

# PROBLEM SOLVING AND SEARCH IN ARTIFICIAL INTELLIGENCE

## Lecture 3 Uninformed Search vs. Informed Search

Sarah Gaggl

Dresden, 28th April 2017

# Agenda

- 1 Introduction
- 2 Constraint Satisfaction (CSP)
- 3 Uninformed Search versus Informed Search (Best First Search, A\* Search, Heuristics)
- 4 Local Search, Stochastic Hill Climbing, Simulated Annealing
- 5 Tabu Search
- 6 Answer-set Programming (ASP)
- 7 Structural Decomposition Techniques (Tree/Hypertree Decompositions)
- 8 Evolutionary Algorithms/ Genetic Algorithms

# Traditional Methods

- There are many classic algorithms to search spaces for an optimal solution.
- Broadly, they fall into two disjoint classes:
  - Algorithms that only evaluate **complete solutions** (exhaustive search, local search, ...).
  - Algorithms that require the evaluation of **partially constructed** or approximate solutions.
- Algorithms that treat **complete** solutions can be **stopped any time**, and give at least one **potential answer**.
- If you interrupt an algorithm that works on **partial** solutions, the **results might be useless**.

# Complete Solutions

- All decision variables are specified.
- For example, binary strings of length  $n$  constitute complete solutions for any  $n$ -variable SAT.
- Permutations of  $n$  cities constitute complete solutions for a TSP.
- We can compare two complete solutions using an evaluation function.
- Many algorithms rely on such comparisons, manipulating one single complete solution at a time.
- When a new solution has a better evaluation than the previous best solution, it replaces that prior solution.
- Exhaustive search, local search, hill climbing as well as modern heuristic methods such as simulated annealing, tabu search and evolutionary algorithms fall into this category.

# Partial Solutions

There are two forms:

- 1 incomplete solution to the problem originally posed, and
  - 2 complete solution to a reduced (i.e. simpler) problem.
- Incomplete solutions reside in a subset of the original problem's search space.
    - In an SAT, consider all of the binary strings where the first two variables were assigned the value 1 (i.e. TRUE).
    - In a TSP, consider every permutation of cities that contains the sequence 7 – 11 – 2 – 16.
    - We fix the attention on a subset of the search space that has a partial property.
    - Hopefully, that property is also shared by the real solution!

# Partial Solutions ctd.

- **Decompose** original problem into a set of **smaller** and **simpler** problems.
  - Hope: solving each of the easier problems and **combine the partial solutions**, results in an answer for the original problem.
  - In a TSP, consider only  $k$  out of  $n$  cities and try to establish the shortest path from city  $i$  to  $j$  that passes through all  $k$  of these cities.
  - **Reduce** the **size of the search space** significantly and search for a complete solution within the restricted domain.
  - Such partial solutions can serve as **building blocks** for the solution to the original problem.

# Partial Solutions ctd.

- **Decompose** original problem into a set of **smaller** and **simpler** problems.
  - Hope: solving each of the easier problems and **combine the partial solutions**, results in an answer for the original problem.
  - In a TSP, consider only  $k$  out of  $n$  cities and try to establish the shortest path from city  $i$  to  $j$  that passes through all  $k$  of these cities.
  - **Reduce** the **size of the search space** significantly and search for a complete solution within the restricted domain.
  - Such partial solutions can serve as **building blocks** for the solution to the original problem.
- But, algorithms that work on partial solutions pose **additional difficulties**. One needs to
  - devise a way to **organize the sub-spaces** so that they can be searched efficiently, and
  - create a **new evaluation function** that can assess the quality of partial solutions.

# Exhaustive Search

- Checks **every** solution in the search space until the **best global** solution has been found.
- Can be used **only for small instances** of problems.
- Exhaustive (**enumerative**) algorithms are **simple**.
- Search space can be reduced by **backtracking**.
- Some optimization methods, e.g., **branch and bound** and **A\*** are based on an exhaustive search.



# Exhaustive Search

- Checks **every** solution in the search space until the **best global** solution has been found.
- Can be used **only for small instances** of problems.
- Exhaustive (**enumerative**) algorithms are **simple**.
- Search space can be reduced by **backtracking**.
- Some optimization methods, e.g., **branch and bound** and **A\*** are based on an exhaustive search.
- **How** can we generate a **sequence** of every possible solution to the problem?
  - The **order** in which the solutions are generated and evaluated is **irrelevant** (because we evaluate all of them).
  - The **answer** for the question depends on the selected **representation**.

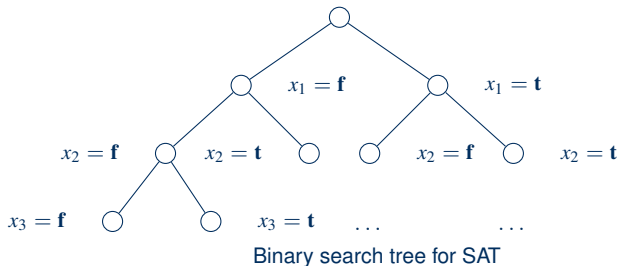
# Enumerating the SAT

- We have to generate every possible binary string of length  $n$ .
- All solutions correspond to whole numbers in a one-to-one mapping.
- Generate all non-negative integers from 0 to  $2^n - 1$  and convert each of these integers into the matching binary string of length  $n$ .

0000	0	0100	4	1000	8	1100	12
0001	1	0101	5	1001	9	1101	13
0010	2	0110	6	1010	10	1110	14
0011	3	0111	7	1011	11	1111	15

- Bits of the string are the truth assignments of the decision variables.
- Organize the search space, for example partition into two disjoint sub-spaces. First contains all the vectors where  $x_1 = \text{f}$  (FALSE), and the second contains all vectors where  $x_1 = \text{t}$  (TRUE).

# Enumerating the SAT ctd.



# Search Strategies

A strategy is defined by picking the **order of node expansion**.

Strategies are evaluated along the following dimensions:

- **Completeness** - does it always find a solution if one exists?
- **Time complexity** - number of nodes generated/expanded.
- **Space complexity** - maximum number of nodes in memory.
- **Optimality** - does it always find a least-cost solution?

Time and space complexity are 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  $\infty$ ).

# Group Work - Posters

- Uninformed Search Strategies
- Informed Search Strategies

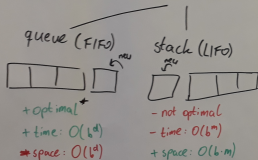


# Uninformed Search Strategies

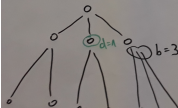
## Uninformed search strategies

### Breadth first and Depth first

- discover next node in structure



\* for ~~monotonic~~ monotonic-rising edge costs



$O(m)$  ... with back-tracking  
completeness in graphs:  
check for loops!

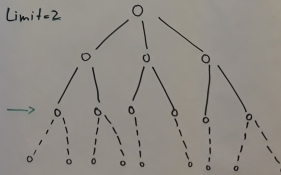


### Iterative Deepening

- general strategy  $\rightarrow$  finds best depth limit
- $\hookrightarrow$  depth increase  $\rightarrow$  goal found

- + optimal
- + time:  $O(b^d)$
- + space:  $O(bd)$

Limit=2



# Informed Search Strategies

Informierte Suche  
in Graphen

$F(n)$  Bewertung des Knotens  $n$   
z.B. Distanz zum Ziel


$h(n)$  Heuristik ist (Unter)schätzung  
der Kosten des günstigsten  
Pfads vom aktuellen Knoten  
 $n$  zum Ziel.

Greedy - Suche:

- nutzt ~~ne~~ Heuristik als Bewertungsfkt.
- der nächst expandierte Knoten ist der billigste
- nicht vollständig, nicht optimal (da lokales Opt.)

$A^*$  - Suche:

- Kostenfkt  $F(n) = g(n) + h(n)$ , wobei
  - $g(n)$  - bisherige Kosten
  - $h(n)$  - geschätzte Kosten zum Ziel
- Anforderungen an  $h$ :
  - Unterschätzung
  - $h(n) \geq 0$ ,  $h(\text{Ziel}) = 0$
  - $h(n) \leq c(g, a, n') + h(n')$   
(Konsistenzkriterium)
- vollständig, optimal



# Dynamic Programming

Principle of finding an overall solution by operating on an **intermediate point** that lies between where you are now and where you want to go.

- Procedure is **recursive**, each next intermediate point is a function of the points already visited.
- Prototypical problem suitable for dynamic programming has the following properties.



# Dynamic Programming

Principle of finding an overall solution by operating on an **intermediate point** that lies between where you are now and where you want to go.

- Procedure is **recursive**, each next intermediate point is a function of the points already visited.
- Prototypical problem suitable for dynamic programming has the following properties.
  - Can be decomposed into a sequence of decisions made at various stages.
  - Each stage has a number of possible states.
  - A decision takes you from a state at one stage to some state at the next stage.
  - Best sequence of decisions (**policy**) at any stage is independent of the decisions made at prior stages.
  - Well-defined cost for traversing from state to state across stages.
  - There is a recursive relationship from choosing the best decisions to make.

# Dynamic Programming ctd.

## Procedure

- Starting at the goal and working backward to the current state.
- First, determine the best decision at last stage.
- From there, determine the best decision at the next to last stage, presuming we will make the best decision at the last stage.
- And so forth . . .

# Dynamic Program for the TSP

$$L = \begin{bmatrix} 0 & 7 & 12 & 8 & 11 \\ 3 & 0 & 10 & 7 & 13 \\ 4 & 8 & 0 & 9 & 12 \\ 6 & 6 & 9 & 0 & 10 \\ 7 & 7 & 11 & 10 & 0 \end{bmatrix}$$

- Suppose, we start from city 1.
- We split the problem into smaller problems.
- $g(i, S)$  length of the shortest path from city  $i$  to 1 that passes through **each** city in  $S$ .
- $g(4, \{5, 2, 3\})$  is the shortest path from city 4 through cities 5, 2 and 3 (in some unspecified order) and then returns to 1.
- $g(1, V - \{1\})$  is the length of the shortest complete tour.
- In general, we claim that

$$g(i, S) = \min_{j \in S} \{L(i, j) + g(j, S - \{j\})\}.$$

# Dynamic Program for the TSP ctd.

$$L = \begin{bmatrix} 0 & 7 & 12 & 8 & 11 \\ 3 & 0 & 10 & 7 & 13 \\ 4 & 8 & 0 & 9 & 12 \\ 6 & 6 & 9 & 0 & 10 \\ 7 & 7 & 11 & 10 & 0 \end{bmatrix}$$

The problem is to find  $g(1, \{2, 3, 4, 5\})$ .

We start backwards with  $S = \emptyset$ .

$$g(2, \emptyset) = L(2, 1) = 3,$$

$$g(3, \emptyset) = L(3, 1) = 4,$$

$$g(4, \emptyset) = L(4, 1) = 6, \text{ and}$$

$$g(5, \emptyset) = L(5, 1) = 7.$$

# Dynamic Program for the TSP ctd.

$$L = \begin{bmatrix} 0 & 7 & 12 & 8 & 11 \\ 3 & 0 & 10 & 7 & 13 \\ 4 & 8 & 0 & 9 & 12 \\ 6 & 6 & 9 & 0 & 10 \\ 7 & 7 & 11 & 10 & 0 \end{bmatrix}$$

Next iteration, find the solutions to all problems where  $|S| = 1$  (12 sub-problems).

$$g(2, \{3\}) = L(2, 3) + g(3, \emptyset) = 10 + 4 = 14,$$

$$g(2, \{4\}) = L(2, 4) + g(4, \emptyset) = 7 + 6 = 13, \text{ and}$$

$$g(2, \{5\}) = L(2, 5) + g(5, \emptyset) = 13 + 7 = 20.$$

# Dynamic Program for the TSP ctd.

For city 3:

$$g(3, \{2\}) = L(3, 2) + g(2, \emptyset) = 8 + 3 = 11,$$

$$g(3, \{4\}) = L(3, 4) + g(4, \emptyset) = 9 + 6 = 15,$$

$$g(3, \{5\}) = L(3, 5) + g(5, \emptyset) = 12 + 7 = 19.$$

For city 4:

$$g(4, \{2\}) = L(4, 2) + g(2, \emptyset) = 6 + 3 = 9,$$

$$g(4, \{3\}) = L(4, 3) + g(3, \emptyset) = 9 + 4 = 13,$$

$$g(4, \{5\}) = L(4, 5) + g(5, \emptyset) = 10 + 7 = 17.$$

For city 5:

$$g(5, \{2\}) = L(5, 2) + g(2, \emptyset) = 7 + 3 = 10,$$

$$g(5, \{3\}) = L(5, 3) + g(3, \emptyset) = 11 + 4 = 15,$$

$$g(5, \{4\}) = L(5, 4) + g(4, \emptyset) = 10 + 6 = 16.$$

# Dynamic Program for the TSP ctd.

$$L = \begin{bmatrix} 0 & 7 & 12 & 8 & 11 \\ 3 & 0 & 10 & 7 & 13 \\ 4 & 8 & 0 & 9 & 12 \\ 6 & 6 & 9 & 0 & 10 \\ 7 & 7 & 11 & 10 & 0 \end{bmatrix}$$

Next iteration,  $|S| = 2$ .

$$\begin{aligned} g(2, \{3, 4\}) &= \min\{L(2, 3) + g(3, \{4\}), L(2, 4) + g(4, \{3\})\} \\ &= \min\{10 + 15, 7 + 13\} = \min\{25, 20\} = 20, \end{aligned}$$

$$\begin{aligned} g(2, \{3, 5\}) &= \min\{L(2, 3) + g(3, \{5\}), L(2, 5) + g(5, \{3\})\} \\ &= \min\{10 + 19, 13 + 15\} = \min\{29, 28\} = 28, \end{aligned}$$

$$\begin{aligned} g(2, \{4, 5\}) &= \min\{L(2, 4) + g(4, \{5\}), L(2, 5) + g(5, \{4\})\} \\ &= \min\{7 + 17, 13 + 16\} = \min\{24, 29\} = 24. \end{aligned}$$

# Dynamic Program for the TSP ctd.

For city 3:

$$g(3, \{2, 5\}) = \min\{L(3, 2) + g(2, \{5\}), L(3, 5) + g(5, \{2\})\}$$

$$= \min\{8 + 20, 12 + 10\} = \min\{28, 22\} = 22,$$

$$g(3, \{2, 4\}) = \min\{L(3, 2) + g(2, \{4\}), L(3, 4) + g(4, \{2\})\}$$

$$= \min\{8 + 13, 9 + 9\} = \min\{21, 18\} = 18,$$

$$g(3, \{4, 5\}) = \min\{L(3, 4) + g(4, \{5\}), L(3, 5) + g(5, \{4\})\}$$

$$= \min\{9 + 17, 12 + 16\} = \min\{26, 28\} = 26.$$

For city 4:

$$g(4, \{2, 3\}) = \min\{L(4, 2) + g(2, \{3\}), L(4, 3) + g(3, \{2\})\}$$

$$= \min\{6 + 14, 9 + 11\} = \min\{20, 20\} = 20,$$

$$g(4, \{2, 5\}) = \min\{L(4, 2) + g(2, \{5\}), L(4, 5) + g(5, \{2\})\}$$

$$= \min\{6 + 20, 10 + 10\} = \min\{26, 20\} = 20,$$

$$g(4, \{3, 5\}) = \min\{L(4, 3) + g(3, \{5\}), L(4, 5) + g(5, \{3\})\}$$

$$= \min\{9 + 19, 10 + 15\} = \min\{28, 25\} = 25.$$



# Dynamic Program for the TSP ctd.

For city 5:

$$\begin{aligned}g(5, \{2, 3\}) &= \min\{L(5, 2) + g(2, \{3\}), L(5, 3) + g(3, \{2\})\} \\&= \min\{7 + 14, 11 + 11\} = \min\{21, 22\} = 21,\end{aligned}$$

$$\begin{aligned}g(5, \{2, 4\}) &= \min\{L(5, 2) + g(2, \{4\}), L(5, 4) + g(4, \{2\})\} \\&= \min\{7 + 13, 10 + 19\} = \min\{20, 29\} = 20,\end{aligned}$$

$$\begin{aligned}g(5, \{3, 4\}) &= \min\{L(5, 3) + g(3, \{4\}), L(5, 4) + g(4, \{3\})\} \\&= \min\{11 + 15, 10 + 13\} = \min\{26, 23\} = 23.\end{aligned}$$

# Dynamic Program for the TSP ctd.

$$L = \begin{bmatrix} 0 & 7 & 12 & 8 & 11 \\ 3 & 0 & 10 & 7 & 13 \\ 4 & 8 & 0 & 9 & 12 \\ 6 & 6 & 9 & 0 & 10 \\ 7 & 7 & 11 & 10 & 0 \end{bmatrix}$$

Next iteration,  $|S| = 3$ .

$$\begin{aligned} g(2, \{3, 4, 5\}) &= \min\{L(2, 3) + g(3, \{4, 5\}), L(2, 4) + g(4, \{3, 5\}), L(2, 5) + g(5, \{3, 4\})\} \\ &= \min\{10 + 26, 7 + 25, 13 + 23\} = \min\{36, 32, 34\} = 32, \end{aligned}$$

# Dynamic Program for the TSP ctd.

Next iteration,  $|S| = 3$ .

$$\begin{aligned}g(2, \{3, 4, 5\}) &= \min\{L(2, 3) + g(3, \{4, 5\}), L(2, 4) + g(4, \{3, 5\}), L(2, 5) + g(5, \{3, 4\})\} \\&= \min\{10 + 26, 7 + 25, 13 + 23\} = \min\{36, 32, 34\} = 32,\end{aligned}$$

$$\begin{aligned}g(3, \{2, 4, 5\}) &= \min\{L(3, 2) + g(2, \{4, 5\}), L(3, 4) + g(4, \{2, 5\}), L(3, 5) + g(5, \{2, 4\})\} \\&= \min\{8 + 24, 9 + 20, 12 + 20\} = \min\{32, 29, 32\} = 29,\end{aligned}$$

$$\begin{aligned}g(4, \{2, 3, 5\}) &= \min\{L(4, 2) + g(2, \{3, 5\}), L(4, 3) + g(3, \{2, 5\}), L(4, 5) + g(5, \{2, 3\})\} \\&= \min\{6 + 28, 9 + 22, 10 + 21\} = \min\{34, 31, 31\} = 31.\end{aligned}$$

$$\begin{aligned}g(5, \{2, 3, 4\}) &= \min\{L(5, 2) + g(2, \{3, 4\}), L(5, 3) + g(3, \{2, 4\}), L(5, 4) + g(4, \{2, 3\})\} \\&= \min\{7 + 20, 11 + 18, 10 + 20\} = \min\{27, 29, 30\} = 27.\end{aligned}$$

# Dynamic Program for the TSP ctd.

$$L = \begin{bmatrix} 0 & 7 & 12 & 8 & 11 \\ 3 & 0 & 10 & 7 & 13 \\ 4 & 8 & 0 & 9 & 12 \\ 6 & 6 & 9 & 0 & 10 \\ 7 & 7 & 11 & 10 & 0 \end{bmatrix}$$

Last iteration,  $|S| = 4$ , **original problem:**

$$\begin{aligned} g(1, \{2, 3, 4, 5\}) &= \min\{L(1, 2) + g(2, \{3, 4, 5\}), L(1, 3) + g(3, \{2, 4, 5\}), \\ &\quad L(1, 4) + g(4, \{2, 3, 5\}), L(1, 5) + g(5, \{2, 3, 4\})\} \\ &= \min\{7 + 32, 12 + 29, 8 + 31, 11 + 27\} = \min\{39, 41, 39, 38\} = 38. \end{aligned}$$

**Shortest tour has length 38.**

**Which tour is that?**

# Dynamic Program for the TSP ctd.

Last iteration,  $|S| = 4$ , **original problem**:

$$\begin{aligned}g(1, \{2, 3, 4, 5\}) &= \min\{L(1, 2) + g(2, \{3, 4, 5\}), L(1, 3) + g(3, \{2, 4, 5\}), \\&\quad L(1, 4) + g(4, \{2, 3, 5\}), L(1, 5) + g(5, \{2, 3, 4\})\} \\&= \min\{7 + 32, 12 + 29, 8 + 31, 11 + 27\} = \min\{39, 41, 39, 38\} = 38.\end{aligned}$$

Shortest tour has length 38. Which tour is that?

- **Additional data structure  $W$**  with information on the next city with **minimal path**.
- $W(1, \{2, 3, 4, 5\}) = 5$ .
- $W(5, \{2, 3, 4\}) = 2$ ,  $W(2, \{3, 4\}) = 4$ ,  $W(4, \{3\}) = 3$ ,
- last we arrive at city 1.
- Length of this tour is  $11 + 7 + 7 + 9 + 4 = 38$ .

# Properties of Dynamic Programming

- Computationally intensive:  $O(n^2 2^n)$ .
- DP algorithms tend to be complicated to understand, because the construction of the program depends on the problem.
- How to formulate sub-problems?

# Summary

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest  $h$ 
  - incomplete and not always optimal
- Dynamic programming
  - complete and optimal
  - time and space consuming
  - how to define the sub-problems?
- A\* search expands lowest  $g + h$ 
  - complete and optimal
  - also optimally efficient
- Admissible heuristics can be derived from exact solution of relaxed problems

# References



Zbigniew Michalewicz and David B. Fogel.

**How to Solve It: Modern Heuristics**, volume 2. Springer, 2004.



Stuart J. Russell and Peter Norvig.

**Artificial Intelligence - A Modern Approach (3. edition)**. Pearson Education, 2010.