Tecnológico de Monterrey

CS4000 Intelligent Systems

Solving Problems with Informed Search

Santiago Conant, Ph.D.
sconant@itesm.mx

Contents

- Informed search concepts
- Heuristic search algorithms
  - greedy search
  - A* search
- Variants of the A* algorithm
- Heuristic design
- Other uses of heuristics
- Conclusions

Heuristic or informed search

- Use some method to control or guide the search for a solution (**heuristics**).
  - Educated assumptions.
  - Intuitive criteria.
  - Common sense.
- What are heuristics used for?
  - To order a search.
    - Search first on the most likely nodes.
  - To control the search width.
    - Try the deepest at the expense of the width.
- Types of heuristics
  - Goal-oriented.
    - Knowing what you are trying to find.
  - Based on knowledge.
    - Using specific domain knowledge to depurate your search.

Heuristics

- **Informed vs uninformed search**
  - DFS, BFS, iterated deepening are inefficient
  - You want to make a more intelligent search exploiting domain knowledge and constraints of the problem.
- *General mechanism:* **Evaluation function (f)**
  - Choose from the queue nodes, always expanding the "best" in each iteration.
  - Estimate "quality" of each node, by means of the estimated cost of the cheapest route to a target state (**h**).
  - It does not have to be exact, any clue is good.

Examples of heuristics

Route between cities
Straight distance to destination city.

Traveling salesman
Path length.

8-puzzle
Number of pieces out of place.

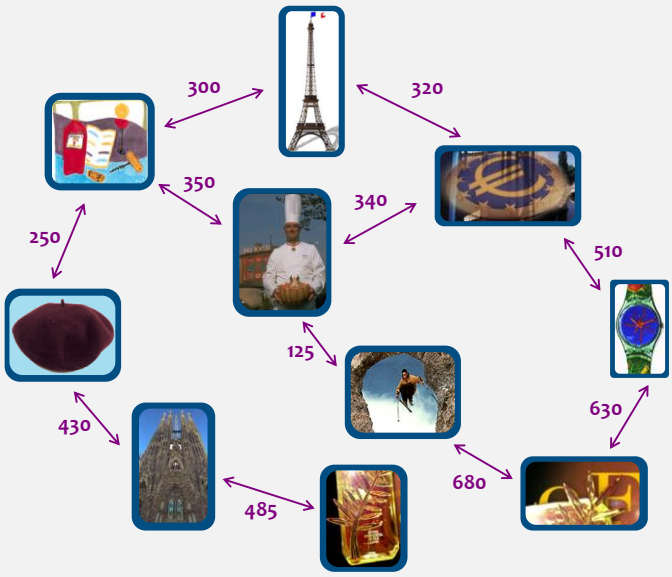
N-queens
Total attacks between queens.

Chess
Material value (Queen=3, pawn=1).

Greedy search

- "Best-first" search where **f (n) = h (n)**
- **Algorithm**:
  - Initialize the cost of the initial node to o
  - Place the initial node in the list
  - If the initial state is the goal
    - You finished
  - If the list is empty
    - There is no solution
  - If not
    - Select the node of the list with the minimum FEV
  - If the selected node is the goal
    - You finished
  - If not
    - Expands the selected node
    - Calculates FEV for each successor = **heuristic estimation**
    - Insert each successor if **not already in the list**
  - Repeat

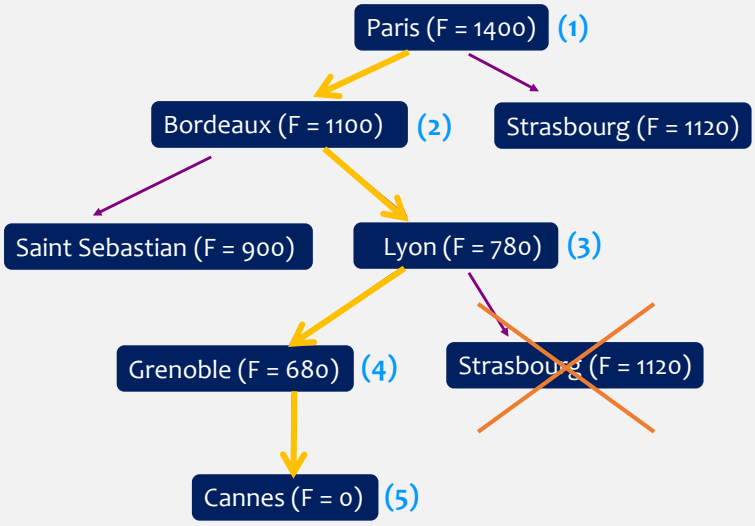
Example: Tour of Europe

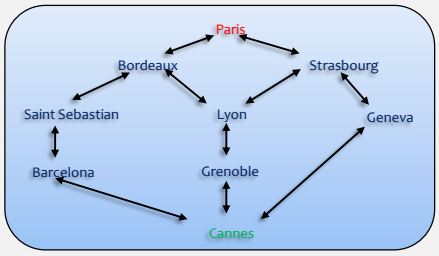


heuristics = Straight distance to the goal.

City	Heur
Paris	1400
Bordeaux	1100
Saint Sebastián	900
Barcelona	480
Lyon	780
Grenoble	680
Strasburg	1120
Geneva	630

Greedy search





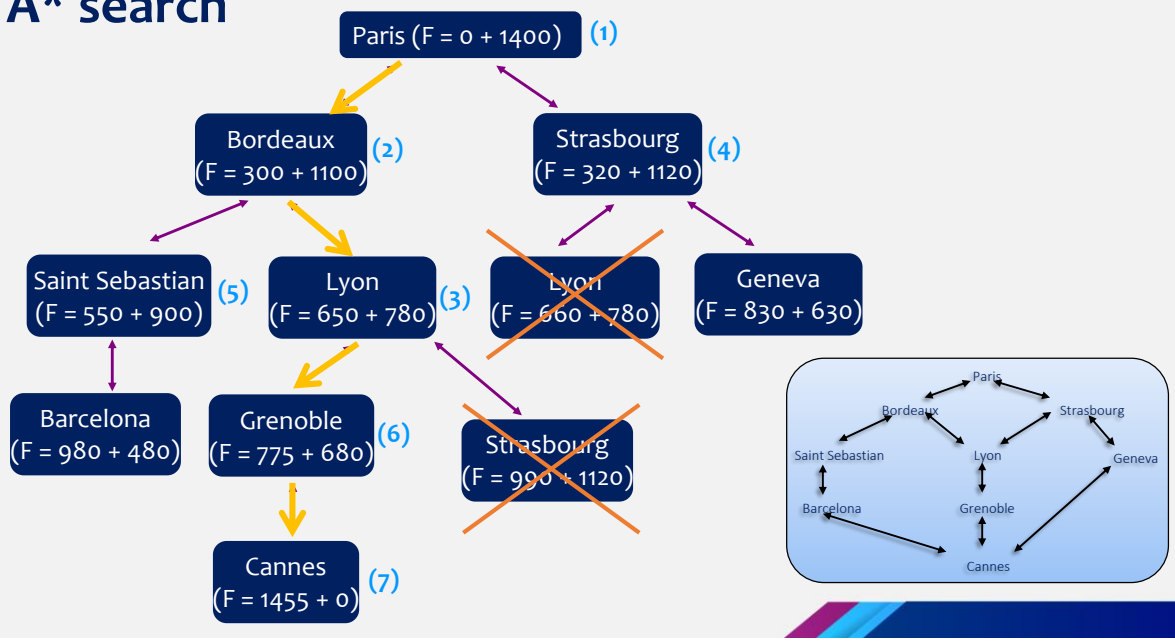
A* search

"Best-first" search where **f (n) = g (n) + h (n)**

Algorithm:

Initialize the cost of the initial node to 0
Place the initial node in the list
If the initial state is the goal
    You finished
If the list is empty
    there is no solution
If not
    Select the node from the list with minimum FEV
If the selected node is the goal
    You finished
If not
    Expands the selected node
    calculates FEV for each successor = **cumulative cost + heuristic estimation**
    Insert each successor in the list by **removing redundant nodes**
Repeat

A* search



A* search

- **f, g, and h** are estimates of the true valueS **f*, g*, and h*.**
- It requires knowledge of the cost of each movement as in the uniform cost search, to calculate g.
  - If the search is on **tree, g = g*,** where there is only one way from the initial node to the current node.
  - In general, the search can be a **graph**. In this case, **g >= g*,** therefore, it can not be less than the cost of the optimal path. This can only exceed the cost.
  - **g** can be = **g*** in a graph, if nodes are chosen appropriately.
- A* search will be **optimal in a tree** if for each node:
  - **h(n) >= 0,**
  - **h(n) <= h*(n)**, i.e., it never overestimate the distance to the closest goal state (**ADMISSIBILITY CONDITION**).

More comments about h

- If **h = h*** (*perfect heuristic estimation*), unnecessary nodes are never expanded.
- If **h = 0** then A* searches blindly as the uniform cost algorithm.
- **Goal:** Let h get as close as possible to h* without exceeding it.
  - If **h* >= h1(n) > h2(n),** h1 is more "informed" that h2.
- The admissibility condition can be relaxed for efficiency purposes, but then the solution found may not be optimal.

More comments on h

- A* search will **optimal in a graph** if in addition to the conditions of the tree search, the heuristic estimation is **consistent** (monotonic).
- A heuristic estimation is consistent if **h(n) <= c(n, a, n') + h(n')** for each node **n** and every action **a** generating successors **n'** from **n** with a cost of **c**.
- This ensures important things for optimality:
  - The values of f(n) through any branch are not decreasing.
  - Each node is explored from its optimal branch
  - The heuristic is admissible.

Variations of the algorithm A *

- *Variants of A**:
  - IDA*: Iterative Deepening A*
  - RBFS: Recursive Best-First Search
  - SMA*: Simplified Memory-bounded A*
- They seek to reduce space complexity of A* while still being optimal.
- It is achieved at the cost of increasing time complexity.

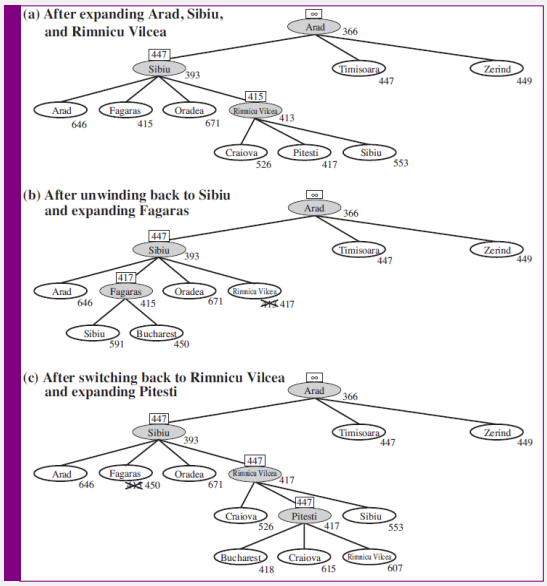
Iterative Deepening A* search (IDA*)

- In this algorithm each iteration is a “depth-limited” search as in a normal iterative deepening search.
- “Depth-limited” search is modified to use the value of the function evaluation (**g + h**) as cutting criteria rather than a depth restriction.
- The **cutting fev value** for an iteration is the smallest of any node that exceeded the cutting fev value in the previous iteration.

Recursive Best First Search

RBFS is a "best-first“ search, but using only linear space

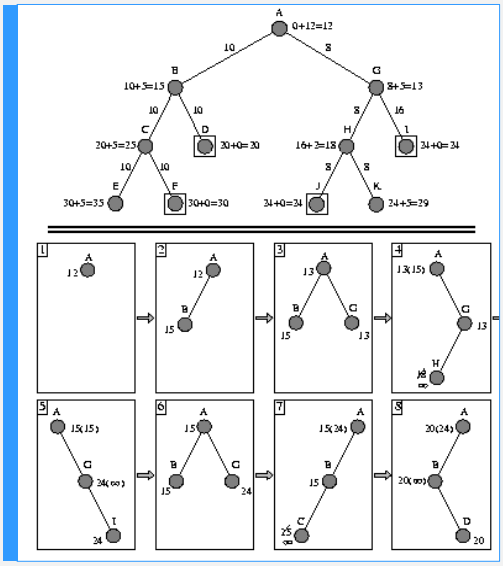
RBFS is like a "depth-first" search, but instead continuing indefinitely down on a branch, it maintains the value of the FEV of the best alternative route available from any ancestor of the current node to decide if it should abandon a branch for searching in a more promising one.



Simplified Memory-Bounded A* search (SMA*)

SMA* uses all available memory to remember the nodes where it has already searched and not have to regenerate them when necessary.

When memory is full to add new and better nodes, it eliminates the worst node, but remembers its cost by passing it to the parent of the deleted node.

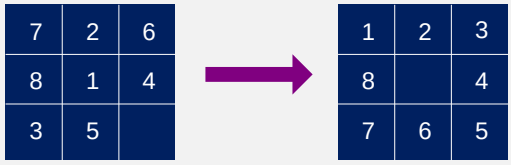


Other uses of heuristics

- To prune the search tree in other search algorithms.
- Examples:
  - If the cost of the route to a node N is greater than some threshold, delete N from the list of candidates.
  - If the path to a node N contains a number of nodes greater than some threshold, delete N from the list.
- Each pruning heuristic may lose the solution, but it could build a feasible solution by reducing the search space.

Example: heuristics for 8-puzzle

Calculate the heuristic value:



- Add the boxes that are out of position with respect to the target state.
  $h_1(n) = 1 + 0 + 1 + 0 + 1 + 0 + 1 + 1 = 5$
- Add the number of moves to bring each box to its position in the target state (Manhattan distance).
  $h_2(n) = 2 + 0 + 3 + 0 + 2 + 0 + 4 + 1 = 12$

Effectiveness of heuristics

Effective branching factor (b*)

$N + 1 = 1 + b + (b^*)^2 + \ldots + (b^*)^d$ where d is the depth of the tree.

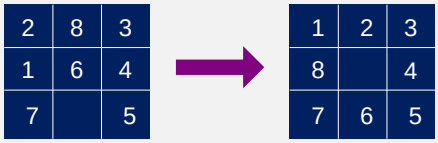
	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Figure 3.29 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.

Heuristic design strategy

(1) Find the optimal solution to a relaxed problem.

Example: 8-puzzle



- **Movement rule:** A piece can move from square A to square B if A is horizontally or vertically adjacent to B and B is empty.
- Relaxation possibilities:
  1. Move a piece from A to B if A is adjacent to B.
  2. Move a piece from A to B if B is empty.
  3. Move a piece from A to B.

It is crucial that the relaxed problem can be essentially solved without searching

Other design strategies

(2) Combine admissible heuristics.

$$h(n) = \max\{h_1(N), ..., h_m(N)\}$$

(3) Use the cost of the solution to a subproblem of the original problem.

Idea behind pattern databases (Use of the exact cost of solutions to each instance of the subproblem).

(4) To learn heuristics from experience.

Conclusions

Conclusions

- Since the uninformed search algorithms are systematic but inefficient, the concept of heuristics helps to solve problems in a better way.
- A* search is optimal and efficient if a good heuristic is chosen.
- There are variants of A* algorithm for operating with limited resources.
- There are other variants of heuristic search methods.

Tecnológico de Monterrey