Artificial Intelligence

Search II

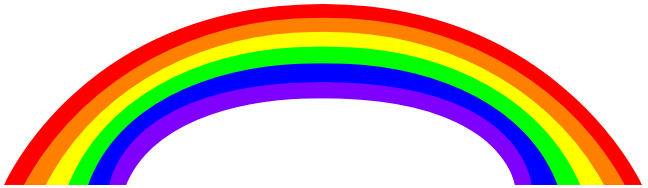
Lecture 4

(September 1, 1999)

Tralvex (Rex) Yeap MAAAI MSCS

University of Leeds

Content: Search II



- Quick Review on Lecture 3
- Constrained Satisfaction Search
- Heuristic Search Methods
- Definition of a Heuristic Function
- Best First Search
  - An example - Sliding Tiles
  - Greedy Search
  - A* Search
- Class Activity 1: DFS, BFS, BDS, UCS and GS workout for Romania Map problem
- Iterative Improvement Algorithms
  - Hill-climbing
  - Simulated Annealing
- Class Activity 2: Real-world Paper Reading - “Smart Moves - Intelligent Pathfinding”
- What’s in Store for Lecture 5

Quick Review on Lecture 3

- Introduction to Search
- Knowledge, Problem Types and Problem Formulation
- Search Strategies
  - General Search Problem
  - 4 Criteria for Evaluating Search Strategies
  - Blind Search Strategies: BFS, UCS, DFS, DLS, IDS, BDS
  - Comparing Search Strategies
- Class Activity 1: Various Blind Searches workouts for CMU Traffic problem
- Class Activity 2: Research Paper Reading: "All the Needles in a Haystack: Can Exhaustive Search Overcome Combinatorial Chaos? "

Constrained Satisfaction Search

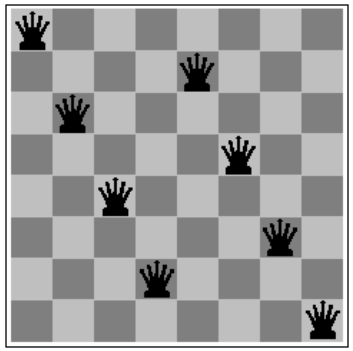
A Constrained Satisfaction Problem (**CSP**) is composed of:

1. **States** correspond to the **values** of a set of variables.
2. The **goal test** specifies a set of **constraints** that the **values must obey**.

CSP Examples: 8-queens, cryptarthmetic, VLSI layout, scheduling, design

Constrained Satisfaction Search (cont)

Example: 8-Queens Problem



- **States:** $<V_1, V_2, \ldots, V_8>$, where V_i is the row occupied by the *i*th queen.
- **Domains:** {1, 2, 3, 4, 5, 6, 7, 8}
- **Constraints:** *no-attack* constraint

  { <1,3>, <1,4>, <1,5>, … <2,4>. <2,5>, …}

  where each element specifies a pair of allowable values for variables V_i and V_j.

Constrained Satisfaction Search (cont)

Types of Constraints

- Discrete (e.g. 8-queens) versus Continuous (e.g. phase I simplex.
- Absolute constraints (violation rules out solution candidate) versus preferential constraints (e.g. goal programming).

Other better search strategies

- Backtracking search, forward checking, arc consistency checking.
- constraint propagation.

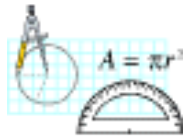
Heuristic Search Methods

- **Blind search** techniques uses **no information** that may be available about the **structure of the tree** or **availability of goals**, or any other information that may help optimise the search.
- They simply **trudge through** the search space until a solution is found.
- Most real world AI problems are susceptible to **combinatorial explosion**.
- A **heuristic search** makes use of **available information** in making the search **more efficient**.

Heuristic Search Methods (cont)

- It uses heuristics, or **rules-of-thumb** to help decide which parts of a tree to examine.
- A heuristic is a **rule** or **method** that **almost always improves** the decision process.
- For example, if in a **shop with several checkouts**, it is **usually best** to go to the one with the **shortest queue**. This holds true in most cases but further information could sway this - **(1)** if you saw that there was a checkout with only one person in that queue but that the person currently at the checkout had three trolleys full of shopping and **(2)** that at the fast-checkout all the people had only one item, you may choose to go to the fast-checkout instead. Also, **(3)** you don't go to a checkout that doesn't have a cashier - it may have the shortest queue but you'll never get anywhere.

Definition of a Heuristic Function



- A heuristic function ***h* : Ψ --> R**, where Ψ is a set of all states and **R** is a set of real numbers, maps each state **s** in the state space Ψ into a measurement ***h(s)*** which is an ***estimate* of the relative cost / benefit of extending the partial path** through ***s***.

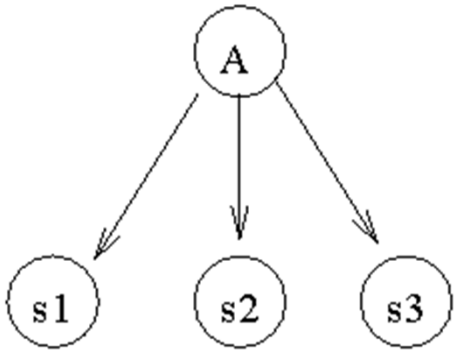


Figure 5.1 Example Graph

- Node A has 3 children.
- *h*(s1)=0.8, *h*(s2)=2.0, *h*(s3)=1.6
- The **value** refers to the **cost involved** for an action. A continual based on H(s1) is 'heuristically' the best.

Best First Search

- A **combination** of depth first (**DFS**) and breadth first search (**BFS**).
- DFS is good because a solution can be found **without computing all nodes** and BFS is good because it **does not get trapped in dead ends**.
- The Best First Search (BestFS) allows us to **switch** between paths thus gaining the **benefits of both approaches**.

Best First Search - An example

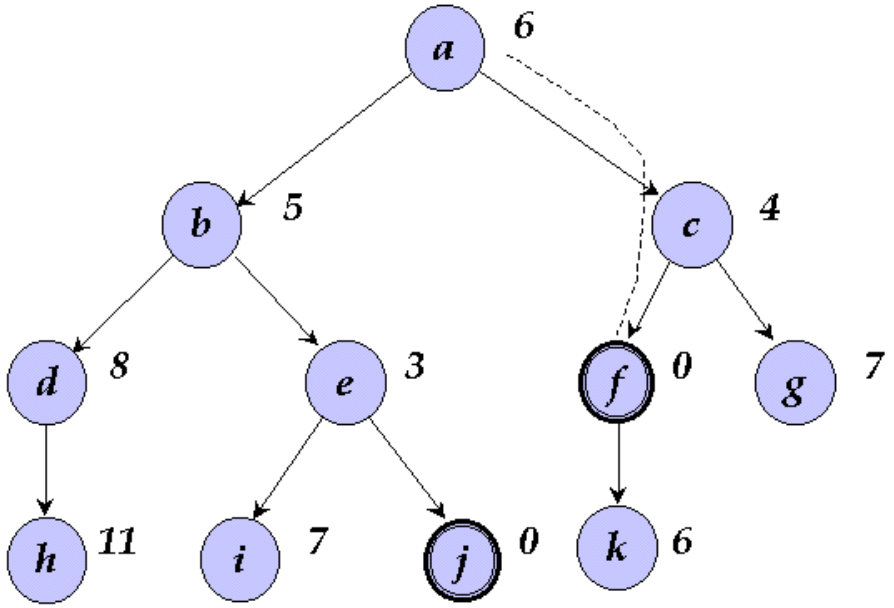


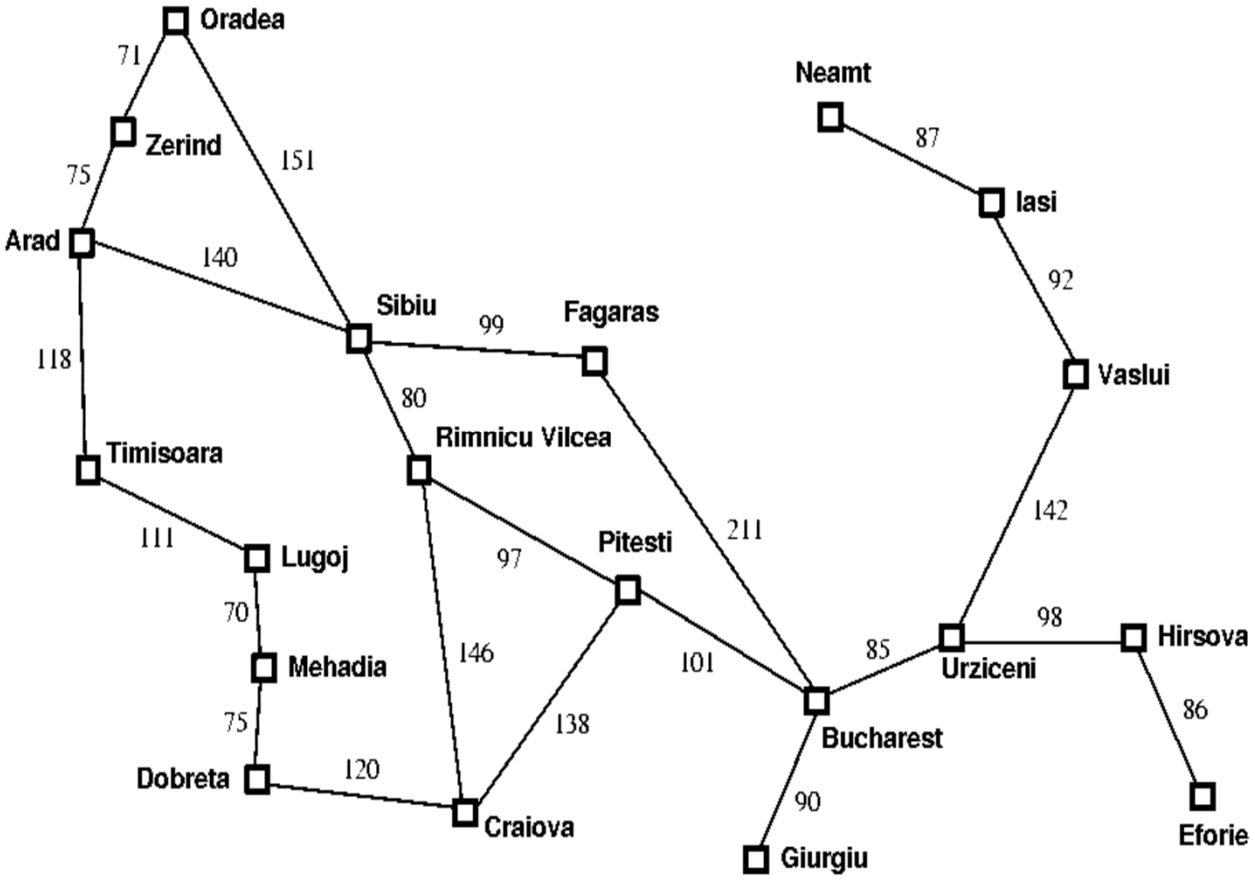
Figure 5.2 A sliding tile Search Tree using BestFS

- For sliding tiles problem, one suitable function is the **number of tiles in the correct position**.

BestFS1: Greedy Search

- This is one of the **simplest** BestFS strategy.
- Heuristic function: *h*(n) - prediction of *path cost left* to the goal.
- Greedy Search: "*To minimize the estimated cost **to reach** the goal*".
- The node whose **state** is judged to be **closest to the goal state** is always expanded first.
- Two route-finding methods **(1)** Straight line distance; **(2)** minimum Manhattan Distance - movements constrained to horizontal and vertical directions.

BestFS1: Greedy Search (cont)



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

Figure 5.3 Map of Romania with road distances in km, and straight-line distances to Bucharest.
h_{SLD}(n) = straight-line distance between n and the goal location.

BestFS1: Greedy Search (cont)

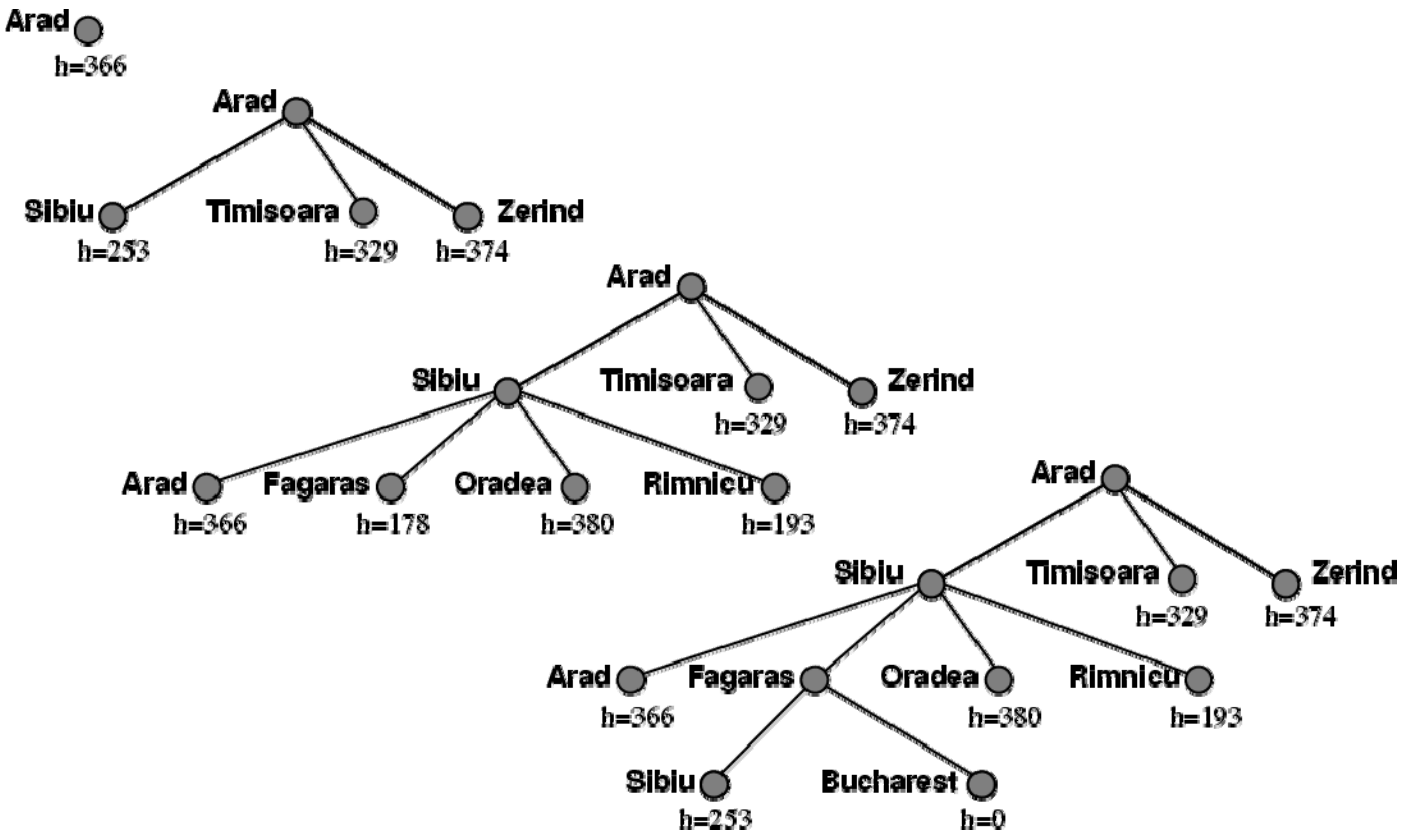


Figure 5.4 Stages in a greedy search for Bucharest, using the straight-line distance to Bucharest as the heuristics function *hSLD*. Nodes are labelled with their *h-values*.

BestFS1: Greedy Search (cont)

- Noticed that the solution for ***A → S → F → B*** **is not optimum**. It is 32 miles longer than the **optimal path** ***A → S → R → P → B***.
- The strategy prefers to take the **biggest bite** possible out of the remaining cost to reach the goal, without worrying whether this is the best in the long run - hence the name 'greedy search'.
- Greed is one of the 7 deadly sins, but it turns out that GS **perform quite well** though **not always optimal**.
- GS is susceptible to **false start**. Consider the case of going from *Iasi* to *Fagaras*. *Neamt* will be consider before *Vaului* even though it is a dead end.

BestFS1: Greedy Search (cont)

- GS **resembles DFS** in the way it prefers to follow a single path all the way to the goal, but will back up when it hits a dead end.
- Thus suffering the same defects as DFS **- not optimal** and is **incomplete**.
- The worst-case time complexity for GS is $O(b^m)$, where m is the max depth of the search space.
- **With good heuristic function**, the space and time **complexity can be reduced substantially**.
- The amount of reduction depends on the **particular problem** and the **quality of *h* function**.

BestFS2: A* Search

- **GS minimize the estimate cost to the goal, *h(n)*,** thereby cuts the search cost considerably **-** but it is **not optimal** and **incomplete**.

- **UCS minimize the cost of the path so far, *g(n)*** and is **optimal** and **complete** but can be **very inefficient**.

- A* Search **combines both GS** *h(n)* and **UCS** *g(n)* to give *f(n)* which estimated cost of the **cheapest solution through n**, ie ***f(n) = g(n) + h(n)***.

BestFS2: A* Search (cont)

- *h(n)* must be a **admissible solution**, ie. It **never overestimates** the actual cost of the best solution through n.
- Also observe that any path from the root, the f-cost never decrease (**Monotone heuristic**).
- Among optimal algorithms of this type - algorithms that extend search paths from the root - **A* is optimally efficient for any given heuristics function.**

BestFS2: A* Search (cont)

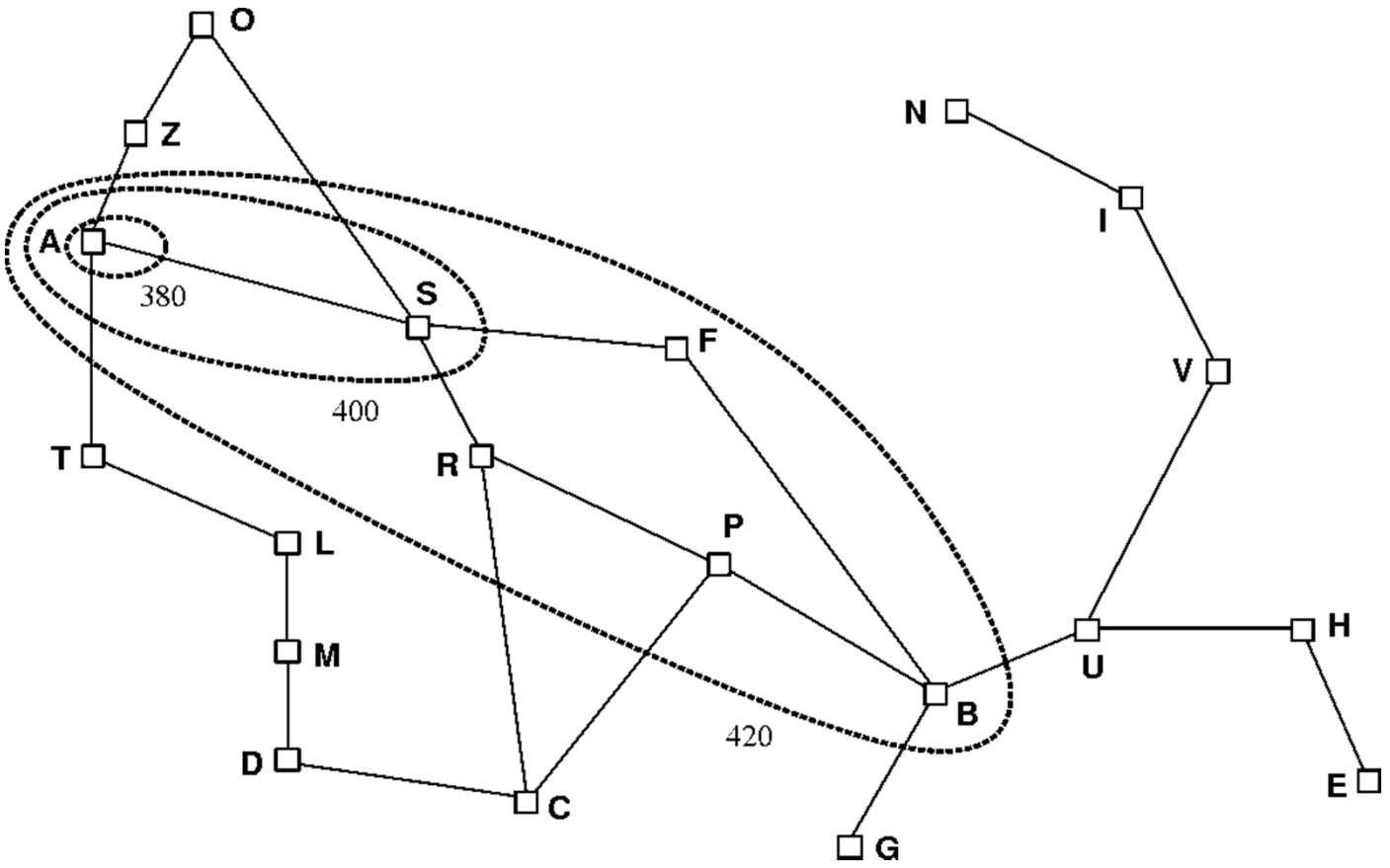


Figure 5.6 Map of Romania showing contours at *f*=380, *f*=400 and *f*=420, with Arad as the start state. Nodes inside a given contour have *f*-costs lower than the contour value.

Heuristics for an 8-puzzle Problem

- The 8-puzzle problem was one of the **earliest heuristics search problems**.
- The **objective** of the puzzle is to **slide the tiles** horizontally or vertically into empty space **until the initial configuration matches the goal configuration**.

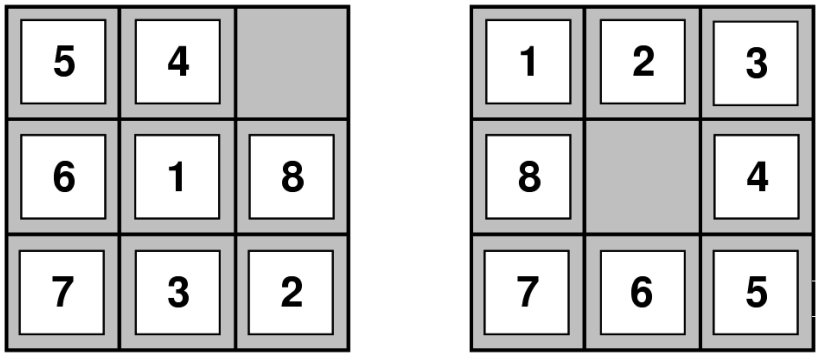


Figure 5.7 A typical instance of the 8-puzzle

Heuristics for an 8-puzzle Problem (cont)

- In total, there are a possible of 9! or **362,880 possible states**.
- However, with a good heuristic function, it is possible to reduce this state to **less than 50**.
- Two possible heuristic function **that never overestimates** the number of steps to the goal are:

1. h_1 **= the number of tiles that are in the wrong position.** In figure 5.7, none of the 8 tiles is in the goal position, so that start state would have $h_1 = 8$. h_1 is admissible heuristic, because it is clear that any tile that is out of the place must be moved at least once.

2. h_2 **= the sum of distance of the tiles from their goal positions**. Since no diagonal moves is allow, we use Manhattan distance or city block distance. h_2 is also admissible, because any move can only move 1 tile 1 step closer to the goal. The 8 tiles in the start state give a Manhattan distance of $h_2 = 2+3+3+2+4+2+0+2 = 18$

Heuristics for an 8-puzzle Problem (cont)

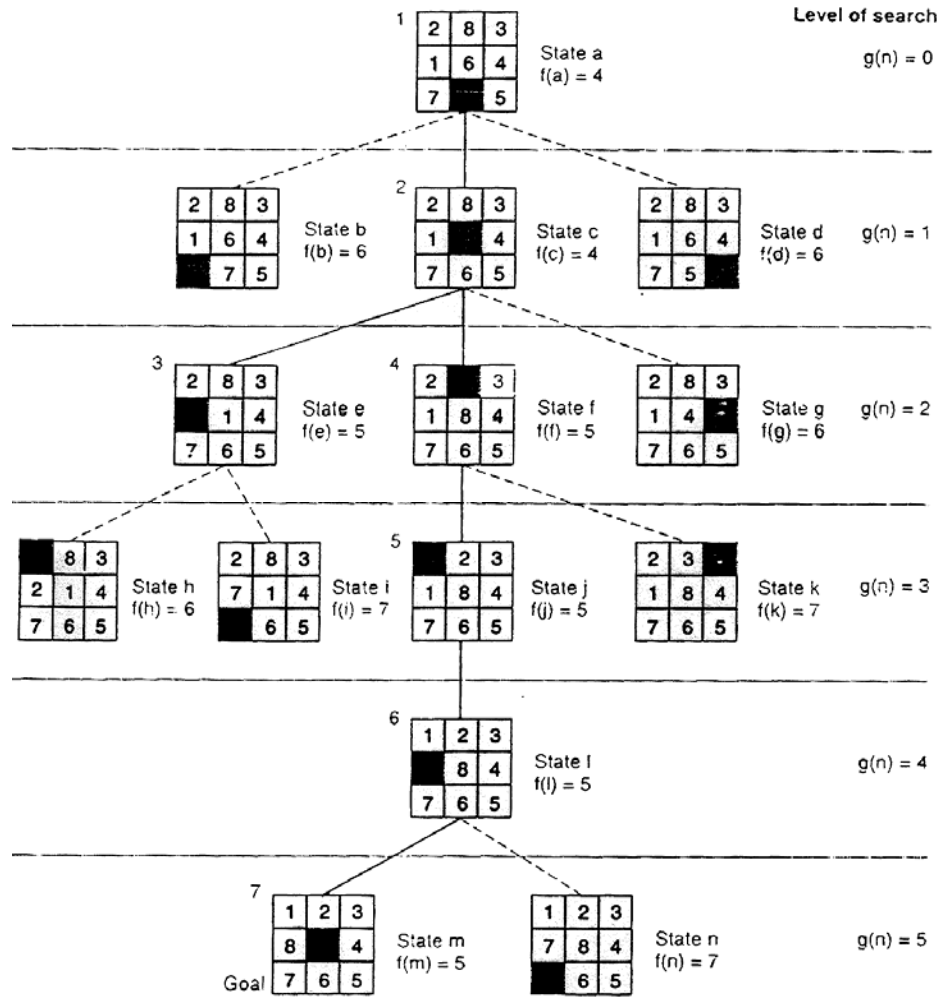


Figure 5.8 State space generated in heuristics search of a 8-puzzle

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

g(n) = **actual distance** from **start to to n** state.

h(n) = **number of tiles out of place**.

Iterative Improvement Algorithms

- Meta-heuristic algorithms for search space to find an optimal state.
- Work best on problems where each state can be **evaluated without regard to the path**.
- These algorithms are used when there are **no mathematical techniques available** and the search space is so large that exhaustive search techniques are too costly.
- Normally do not find optimal solution but **can find 'good' solutions**.
- Let f(s) be the objective function and we are require to minimize it.

Meta - things that embrace more than the usual.

IIA1: Hill-climbing

- It is simply a **loop** that **continually moves** in the direction of **increasing value**.
- **No search tree** is maintained.
- One important refinement is that when there is **more than one best successor** to choose from, the algorithm can select among them at **random**.

IIA1: Hill-climbing (cont)

This simple policy has three well-known drawbacks:

1. **Local Maxima**: a local maximum as opposed to global maximum.

2. **Plateaus**: An area of the search space where evaluation function is flat, thus requiring random walk.

3. **Ridge**: Where there are steep slopes and the search direction is not towards the top but towards the side.

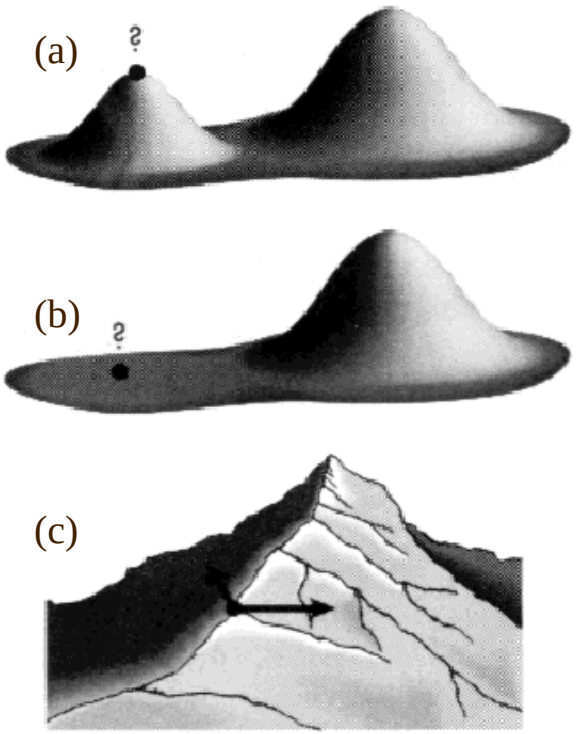


Figure 5.9 Local maxima, Plateaus and ridge situation for Hill Climbing

IIA1: Hill-climbing (cont)

- In each of the previous cases (local maxima, plateaus & ridge), the algorithm reaches a point at which no progress is being made.

- A solution is to do a **random-restart hill-climbing** - where random initial states are generated, running each until it halts or makes no discernible progress. The best result is then chosen.

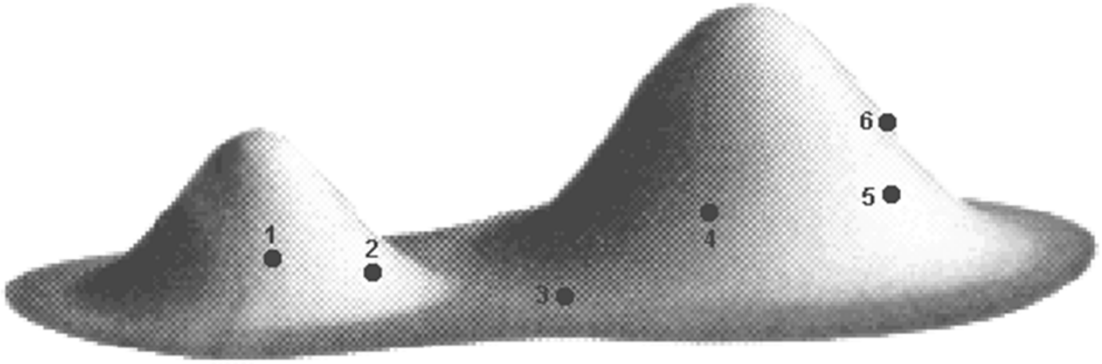


Figure 5.10 Random-restart hill-climbing (6 initial values) for 5.9(a)

IIA2: Simulated Annealing

- A alternative to a random-restart hill-climbing when stuck on a local maximum is to do a '**reverse walk'** to **escape the local maximum**.
- This is the idea of simulated annealing.
- The term simulated annealing derives from the roughly analogous physical process of **heating** and then **slowly cooling** a substance to obtain a strong crystalline structure.
- The simulated annealing process lowers the temperature by slow stages until the system ``freezes'' and no further changes occur.

IIA2: Simulated Annealing (cont)

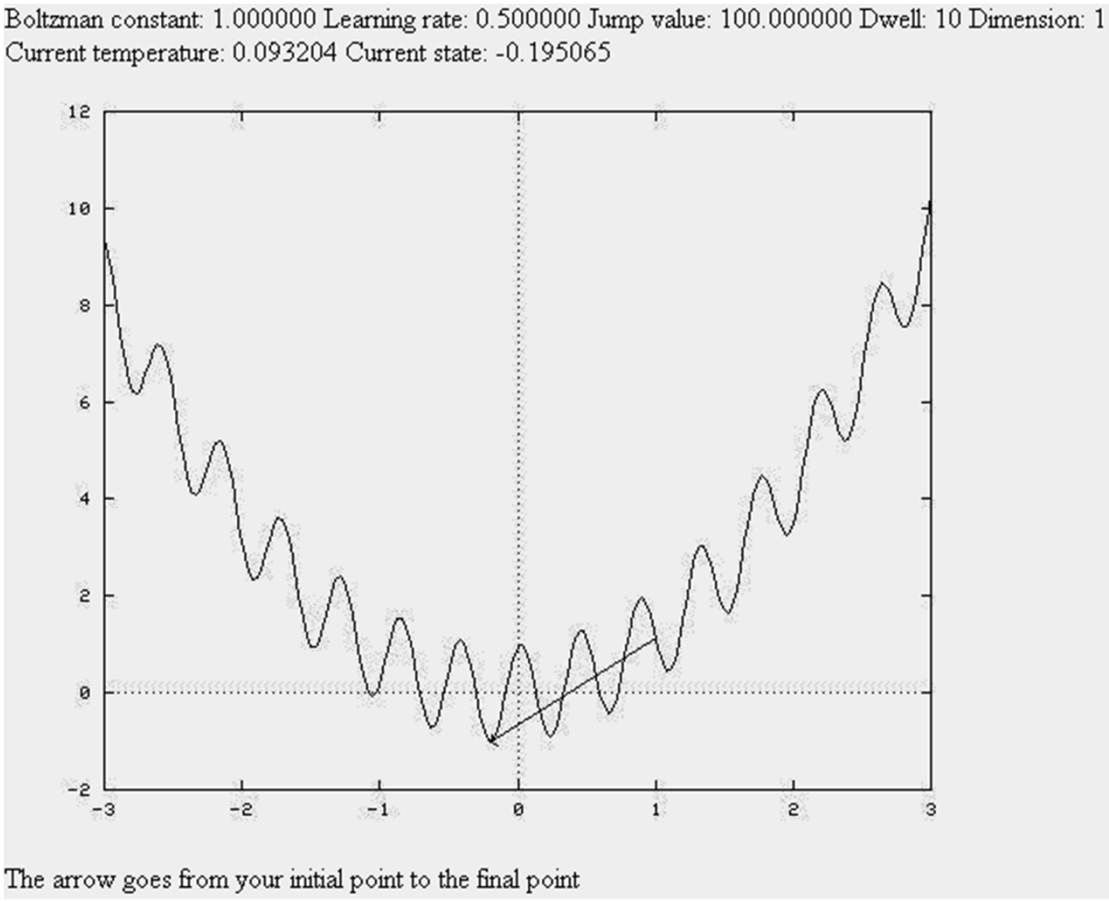
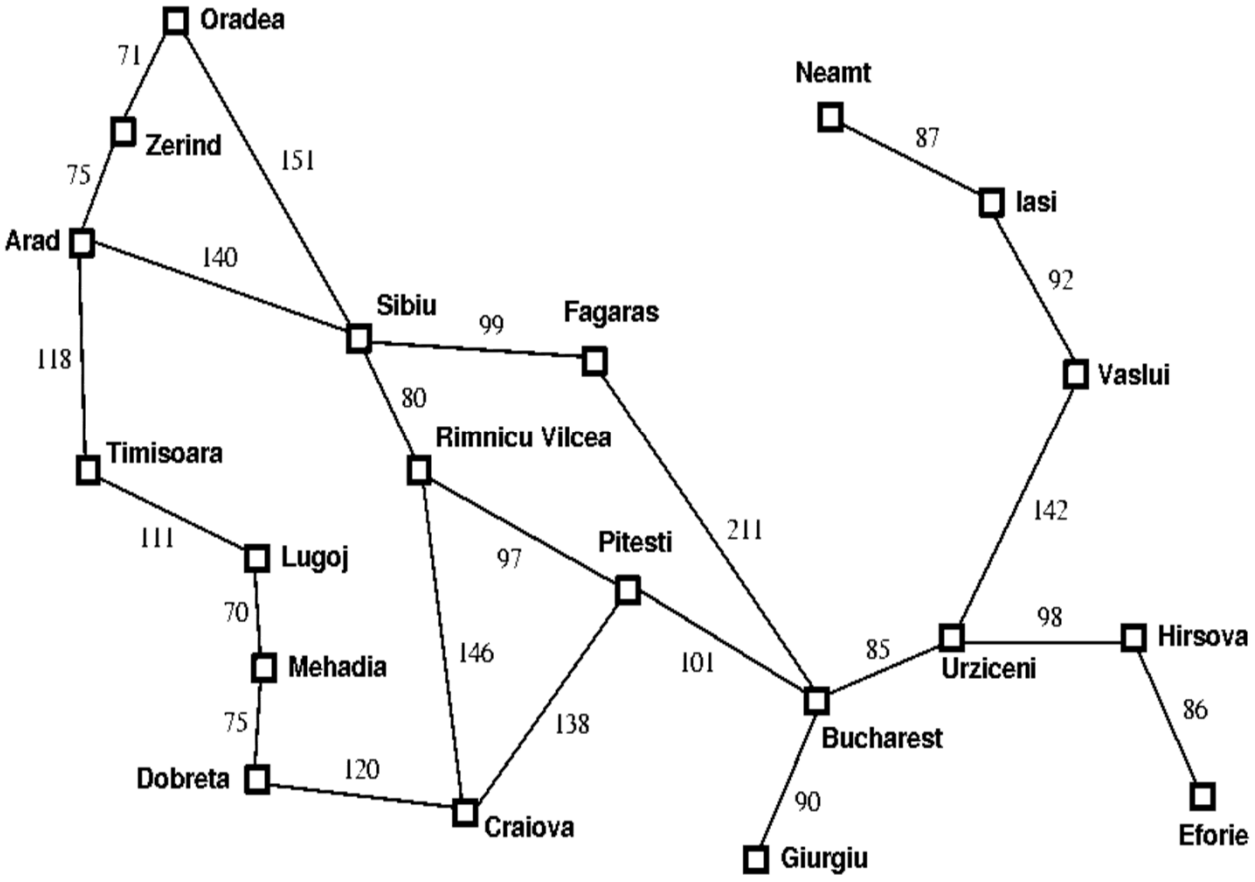


Figure 5.11 Simulated Annealing Demo (http://www.taygeta.com/annealing/demo1.html)

Class Activity 1: DFS, BFS, BDS, UCS and GS workout for Romania Map problem



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

Figure 5.12 Map of Romania with road distances in km, and straight-line distances to Bucharest.
h_{SLD}(n) = straight-line distance between n and the goal location.

Class Activity 2: Real-world Paper Reading

"Smart Moves: Intelligent Pathfinding"

- Pathfinding on the move
- Looking before you leap
- The star of the search algorithm (A* Search)
- How do I use A*
- The Limitation of A*
- Transforming the search space
- Storing it better
- Fine-tuning your search engine
- What if I'm in a smooth world

Source: http://www.gamasutra.com/features/programming/080197/pathfinding.htm

What's in Store for Lecture 5

- Game Playing
- Revision on Search I, II and III
- Class Activity 1: Practical Session on PathFinding Software by Bryan Stout (bring along your notebook/laptop)
- Class Activity 2: One paper on Heuristic Search Strategies - Reading

End of Lecture 4

Good Night.