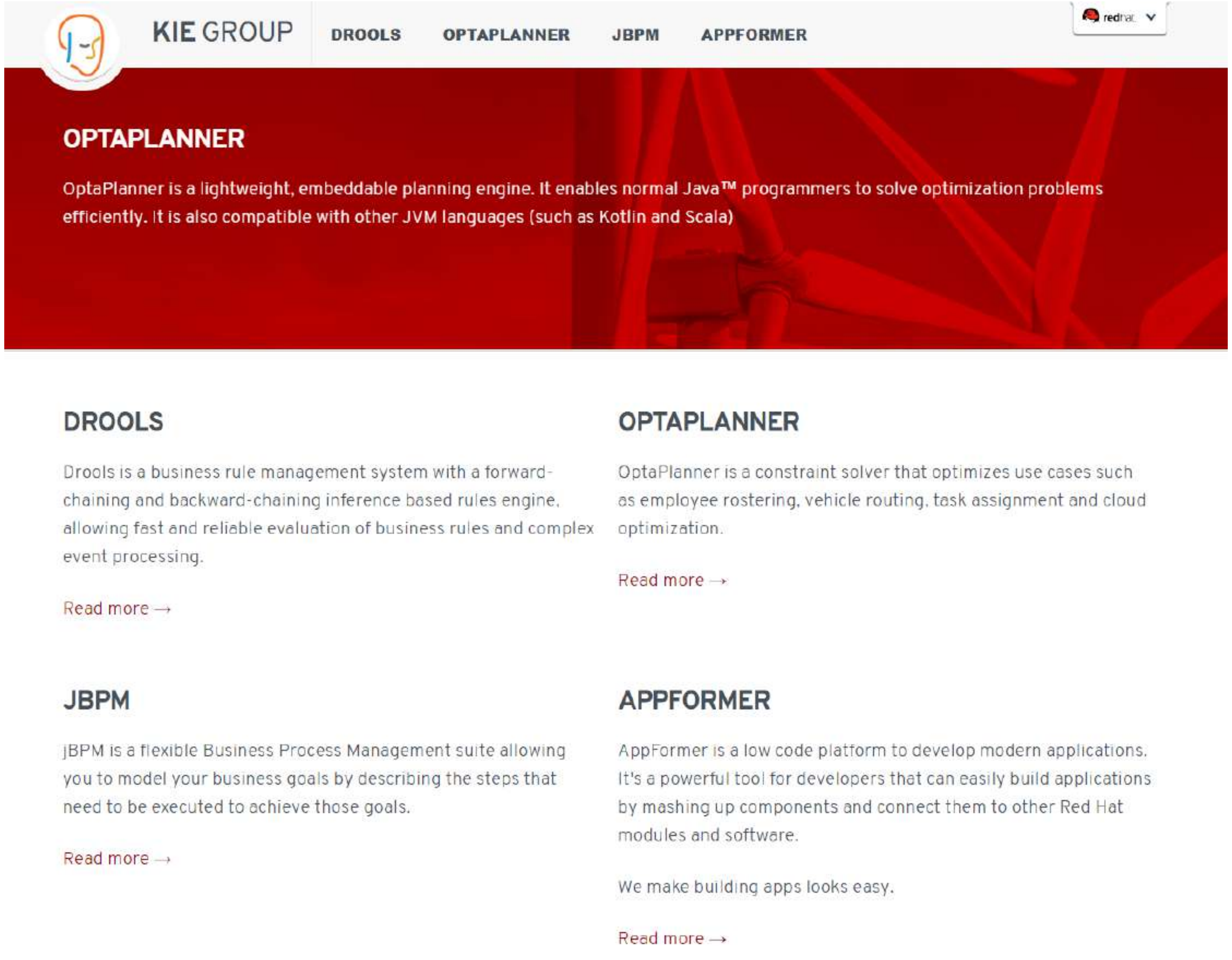
Search Modelling & Representation

Search Representation



1.4 WORKSHOP SEARCH REPRESENTATION

KIE OptaPlanner Tutorial



JBoss KIE

http://www.kiegroup.org/

JBoss KIE OptaPlanner

http://www.optaplanner.org/

Search

Training Courses ▾ Bootcamps ▾ Career ▾ About Us ▾ Contact Us

OptaPlanner in Practice Training Course

Course Code

optaprac

Duration

21 hours (usually 3 days including breaks)

Overview

This course uses a practical approach to teaching OptaPlanner. It provides participants with the tools needed to perform the basic functions of this tool.

Course Outline

Planner introduction

- What is OptaPlanner?
- What is a planning problem?
- Use Cases and examples

Bin Packaging Problem Example

- Problem statement
- Problem size
- Domain model diagram
- Main method
- Solver configuration
- Domain model implementation
- Score configuration

Travelling Salesman Problem (TSP)

- Problem statement

Testimonials

★★★★★

Knowledge of Tableau was "built up" in a solid way, it was clear that the trainer knew how best to introduce newbies to Tableau, this made it seem very easy..

Siemens Gamesa c/o Hemsley Fraser

Course:Tableau Fundamentals

Trainer went away and found out answers to the questions we had that he didn't know

Bookings, Prices and Enquiries

Guaranteed to run even with a single delegate!

Private Classroom

From 11381SGD

Request quote

Private Remote

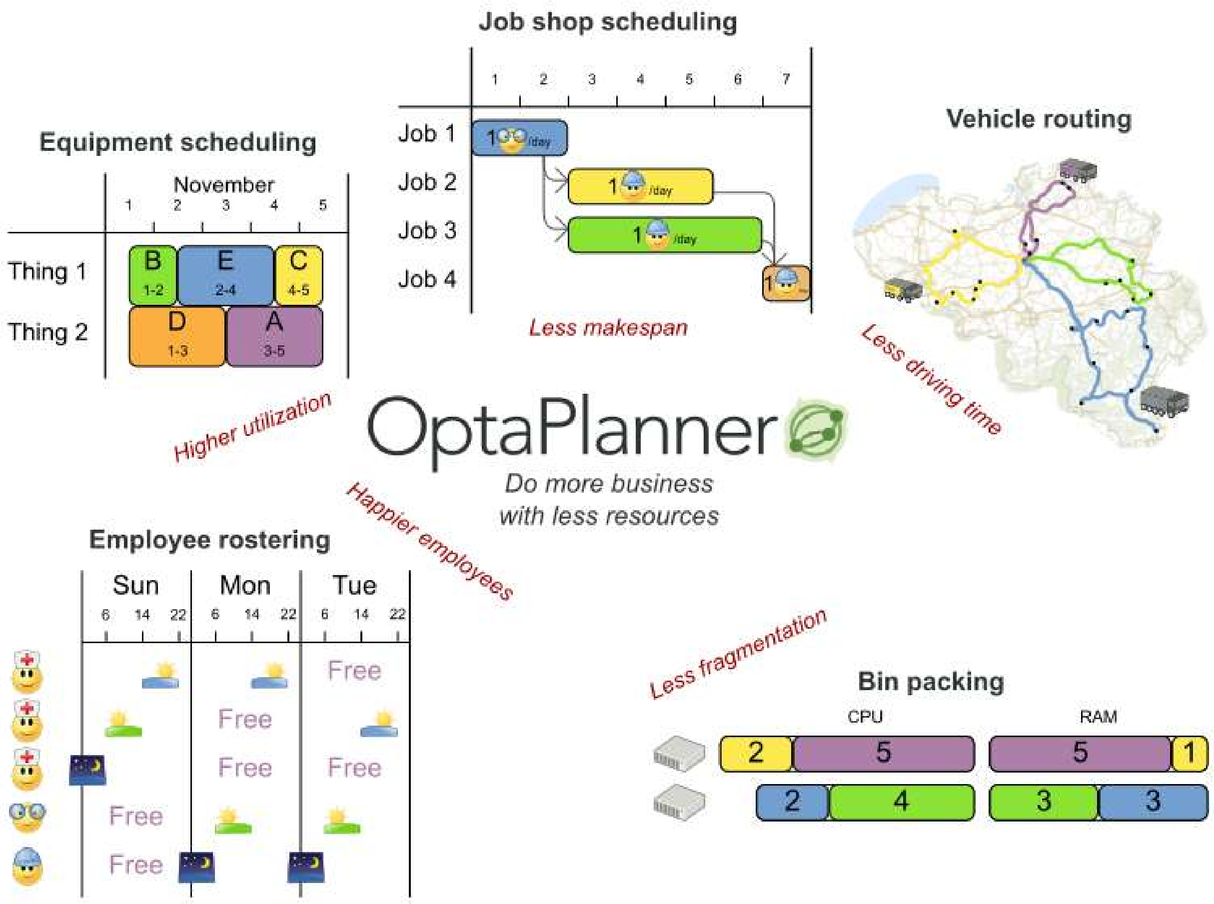
From 8181SGD

Request quote Request course date

1.4 WORKSHOP SEARCH REPRESENTATION

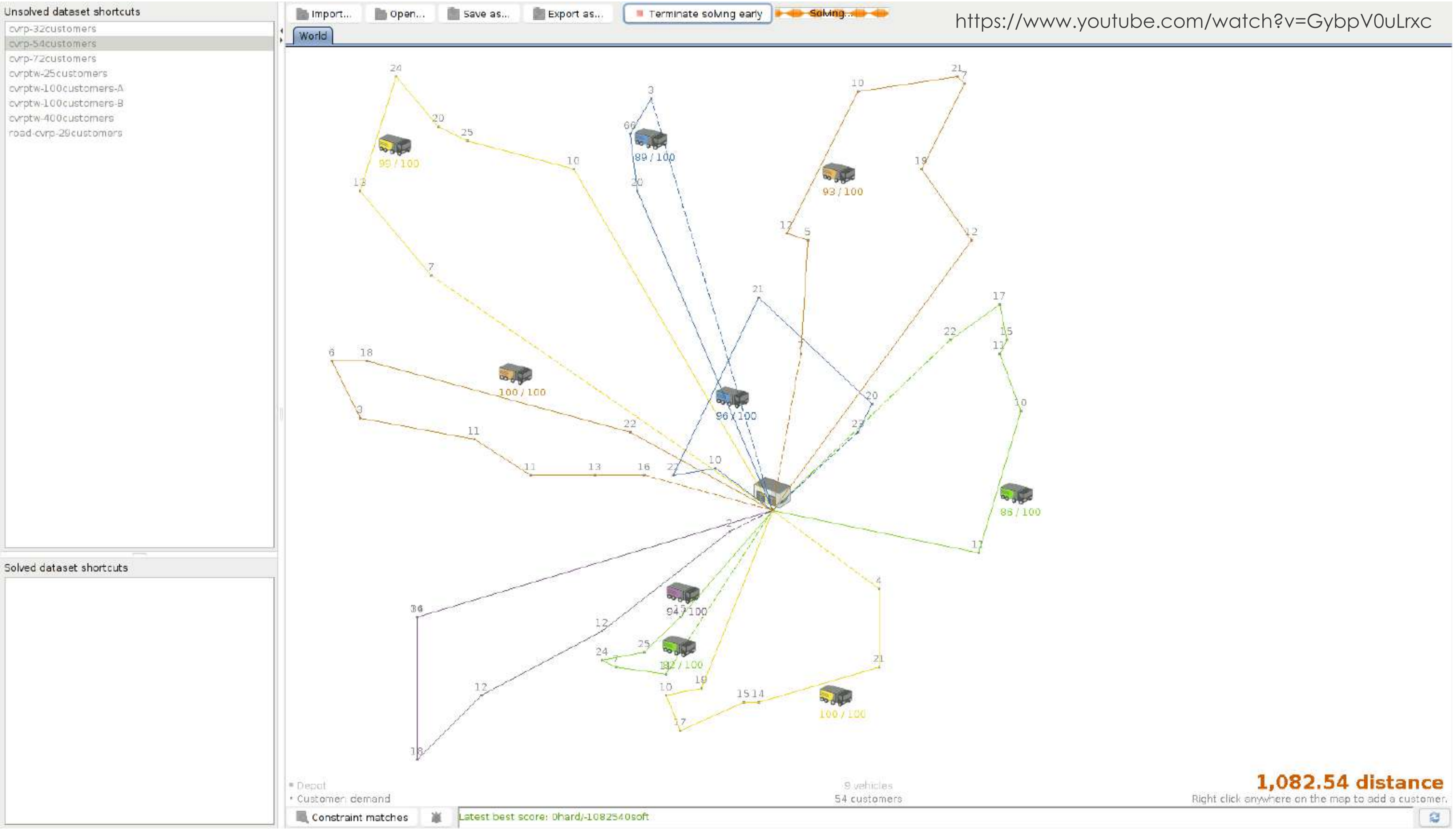
KIE OptaPlanner Tutorial

- **Constrain Satisfaction: Business Resource Optimizer**



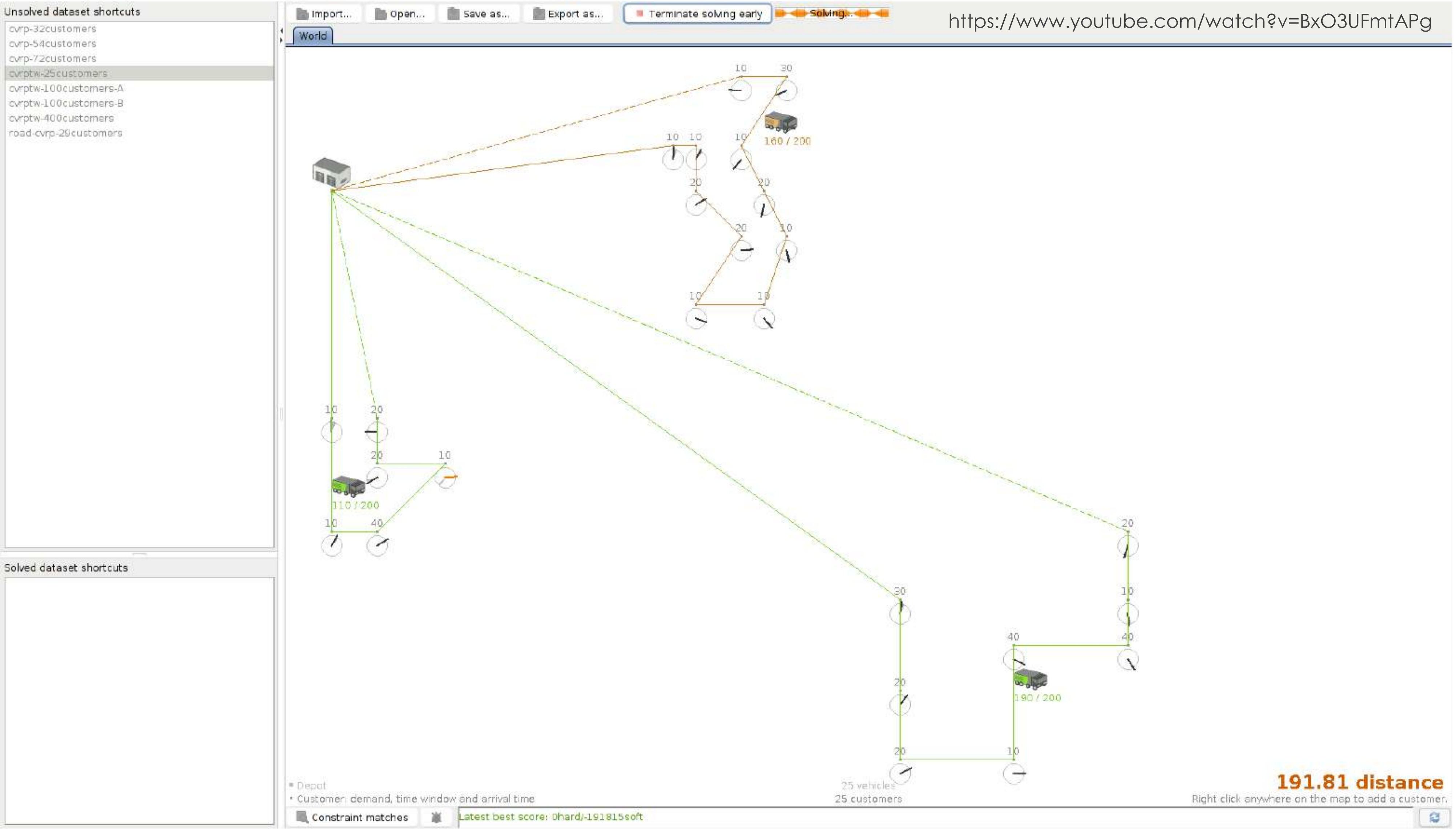
1.4 WORKSHOP SEARCH REPRESENTATION

KIE OptaPlanner Tutorial – VRP: Customer demand (vehicle load)



1.4 WORKSHOP SEARCH REPRESENTATION

KIE OptaPlanner Tutorial – VRP: Customer demand, Time window



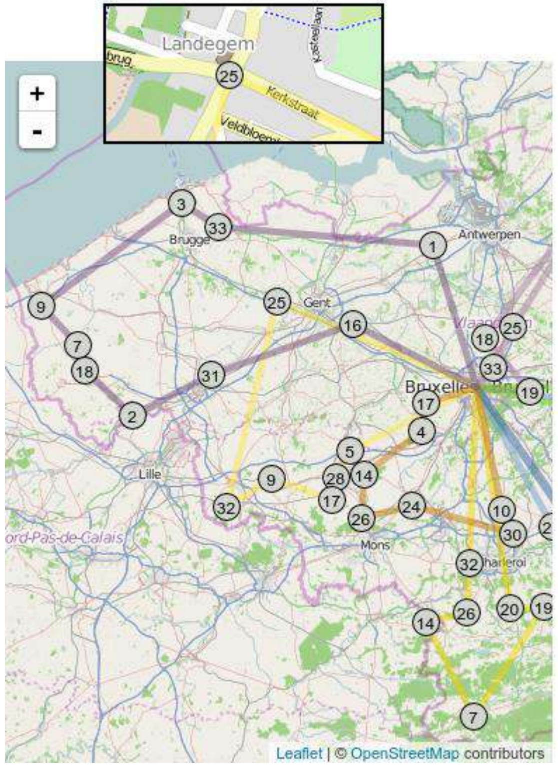
https://www.youtube.com/watch?v=BxO3UFmtAPg

1.4 WORKSHOP SEARCH REPRESENTATION

KIE OptaPlanner Tutorial – VRP with map integration

Visualizing Vehicle Routing with Leaflet and Google Maps

Leaflet.js



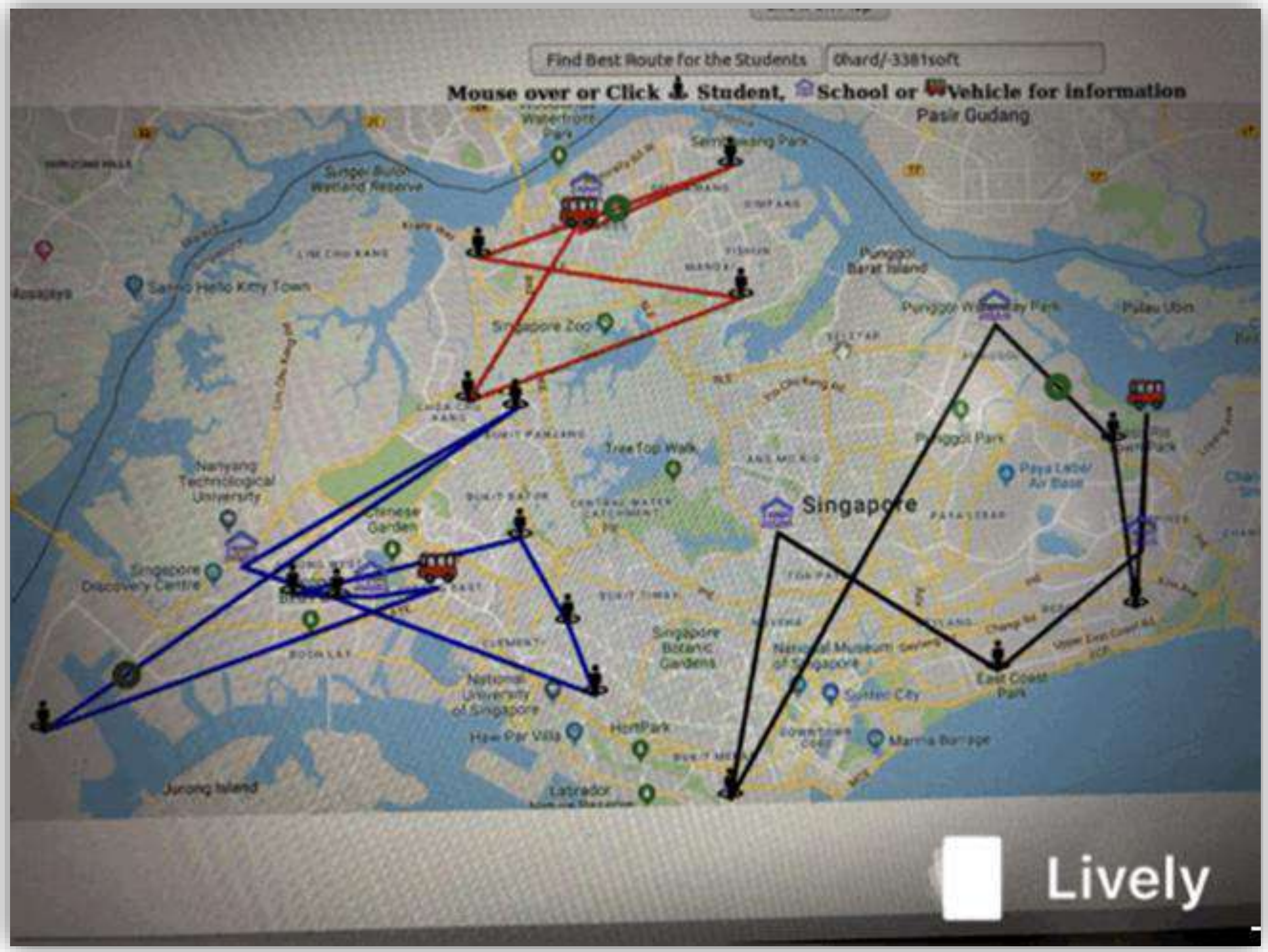
Google Maps

https://www.optaplanner.org/blog/2015/03/10/VisualizingVehicleRoutingWithLeafletAndGoogleMaps.html

1.4 WORKSHOP SEARCH REPRESENTATION

Past project – VRP with map integration

Intelligent Rapid Shuttle (IRS) System



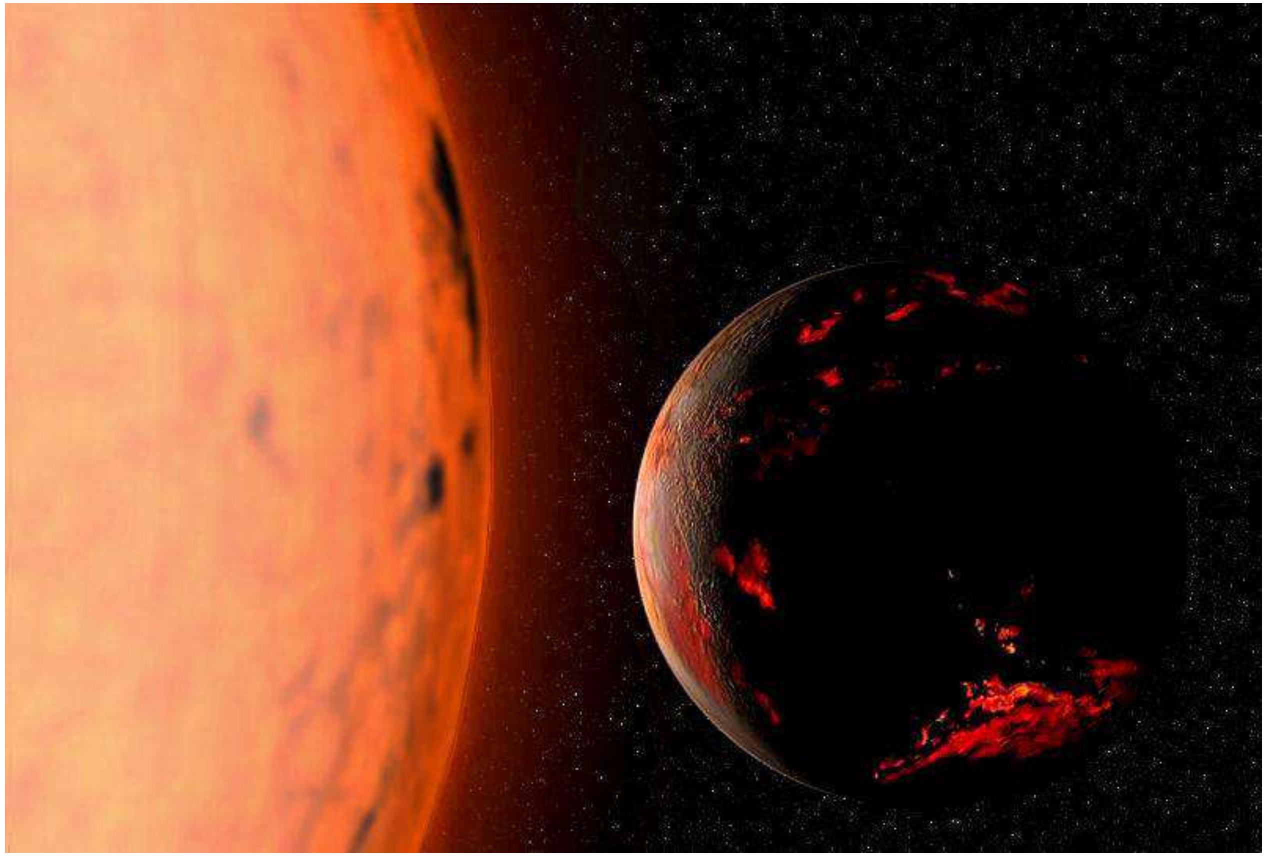
Source https://github.com/IRS-RS/IRS-RS-2019-03-09-IS1PT-GRP-aiVoyagers-irs-Intelligent-Rapid-Shuttle

1.4 WORKSHOP SEARCH REPRESENTATION

Business Travel Planning: Traveling Salesman Problem (TSP)

"Find the shortest route to visit N cities and finishing back at start city."

N = 20 cities, there are N! possible routes to consider: 20 x 19 x 18 x … x 2 x 1



Artist's concept of the carbonized Earth 7 billion years from now, after the Sun has entered the red giant stage.

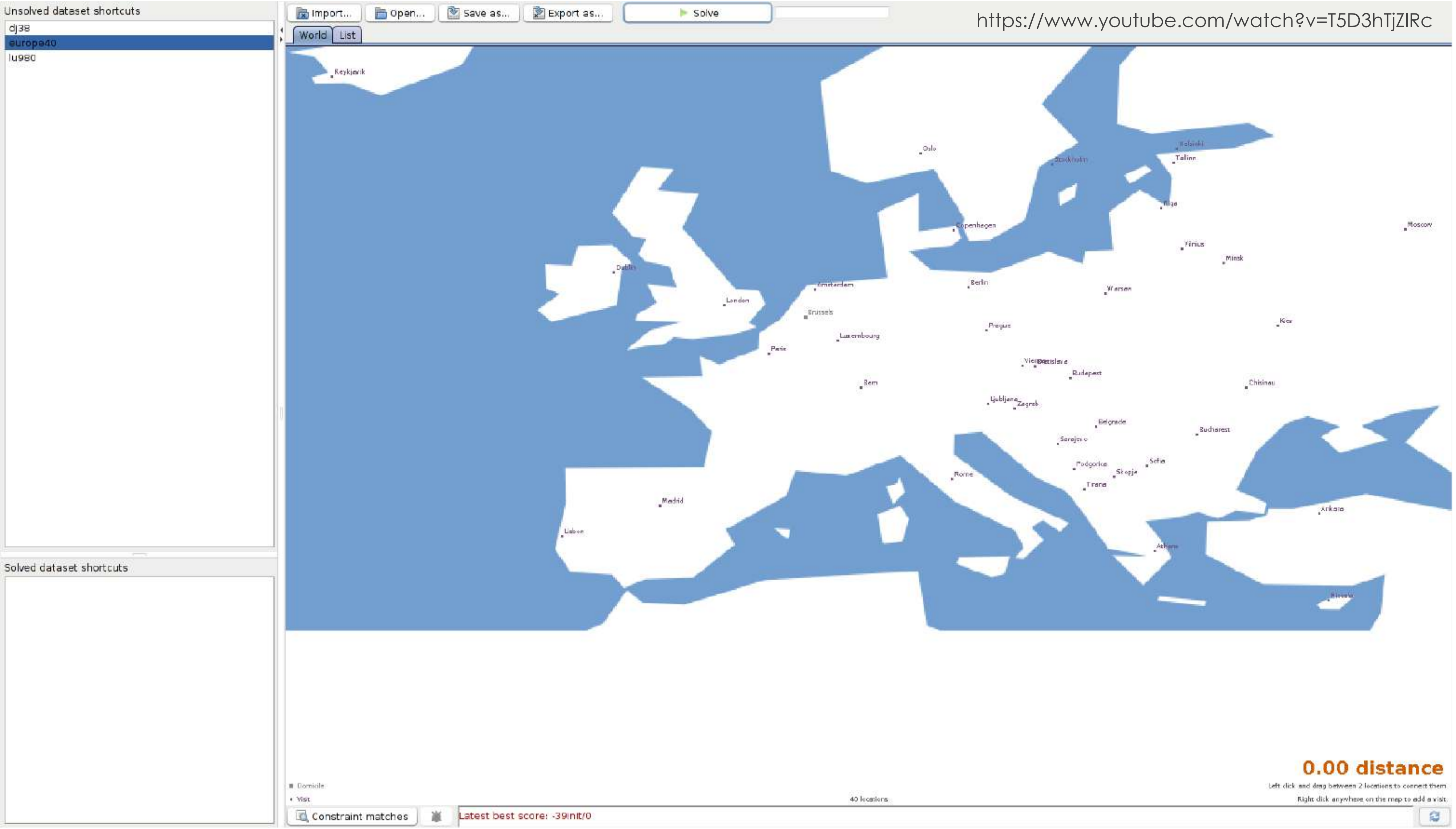
1.4 WORKSHOP SEARCH REPRESENTATION

KIE OptaPlanner Tutorial – TSP: Europe cities

https://www.youtube.com/watch?v=T5D3hTjZlRc

1.4 WORKSHOP SEARCH REPRESENTATION

KIE OptaPlanner Tutorial – TSP: Europe cities

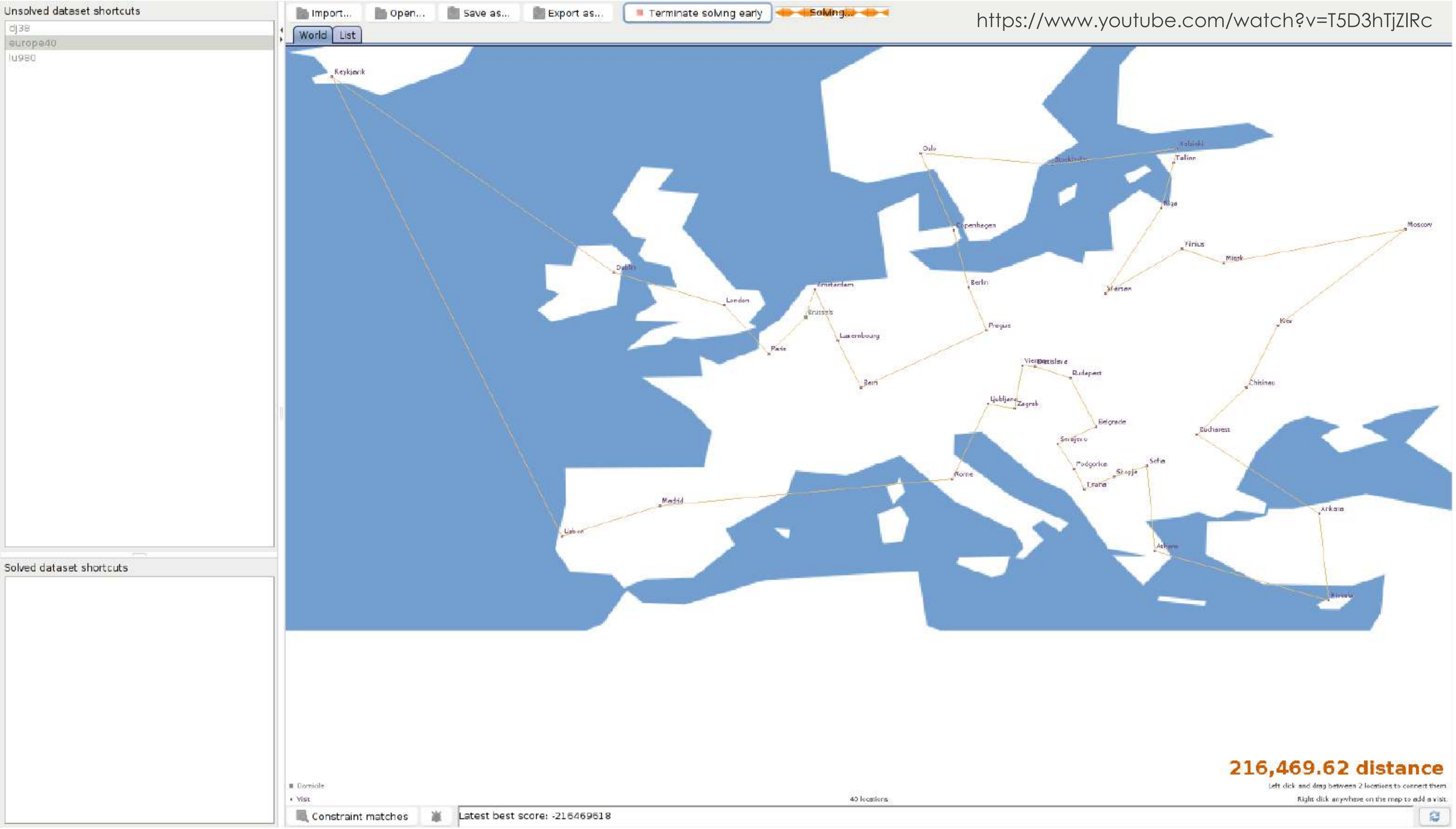


https://www.youtube.com/watch?v=T5D3hTjZlRc

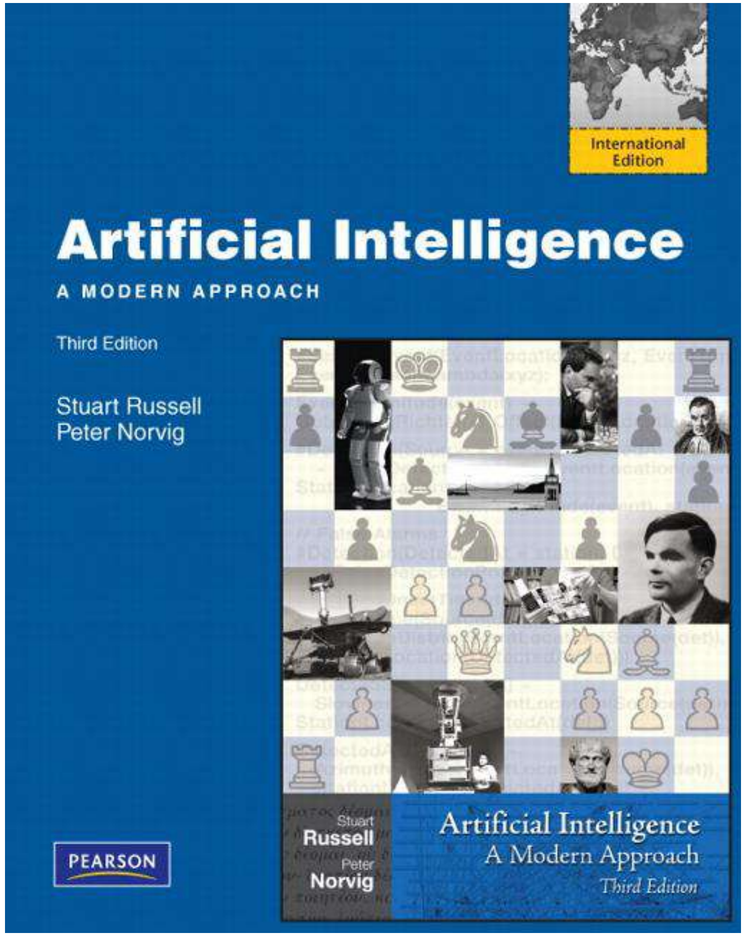
1.4 WORKSHOP SEARCH REPRESENTATION

KIE OptaPlanner Tutorial – TSP: Europe cities

https://www.youtube.com/watch?v=T5D3hTjZIRc



DAY 1 REFERENCE



1. **OptaPlanner : Do more business with less recourses**

   http://www.optaplanner.org/learn/slides/optaplanner-presentation/index.html#/1

2. OptaPlanner

   https://www.optaplanner.org/

3. OptaPlanner Use Cases & Demo Videos

   http://www.optaplanner.org/learn/useCases/index.html

4. OptaPlanner Video Tutorials

   http://www.optaplanner.org/learn/video.html

   https://www.youtube.com/user/ge0ffrey2

5. Onne Beek. (2011). Efficient Local Search Methods For Vehicle Routing

   https://lib.ugent.be/fulltxt/RUG01/001/788/544/RUG01-001788544_2012_0001_AC.pdf

DAY 1 SUMMARY

1.1 Reasoning Systems Overview

- History of Artificial Intelligence
- Question Answering System; Image Object Recognition; Chat-Bot
- Problem Solving: Analytic Tasks vs. Synthetic Tasks
- Synthetic Techniques: Search; Simulations; Genetic Algorithms; Data Mining

1.2 Uninformed Search Techniques

- Search Representation
- Depth First Search (DFS)
- Breadth First Search (BFS)

1.3 Informed Search Techniques (1/2)

- Heuristic Knowledge
- Evaluation (Scoring) Function; Heuristic Function; Past Cost Function
- Hill Climbing Search (HC); A Star Search (A*)

1.4 Search Representation Workshop

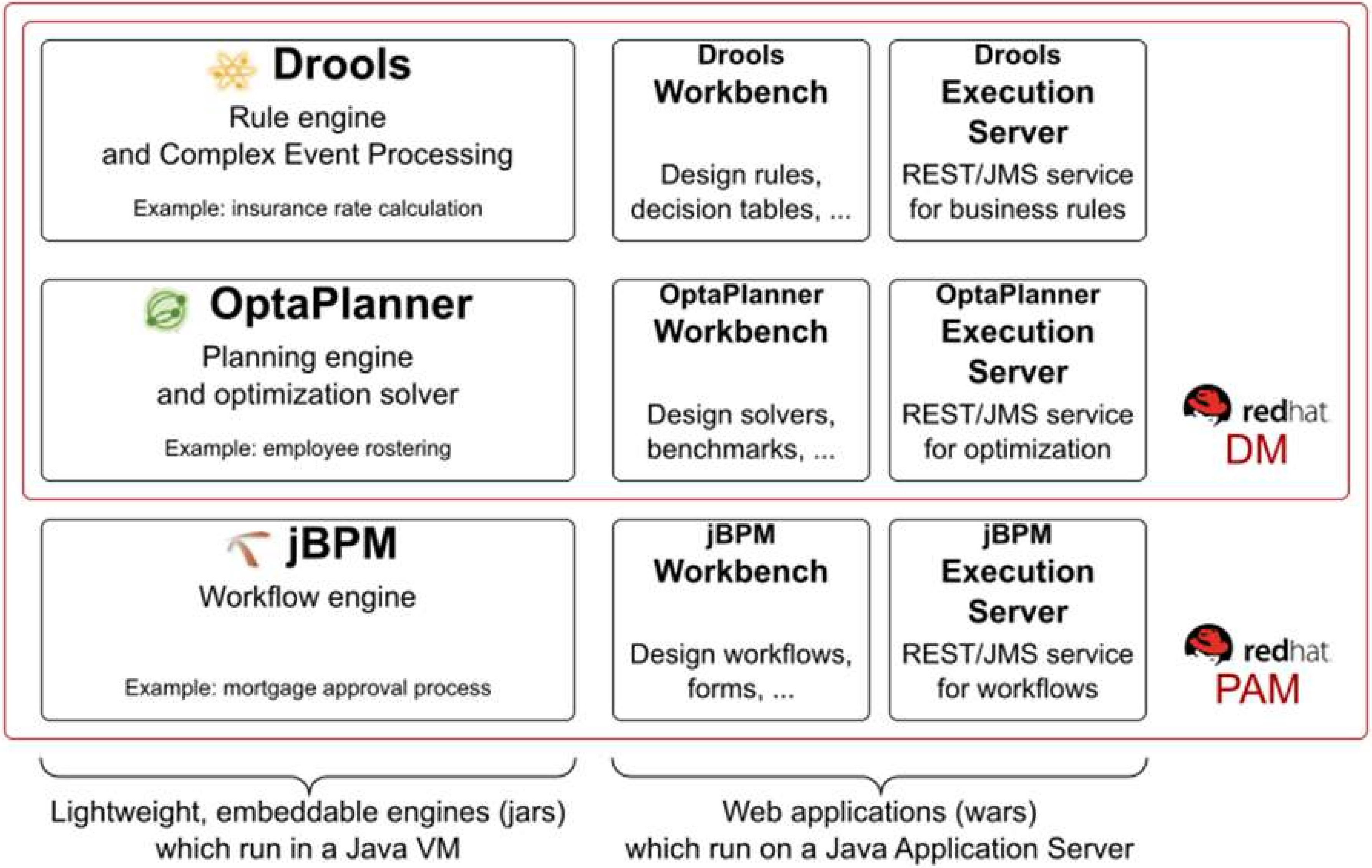
END OF LECTURE NOTES

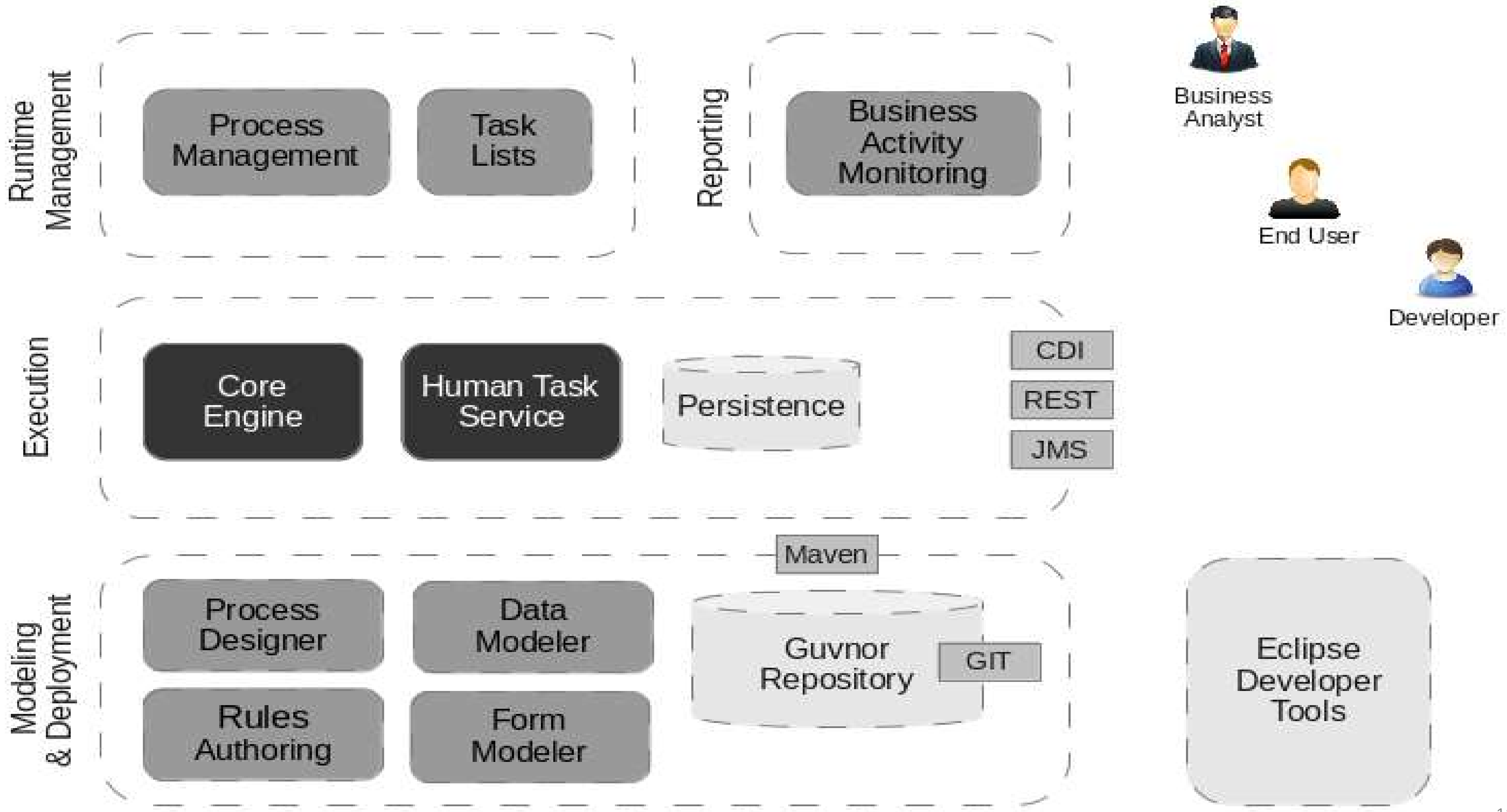
APPENDICES

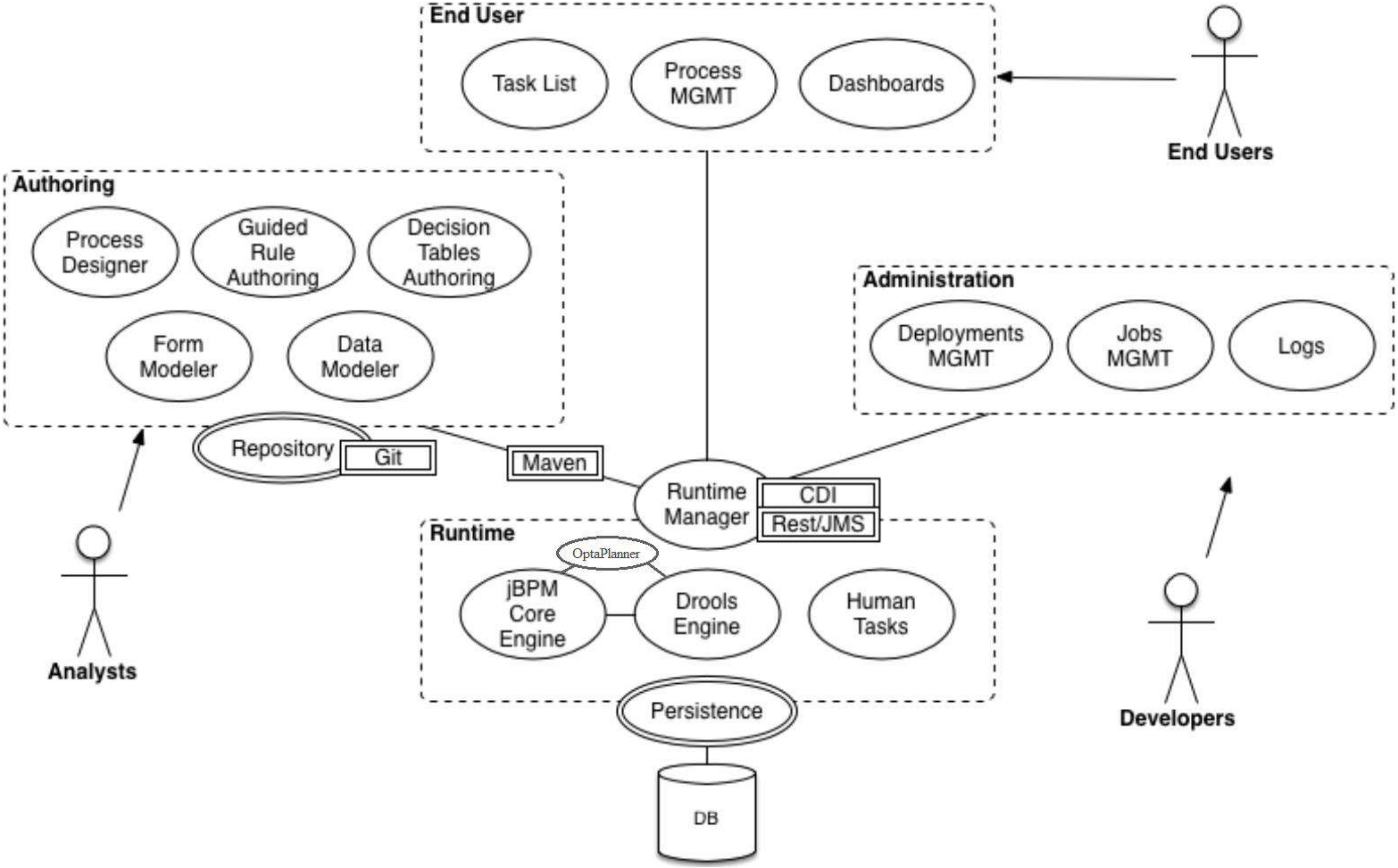
KIE System Architecture

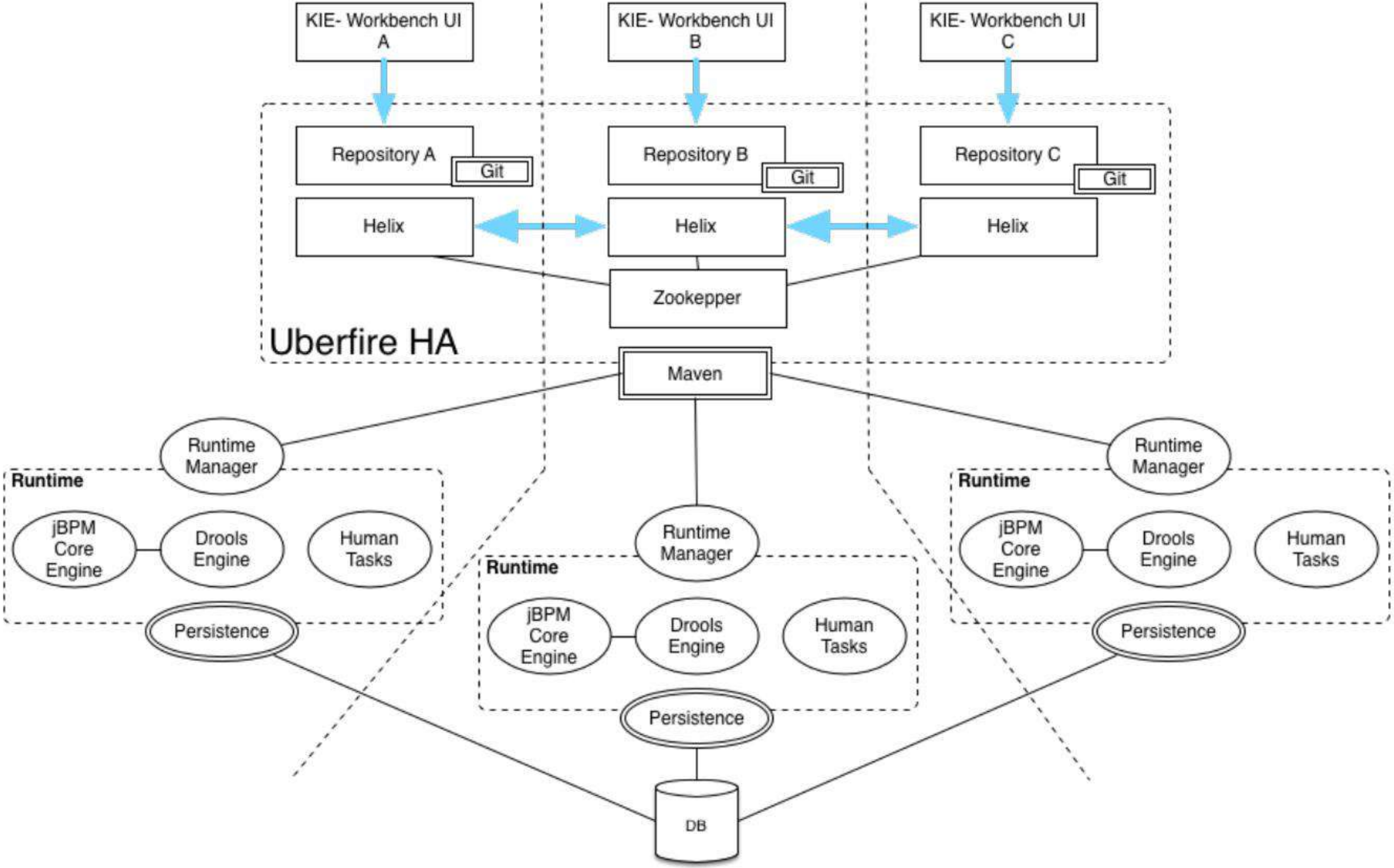
KIE functionality overview

What are the KIE projects?




Runtime Management
Process Management
Task Lists
Reporting
Business Activity Monitoring
Business Analyst
End User
Developer
Execution
Core Engine
Human Task Service
Persistence
CDI
REST
JMS
Maven
Modeling & Deployment
Process Designer
Data Modeler
Rules Authoring
Form Modeler
Guvnor Repository
GIT
Eclipse Developer Tools


End User
Task List
Process MGMT
Dashboards
End Users
Authoring
Process Designer
Guided Rule Authoring
Decision Tables Authoring
Form Modeler
Data Modeler
Administration
Deployments MGMT
Jobs MGMT
Logs
Repository
Git
Maven
Runtime Manager
CDI
Rest/JMS
Runtime
OptaPlanner
jBPM Core Engine
Drools Engine
Human Tasks
Persistence
DB
Analysts
Developers


KIE- Workbench UI A
KIE- Workbench UI B
KIE- Workbench UI C
Repository A
Git
Repository B
Git
Repository C
Git
Helix
Helix
Helix
Zookepper
Uberfire HA
Maven
Runtime Manager
Runtime
jBPM Core Engine
Drools Engine
Human Tasks
Persistence
Runtime Manager
Runtime
jBPM Core Engine
Drools Engine
Human Tasks
Persistence
Runtime Manager
Runtime
jBPM Core Engine
Drools Engine
Human Tasks
Persistence
DB

END OF APPENDICES