This time: informed search

Informed search:

Use heuristics to guide the search

- Best first
- A*
- Heuristics
- Hill-climbing
- Simulated annealing

Best-first search

- Idea:

  use an evaluation function for each node; estimate of ***"desirability"***

  ⇒ expand most desirable unexpanded node.

- Implementation:

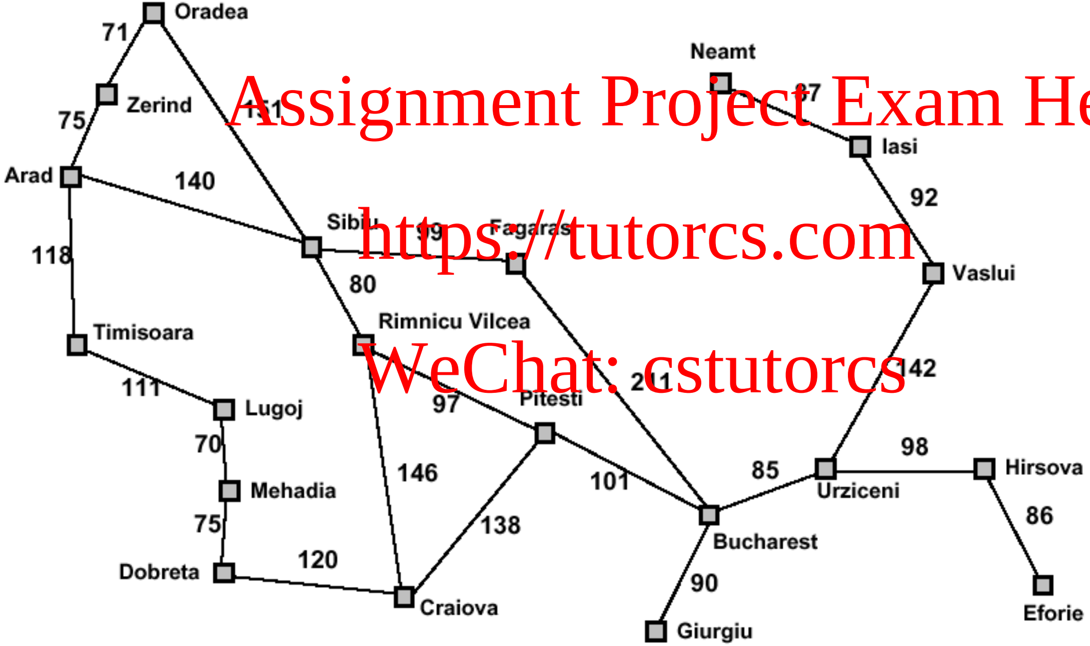
  QueueingFn = insert successors in decreasing order of desirability

- Special cases:

  greedy search

  A* search

Romania with step costs in km



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

Greedy search

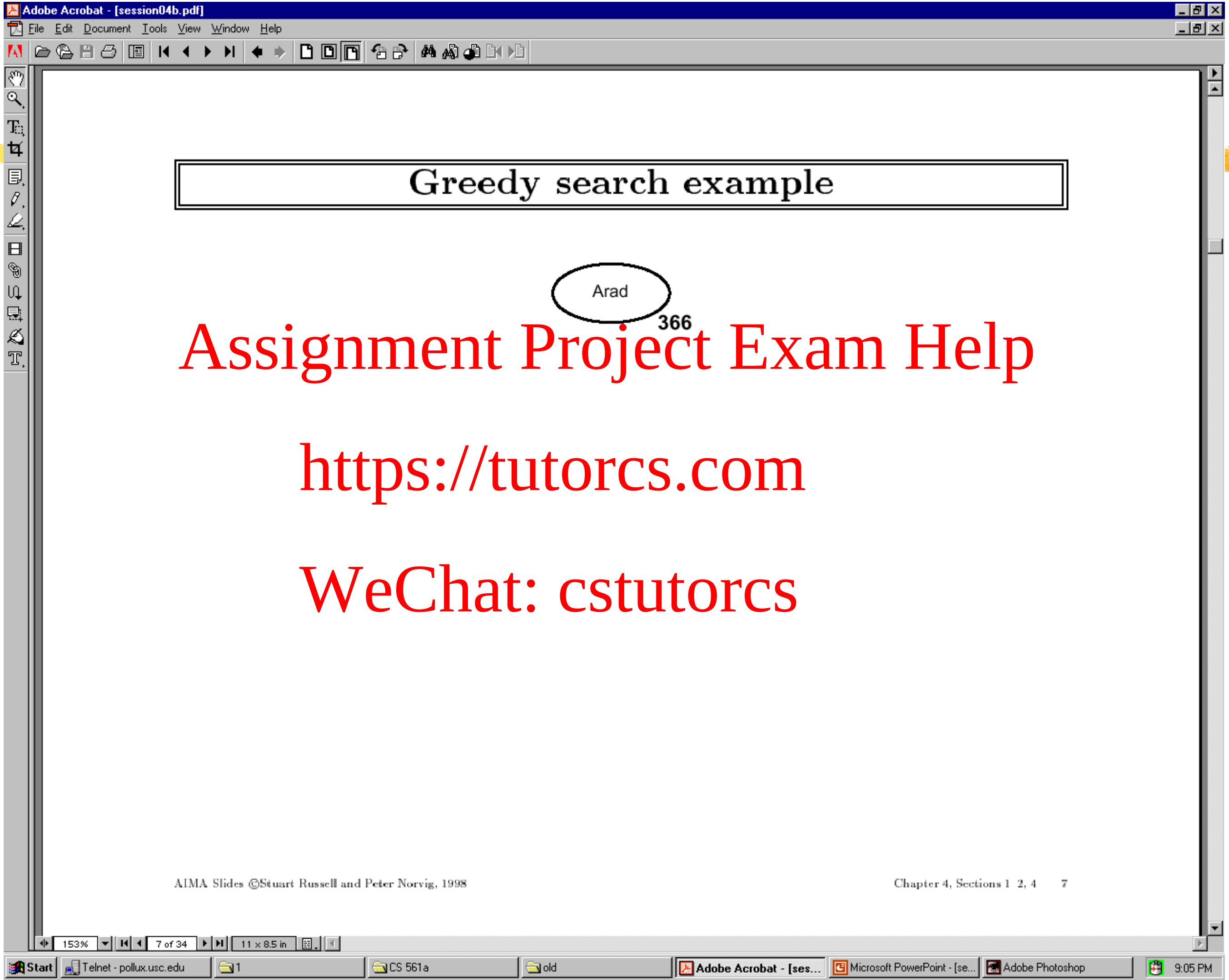
- Estimation function:

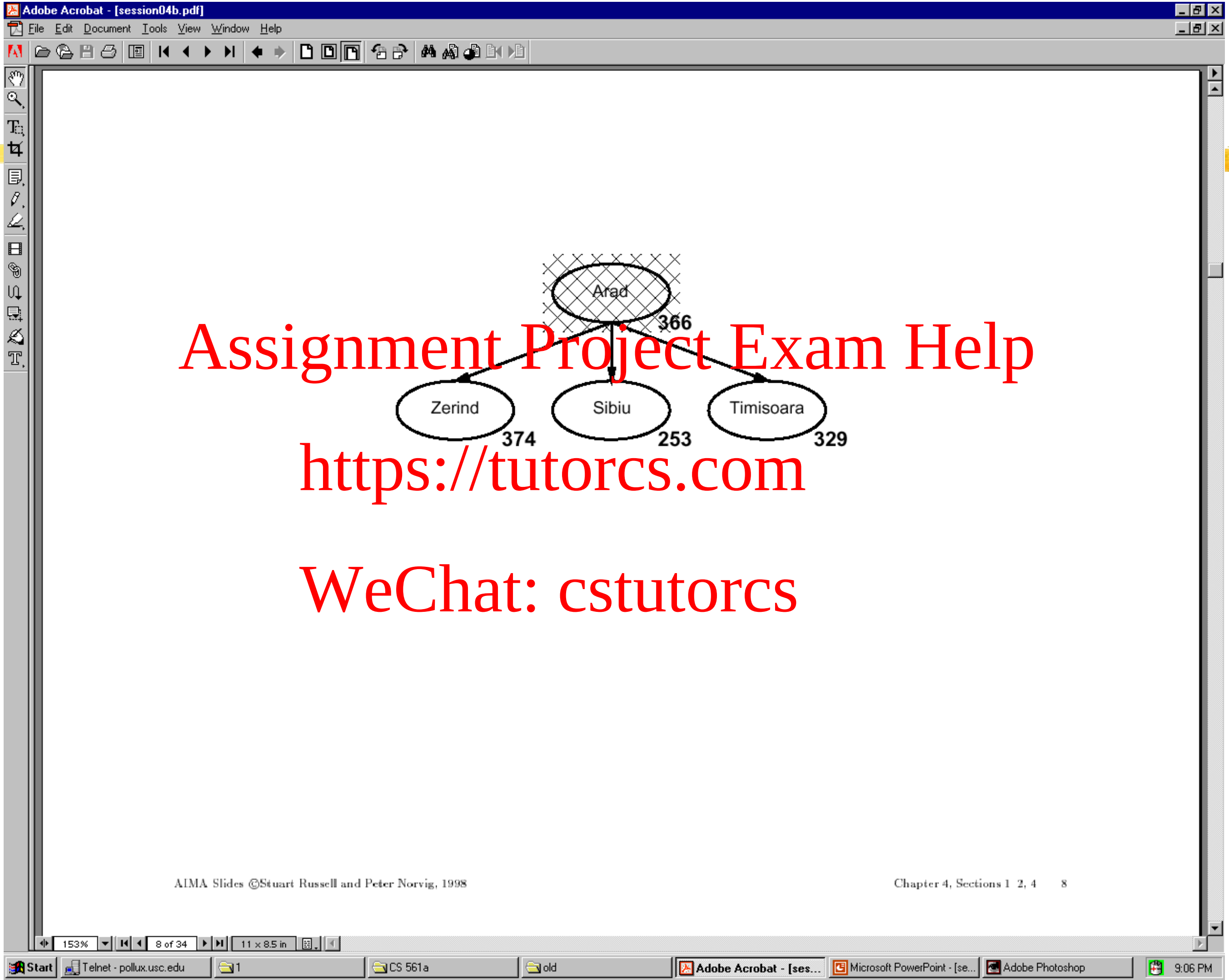
  $h(n)$ = estimate of cost from n to goal (heuristic)

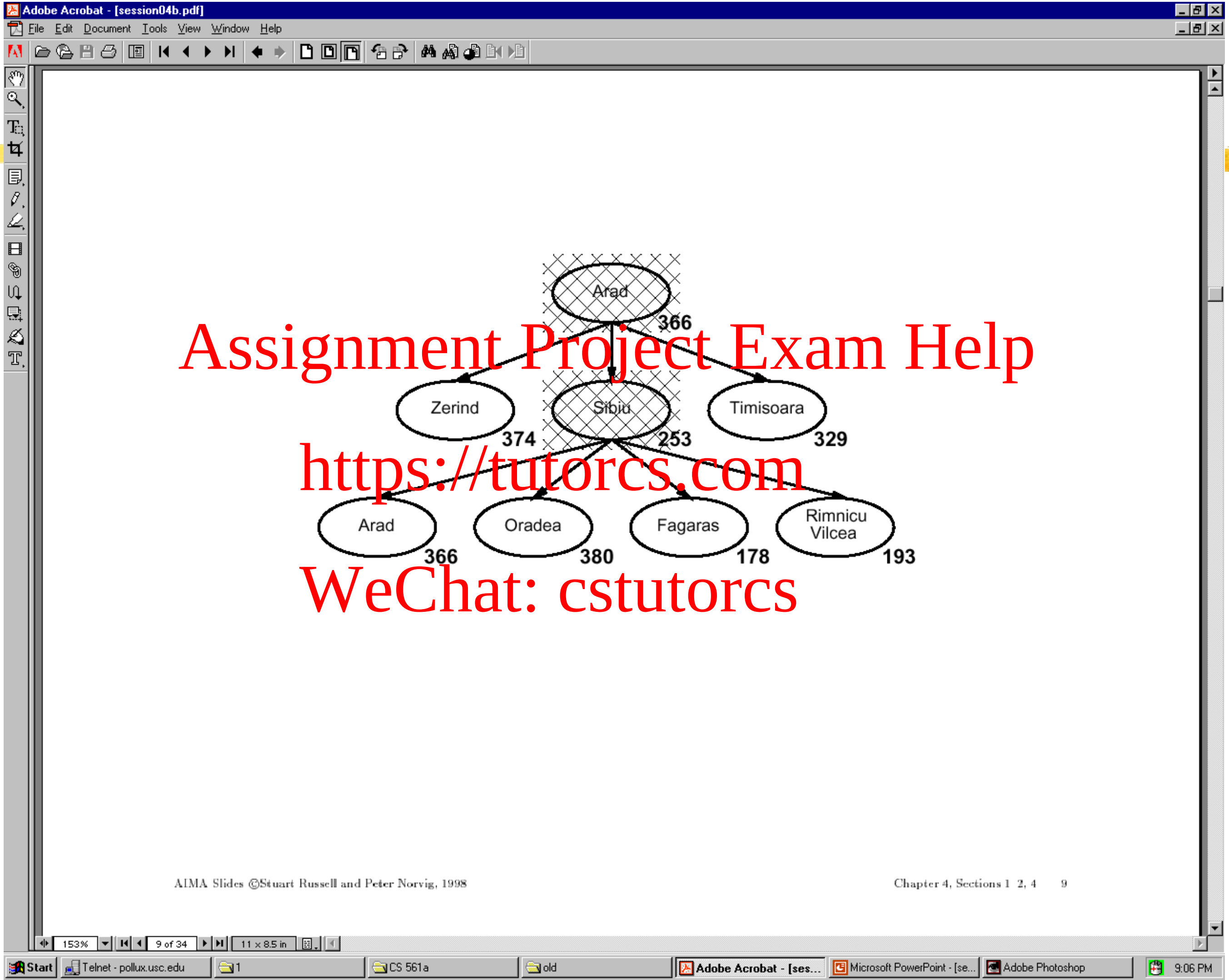
- For example:

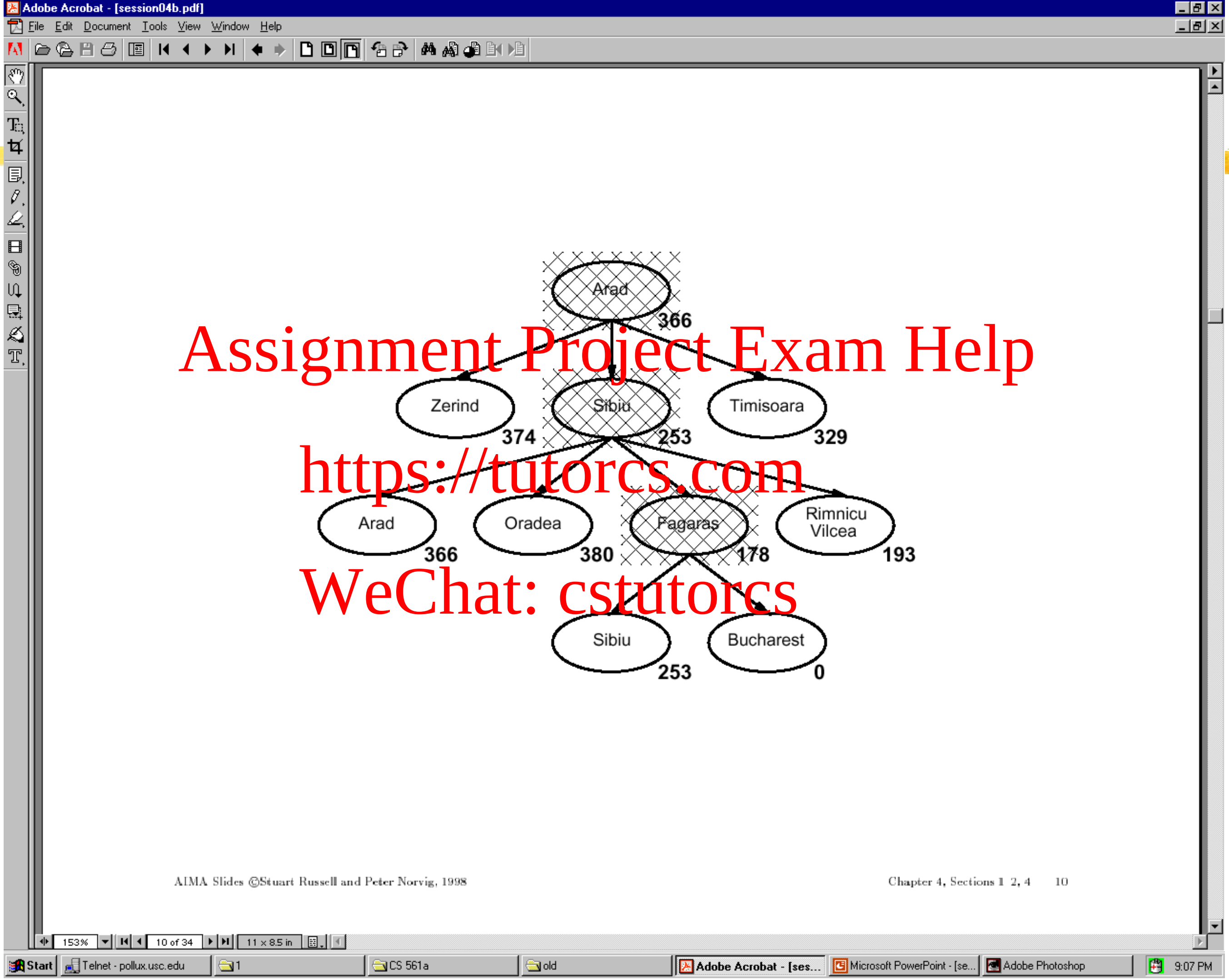
  $h_{SLD}(n)$ = straight-line distance from n to Bucharest

- Greedy search expands first the node that appears to be closest to the goal, according to $h(n)$.


Adobe Acrobat - [session04b.pdf]
File Edit Document Tools View Window Help
Greedy search example
Arad
366
AIMA Slides ©Stuart Russell and Peter Norvig, 1998
Chapter 4, Sections 1 2, 4 7
153%
7 of 34
11 x 8.5 in
Start
Telnet - pollux.usc.edu
1
CS 561a
old
Adobe Acrobat - [ses...
Microsoft PowerPoint - [se...
Adobe Photoshop
9:05 PM


Adobe Acrobat - [session04b.pdf]
File Edit Document Tools View Window Help
Arad
366
Zerind
374
Sibiu
253
Timisoara
329
AIMA Slides ©Stuart Russell and Peter Norvig, 1998
Chapter 4, Sections 1 2, 4 8
153%
8 of 34
11 x 8.5 in
Start
Telnet - pollux.usc.edu
1
CS 561a
old
Adobe Acrobat - [ses...
Microsoft PowerPoint - [se...
Adobe Photoshop
9:06 PM


Adobe Acrobat - [session04b.pdf]
File Edit Document Tools View Window Help
Arad
366
Zerind
374
Sibiu
253
Timisoara
329
Arad
366
Oradea
380
Fagaras
178
Rimnicu Vilcea
193
AIMA Slides ©Stuart Russell and Peter Norvig, 1998
Chapter 4, Sections 1 2, 4 9
153%
9 of 34
11 x 8.5 in
Start
Telnet - pollux.usc.edu
1
CS 561a
old
Adobe Acrobat - [ses...
Microsoft PowerPoint - [se...
Adobe Photoshop
9:06 PM


Adobe Acrobat - [session04b.pdf]
File Edit Document Tools View Window Help
Arad
366
Zerind
374
Sibiu
253
Timisoara
329
Arad
366
Oradea
380
Fagaras
178
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0
AIMA Slides ©Stuart Russell and Peter Norvig, 1998
Chapter 4, Sections 1 2, 4 10
153%
10 of 34
11 x 8.5 in
Start
Telnet - pollux.usc.edu
1
CS 561a
old
Adobe Acrobat - [ses...
Microsoft PowerPoint - [se...
Adobe Photoshop
9:07 PM

Properties of Greedy Search

- Complete?

- Time?

- Space?

- Optimal?

Properties of Greedy Search

- Complete? No – can get stuck in loops
  e.g., Iasi > Neamt > Iasi > Neamt > …
  Complete in finite space with repeated-state checking.

- Time? O(b^m) but a good heuristic can give dramatic improvement

- Space? O(b^m) – keeps all nodes in memory

- Optimal? No.

A* search

- Idea: avoid expanding paths that are already expensive

  evaluation function: $f(n) = g(n) + h(n)$ with:

  $g(n)$ – cost so far to reach n

  $h(n)$ – estimated cost to goal from n

  $f(n)$ – estimated total cost of path through n to goal

- A* search uses an admissible heuristic, that is,

  $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n.

  For example: $h_{SLD}(n)$ never overestimates actual road distance.

- Theorem: A* search is optimal

A* search

- A* search uses an admissible heuristic, that is,

  $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n.

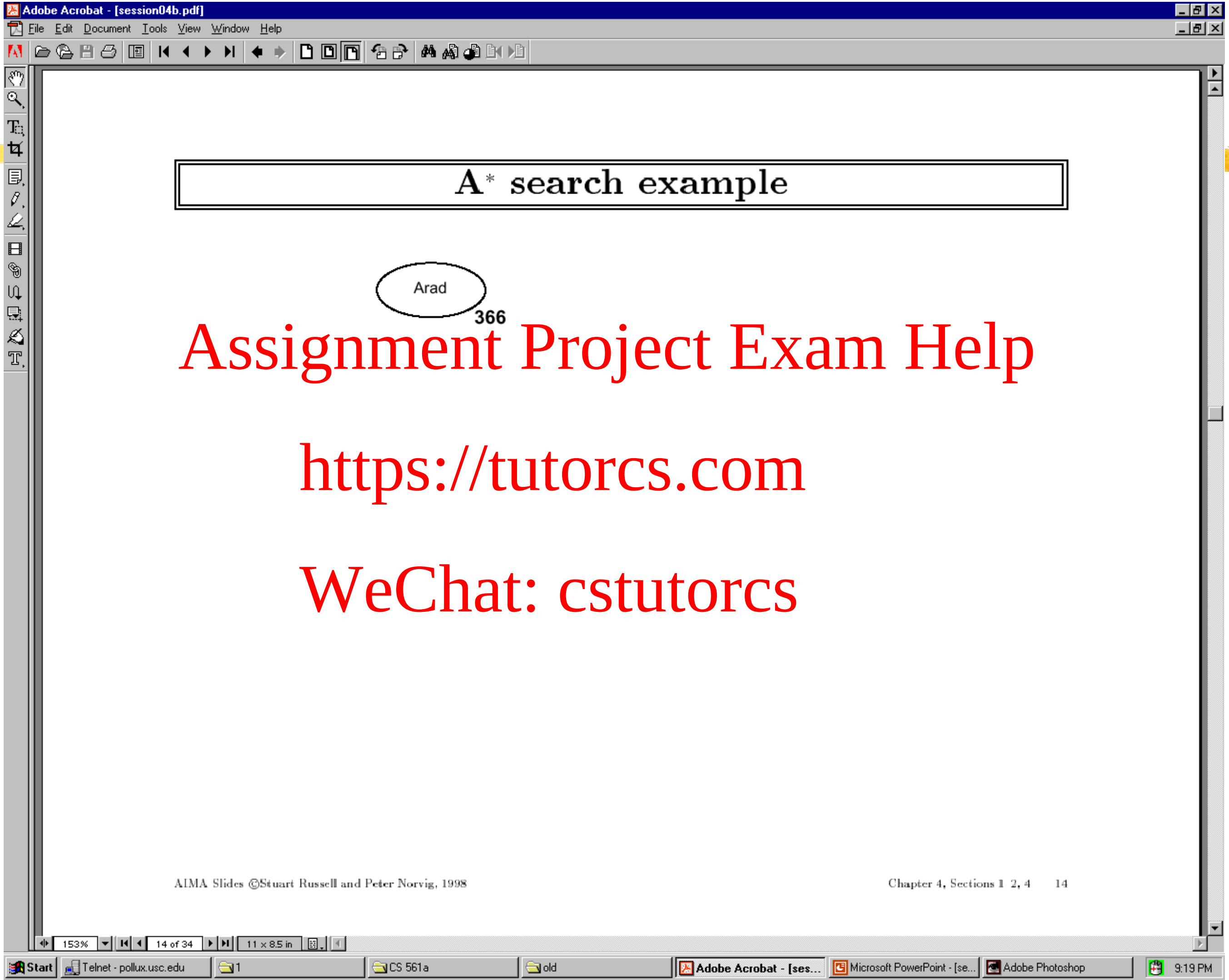
- Theorem: A* search is optimal

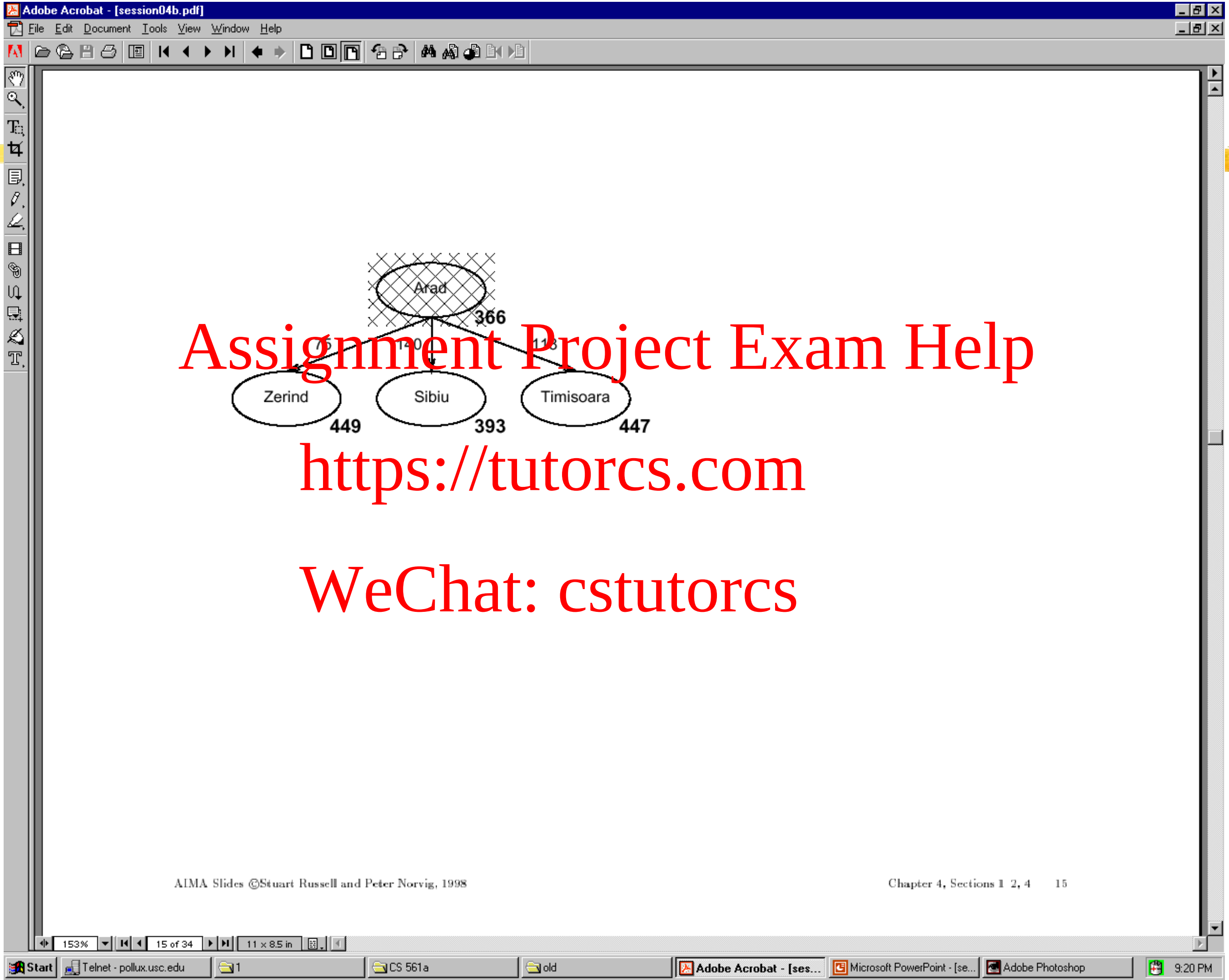
- Note: A* is also optimal if the heuristic is consistent, i.e.,

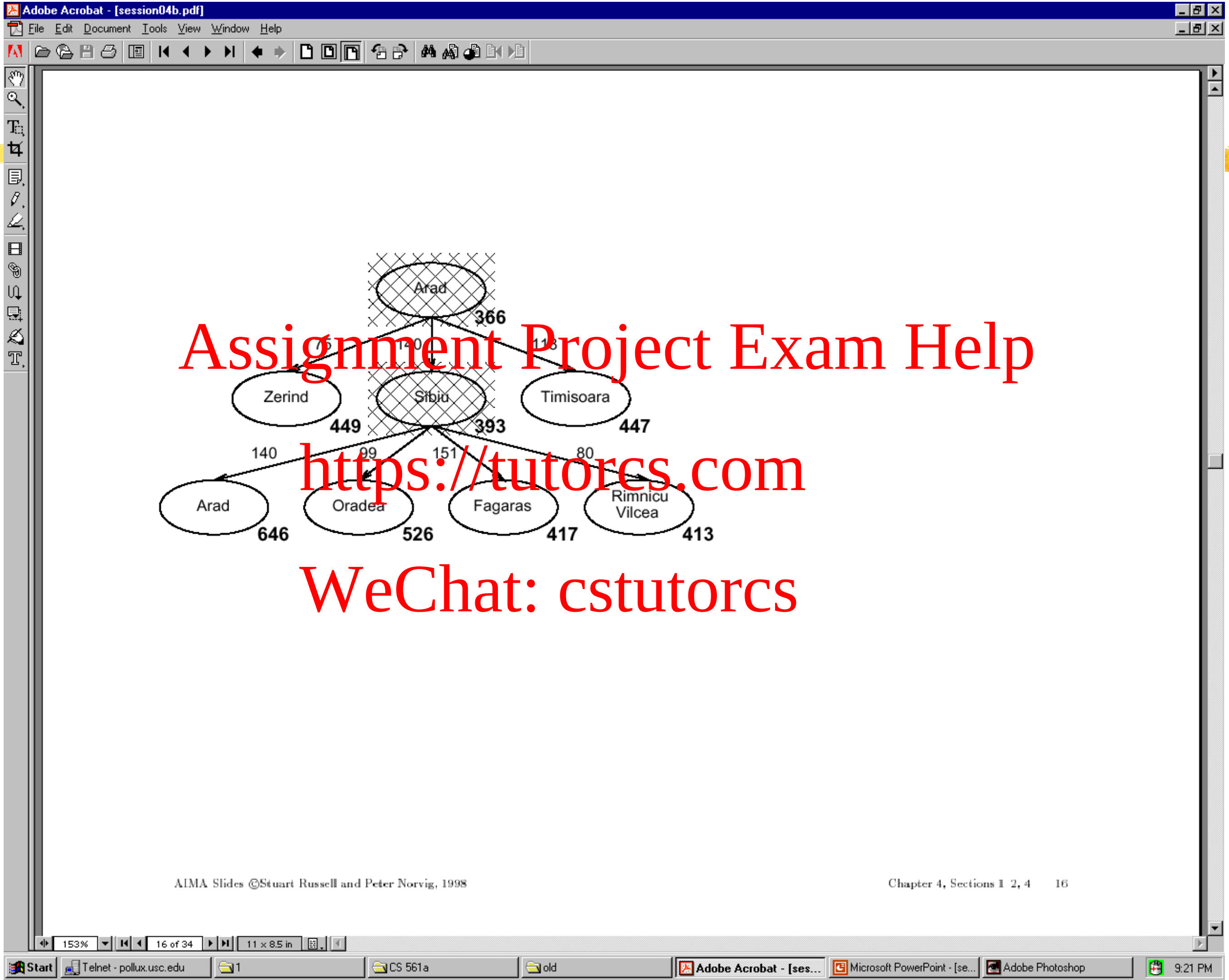
$$h(N) \leq c(N, P) + h(P) \text{ and}$$
$$h(G) = 0.$$

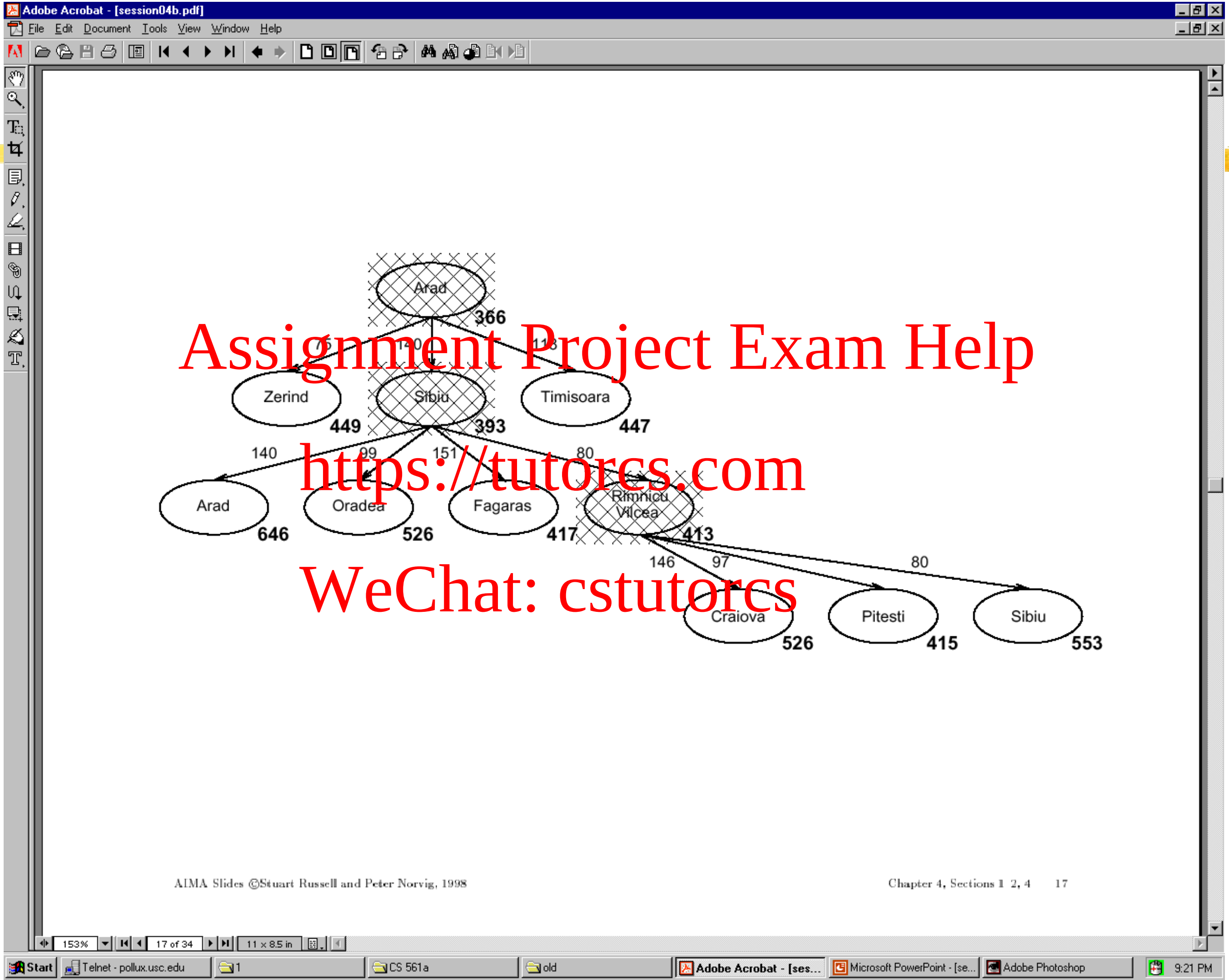
Actual cost
From N to P

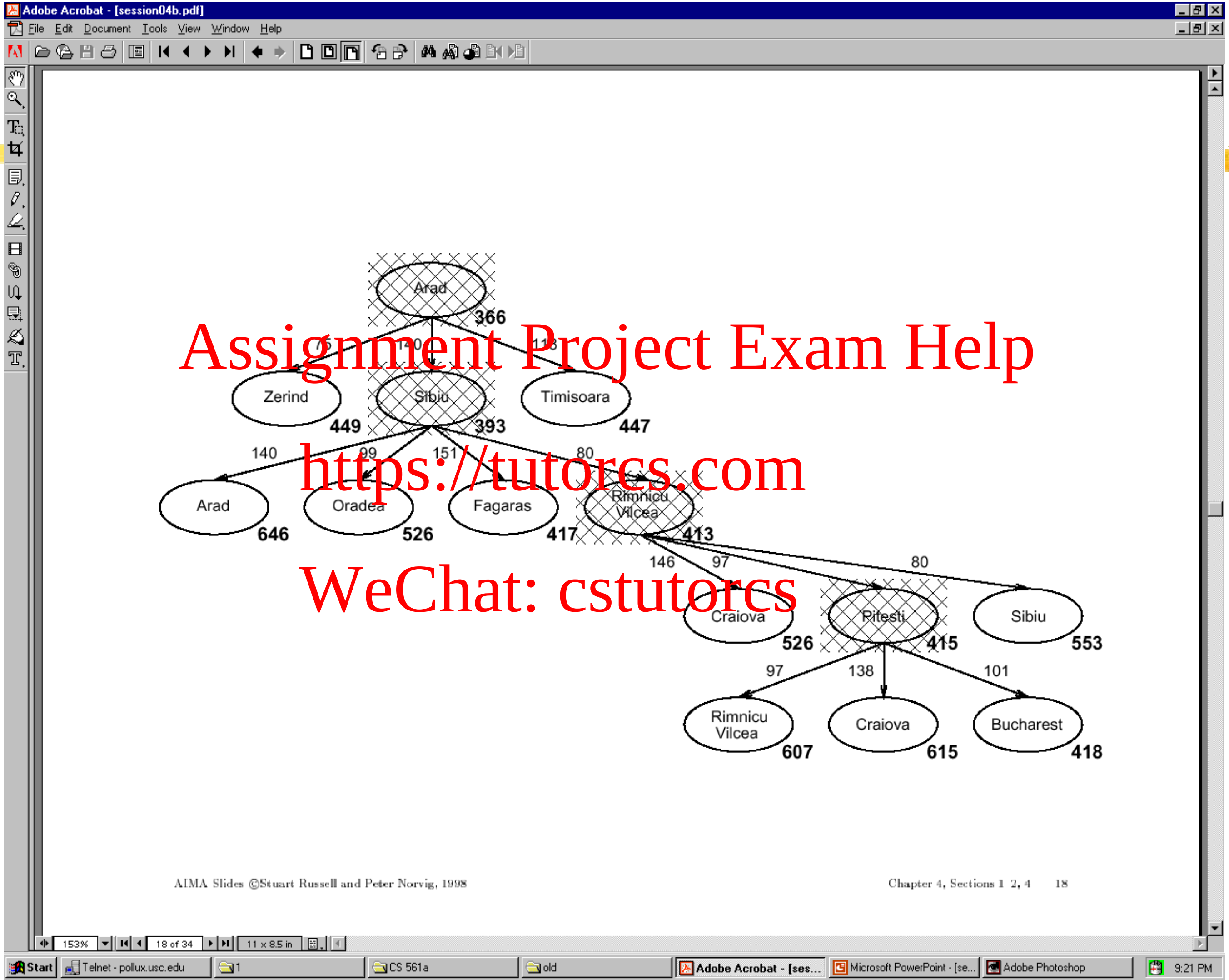
- (a consistent heuristic is admissible (by induction), but the converse is not always true)

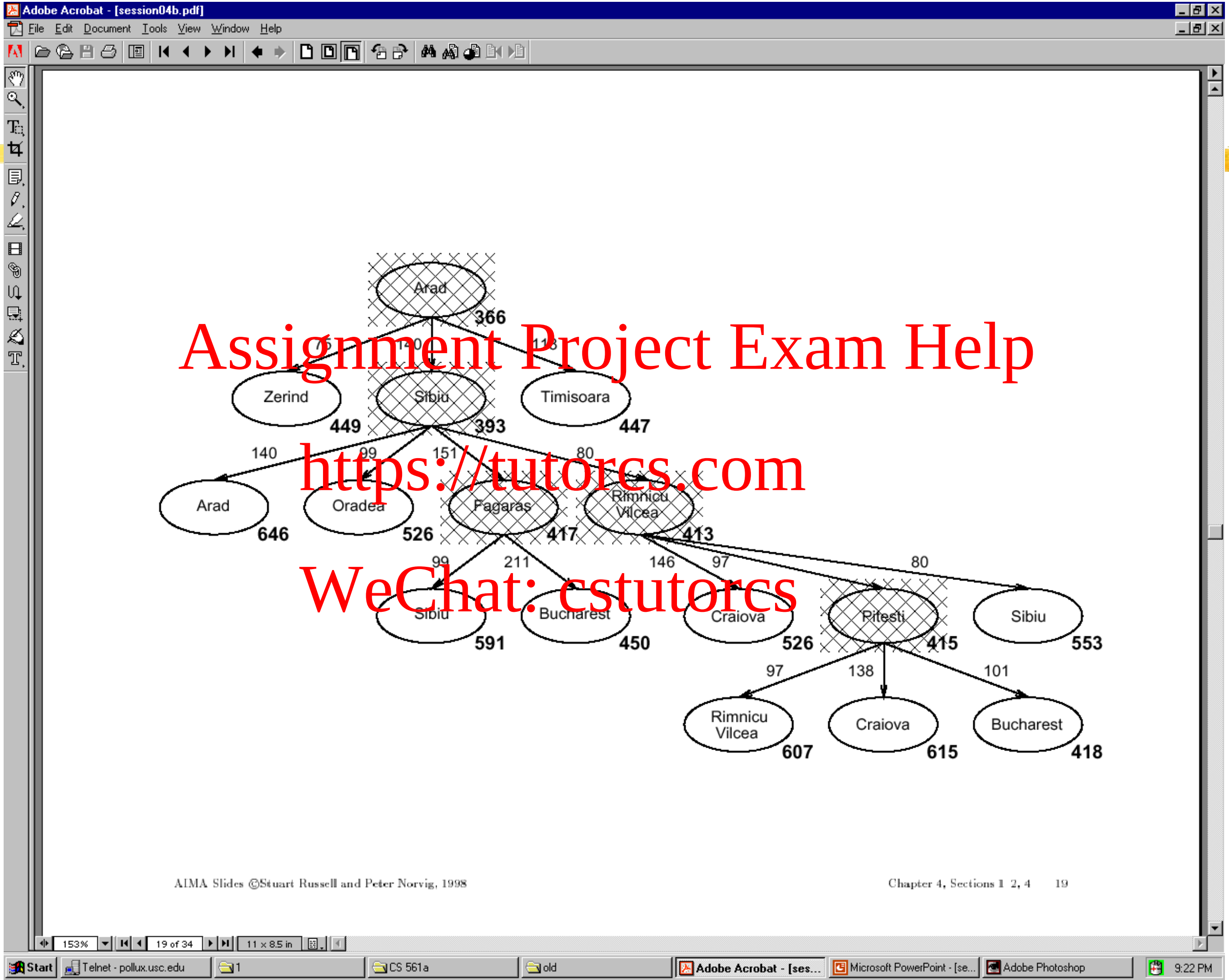

Adobe Acrobat - [session04b.pdf]
File Edit Document Tools View Window Help
A* search example
Arad
366
AIMA Slides ©Stuart Russell and Peter Norvig, 1998
Chapter 4, Sections 1 2, 4 14
153%
14 of 34
11 x 8.5 in
Start
Telnet - pollux.usc.edu
1
CS 561a
old
Adobe Acrobat - [ses...
Microsoft PowerPoint - [se...
Adobe Photoshop
9:19 PM


Adobe Acrobat - [session04b.pdf]
File Edit Document Tools View Window Help
Arad
366
140
118
Zerind
449
Sibiu
393
Timisoara
447
AIMA Slides ©Stuart Russell and Peter Norvig, 1998
Chapter 4, Sections 1 2, 4 15
153%
15 of 34
11 x 8.5 in
Start
Telnet - pollux.usc.edu
1
CS 561a
old
Adobe Acrobat - [ses...
Microsoft PowerPoint - [se...
Adobe Photoshop
9:20 PM


Adobe Acrobat - [session04b.pdf]
File Edit Document Tools View Window Help
Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu Vilcea
413
AIMA Slides ©Stuart Russell and Peter Norvig, 1998
Chapter 4, Sections 1 2, 4
16
153%
16 of 34
11 x 8.5 in
Start
Telnet - pollux.usc.edu
CS 561a
old
Adobe Acrobat - [ses...
Microsoft PowerPoint - [se...
Adobe Photoshop
9:21 PM

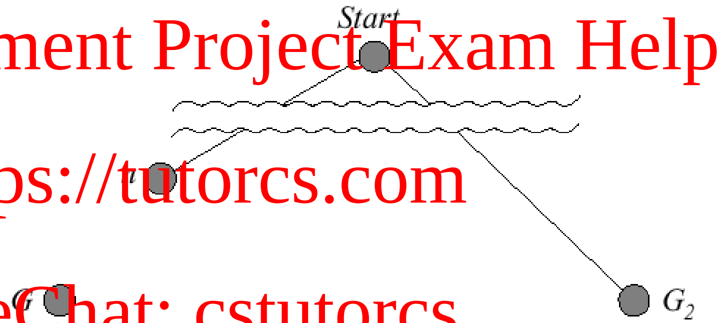

Adobe Acrobat - [session04b.pdf]
File Edit Document Tools View Window Help
Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu Vilcea
413
146
97
80
Craiova
526
Pitesti
415
Sibiu
553
AIMA Slides ©Stuart Russell and Peter Norvig, 1998
Chapter 4, Sections 1 2, 4 17
153%
17 of 34
11 x 8.5 in
Start
Telnet - pollux.usc.edu
1
CS 561a
old
Adobe Acrobat - [ses...
Microsoft PowerPoint - [se...
Adobe Photoshop
9:21 PM


Adobe Acrobat - [session04b.pdf]
File Edit Document Tools View Window Help
Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu Vilcea
413
146
97
80
Craiova
526
Pitesti
415
Sibiu
553
97
138
101
Rimnicu Vilcea
607
Craiova
615
Bucharest
418
AIMA Slides ©Stuart Russell and Peter Norvig, 1998
Chapter 4, Sections 1 2, 4 18
153%
18 of 34
11 x 8.5 in
Start
Telnet - pollux.usc.edu
CS 561a
old
Adobe Acrobat - [ses...
Microsoft PowerPoint - [se...
Adobe Photoshop
9:21 PM


Adobe Acrobat - [session04b.pdf]
File Edit Document Tools View Window Help
Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu Vilcea
413
99
211
146
97
80
Sibiu
591
Bucharest
450
Craiova
526
Pitesti
415
Sibiu
553
97
138
101
Rimnicu Vilcea
607
Craiova
615
Bucharest
418
AIMA Slides ©Stuart Russell and Peter Norvig, 1998
Chapter 4, Sections 1 2, 4 19
153%
19 of 34
11 x 8.5 in
Start
Telnet - pollux.usc.edu
CS 561a
old
Adobe Acrobat - [ses...
Microsoft PowerPoint - [se...
Adobe Photoshop
9:22 PM

Optimality of A* (standard proof)

Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node on a shortest path to an optimal goal G_1.



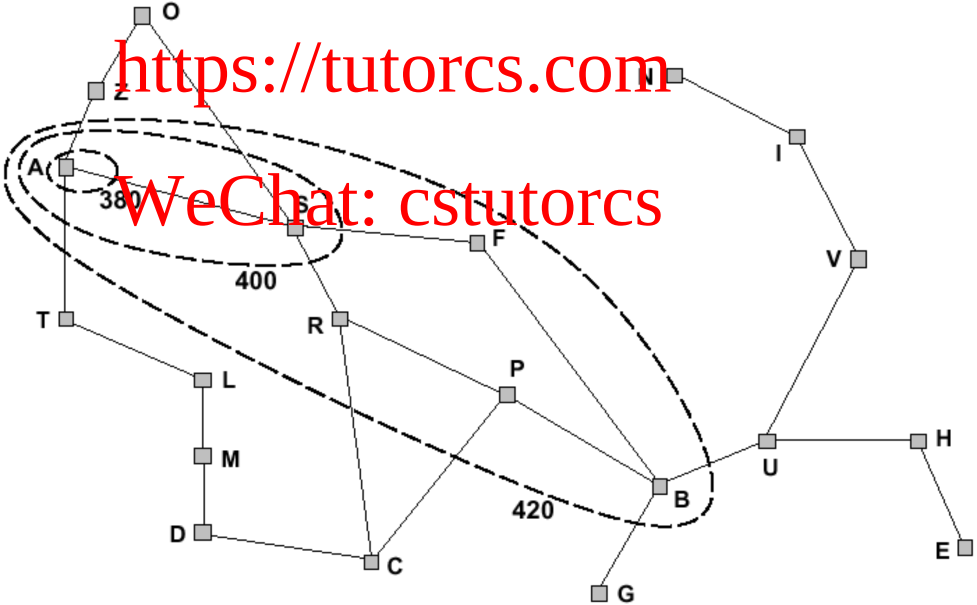
$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Optimality of A* (more useful proof)

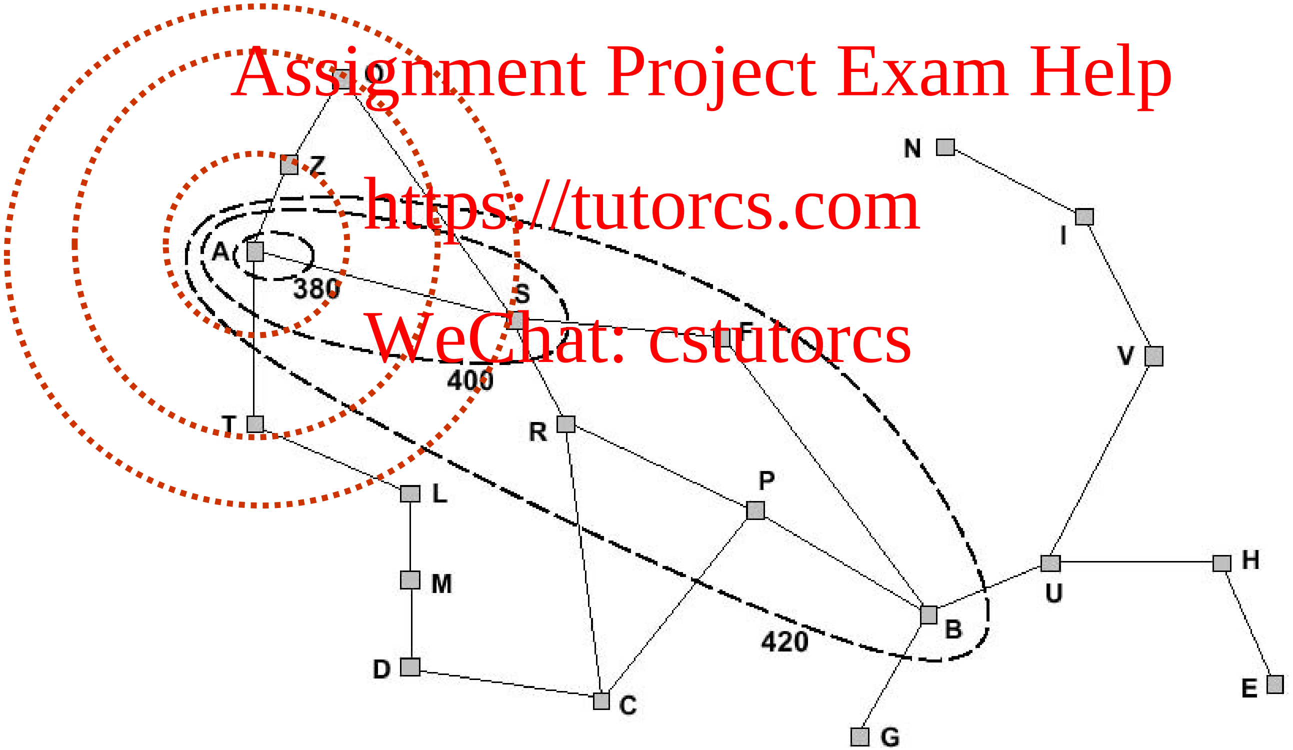
Lemma: A* expands nodes in order of increasing f value

Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)
Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



f-contours

How do the contours look like when h(n) =0?



Properties of A*

- Complete?

- Time?

- Space?

- Optimal?

Properties of A*

- Complete? Yes, unless infinitely many nodes with $f \leq f(G)$

- Time? Exponential in [(relative error in h) x (length of solution)]

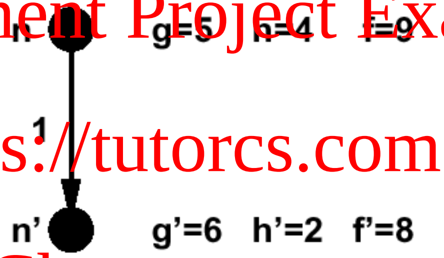
- Space? Keeps all nodes in memory

- Optimal? Yes – cannot expand f_{i+1} until f_i is finished

Proof of lemma: pathmax

For some admissible heuristics, f may *decrease* along a path

E.g., suppose n' is a successor of n



But this throws away information!
$f(n) = 9 \Rightarrow$ true cost of a path through n is ≥ 9
Hence true cost of a path through n' is ≥ 9 also

Pathmax modification to A*:
Instead of $f(n') = g(n') + h(n')$, use $f(n') = max(g(n') + h(n'), f(n))$

With pathmax, f is always nondecreasing along any path

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

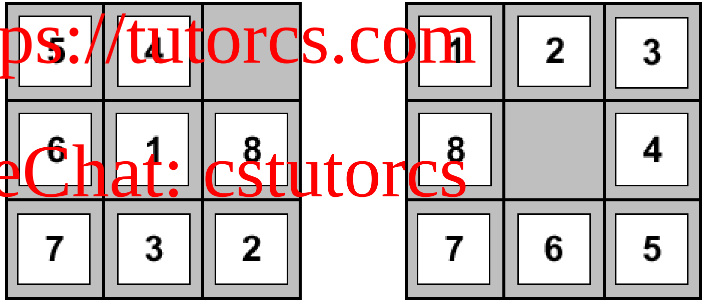
$\underline{\underline{h_1(S)\ =}}??$

$\underline{\underline{h_2(S)\ =}}??$

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



Start State Goal State

$h_1(S) =$?? 7
$h_2(S) =$?? 2+3+3+2+4+2+0+2 = 18

Relaxed Problem

- Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem.

- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.

- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.

This time

- Iterative improvement
- Hill climbing
- Simulated annealing

Iterative improvement

- In many optimization problems, path is irrelevant; the goal state itself is the solution.

- Then, state space = space of "complete" configurations. Algorithm goal:

  - find optimal configuration (e.g., TSP), or,
  - find configuration satisfying constraints (e.g., n-queens)

- In such cases, can use iterative improvement algorithms: keep a single "current" state, and try to improve it.

Iterative improvement example: vacuum world

Simplified world: 2 locations, each may or not contain dirt,
each may or not contain vacuuming agent.

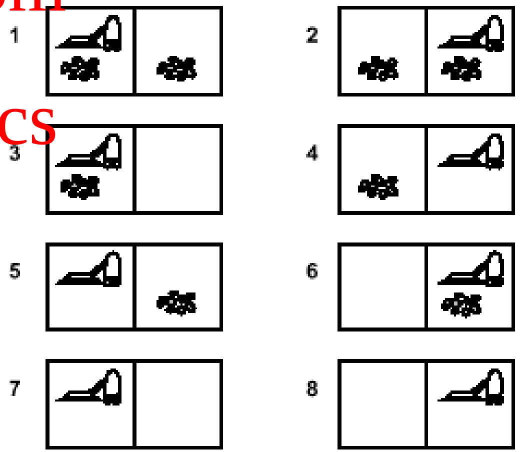
Goal of agent: clean up the dirt.

If path does not matter, do not need to keep track of it.

Single-state, start in #5. Solution??

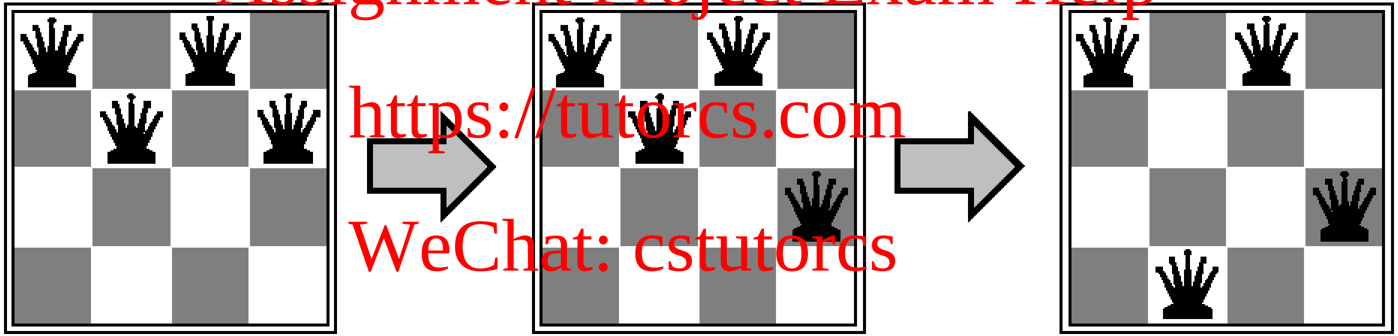
Multiple-state, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution??

Contingency, start in #5
Murphy's Law: $Suck$ can dirty a clean carpet
Local sensing: dirt, location only.
Solution??



Iterative improvement example: n-queens

- Goal: Put n chess-game queens on an n x n board, with no two queens on the same row, column, or diagonal.



- Here, goal state is initially unknown but is specified by constraints that it must satisfy.

Hill climbing (or gradient ascent/descent)

- Iteratively maximize "value" of current state, by replacing it by successor state that has highest value, as long as possible.

"Like climbing Everest in thick fog with amnesia"


```
function HILL-CLIMBING( problem) returns a solution state
   inputs: problem, a problem
   local variables: current, a node
                    next, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      next ← a highest-valued successor of current
      if VALUE[next] < VALUE[current] then return current
      current ← next
   end
```

Hill climbing

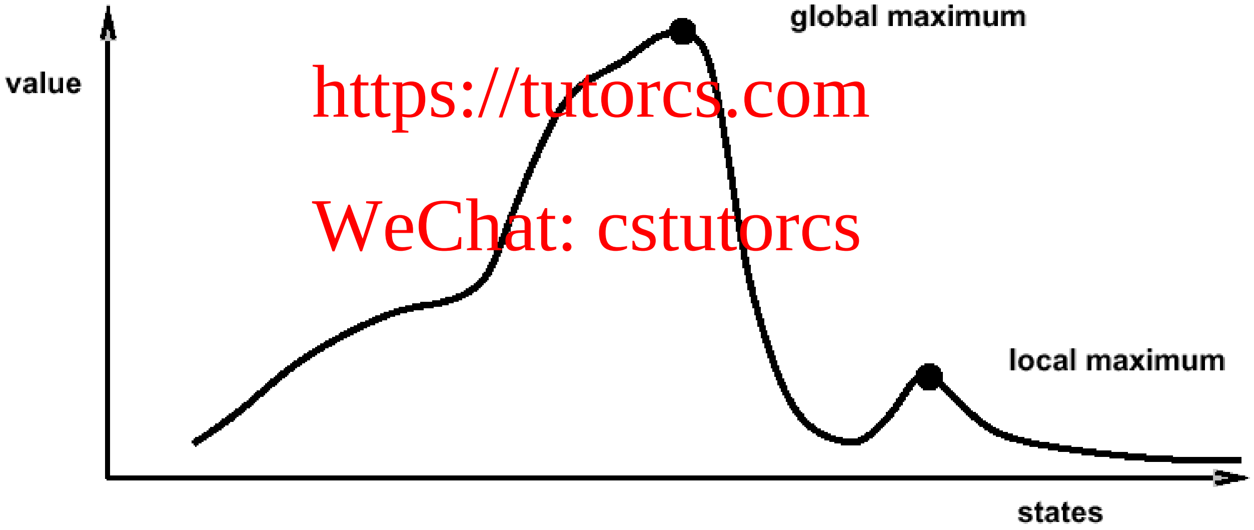
- Note: minimizing a "value" function v(n) is equivalent to maximizing –v(n),

  thus both notions are used interchangeably.

- Notion of "extremization": find extrema (minima or maxima) of a value function.

Hill climbing

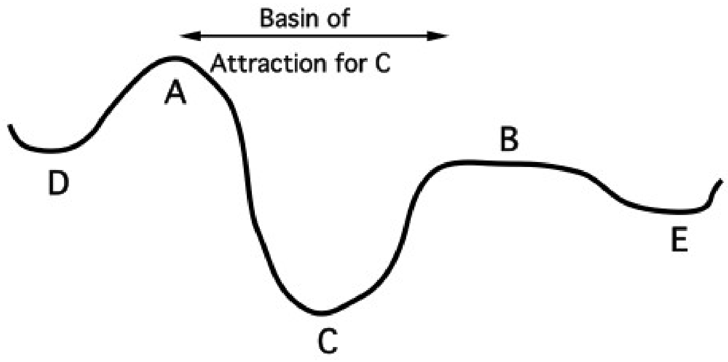
- Problem: depending on initial state, may get stuck in local extremum.



Minimizing energy

- Let’s now change the formulation of the problem a bit, so that we can employ new formalism:
  - let’s compare our state space to that of a physical system that is subject to natural interactions,
  - and let’s compare our value function to the overall potential energy E of the system.

- On every updating, we have $\Delta E \leq 0$

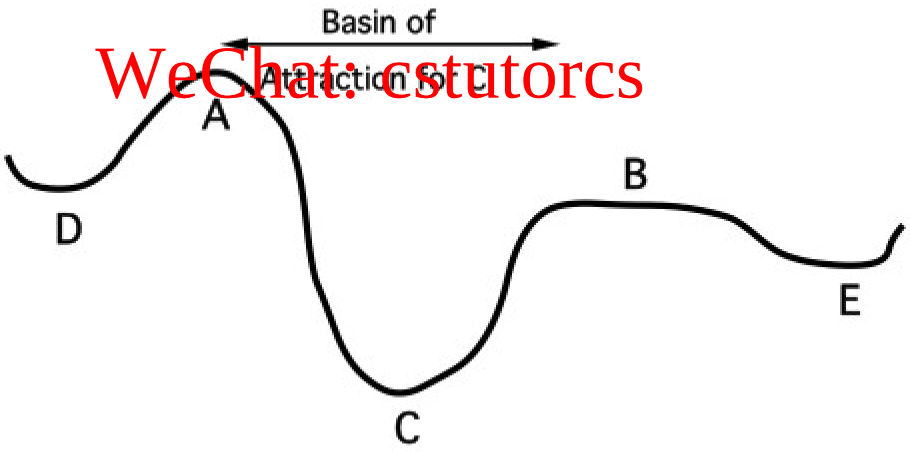


C

Minimizing energy

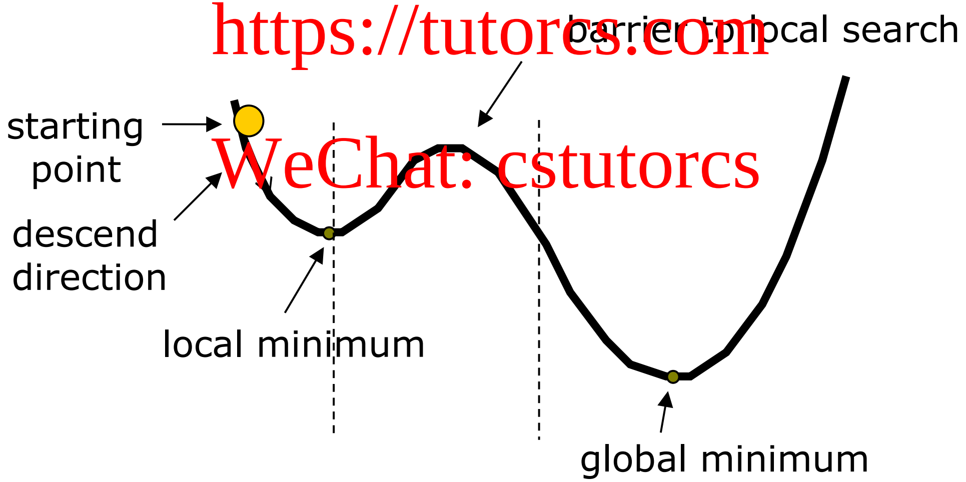
- Hence the dynamics of the system tend to move E toward a minimum.

- We stress that there may be different such states — they are *local* minima. Global minimization is not guaranteed.

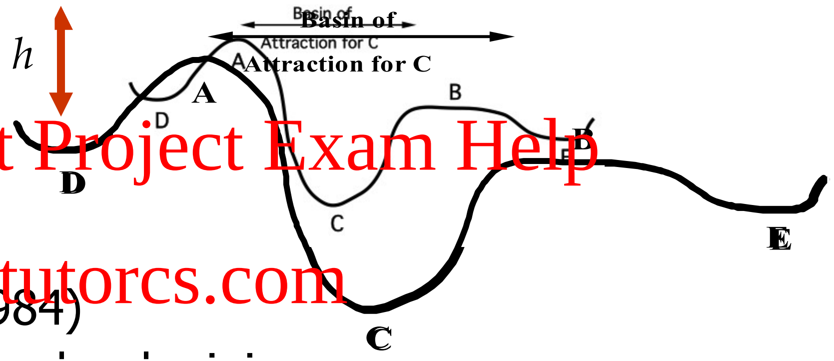


Local Minima Problem

- Question: How do you avoid this local minimum?



Boltzmann machines



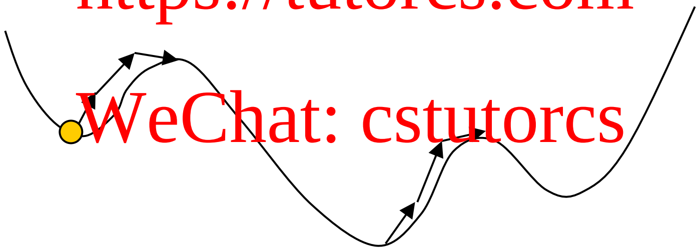
The Boltzmann Machine of Hinton, Sejnowski, and Ackley (1984) uses simulated annealing to escape local minima.

To motivate their solution, consider how one might get a ball-bearing traveling along the curve to "probably end up" in the deepest minimum. The idea is to shake the box "about h hard" — then the ball is more likely to go from D to C than from C to D. So, on average, the ball should end up in C's valley.

Consequences of the Occasional Ascents

desired effect

Help escaping the local optima.



adverse effect

Might pass global optima after reaching it

(easy to avoid by keeping track of best-ever state)

Simulated annealing: basic idea

- From current state, pick a **random** successor state;

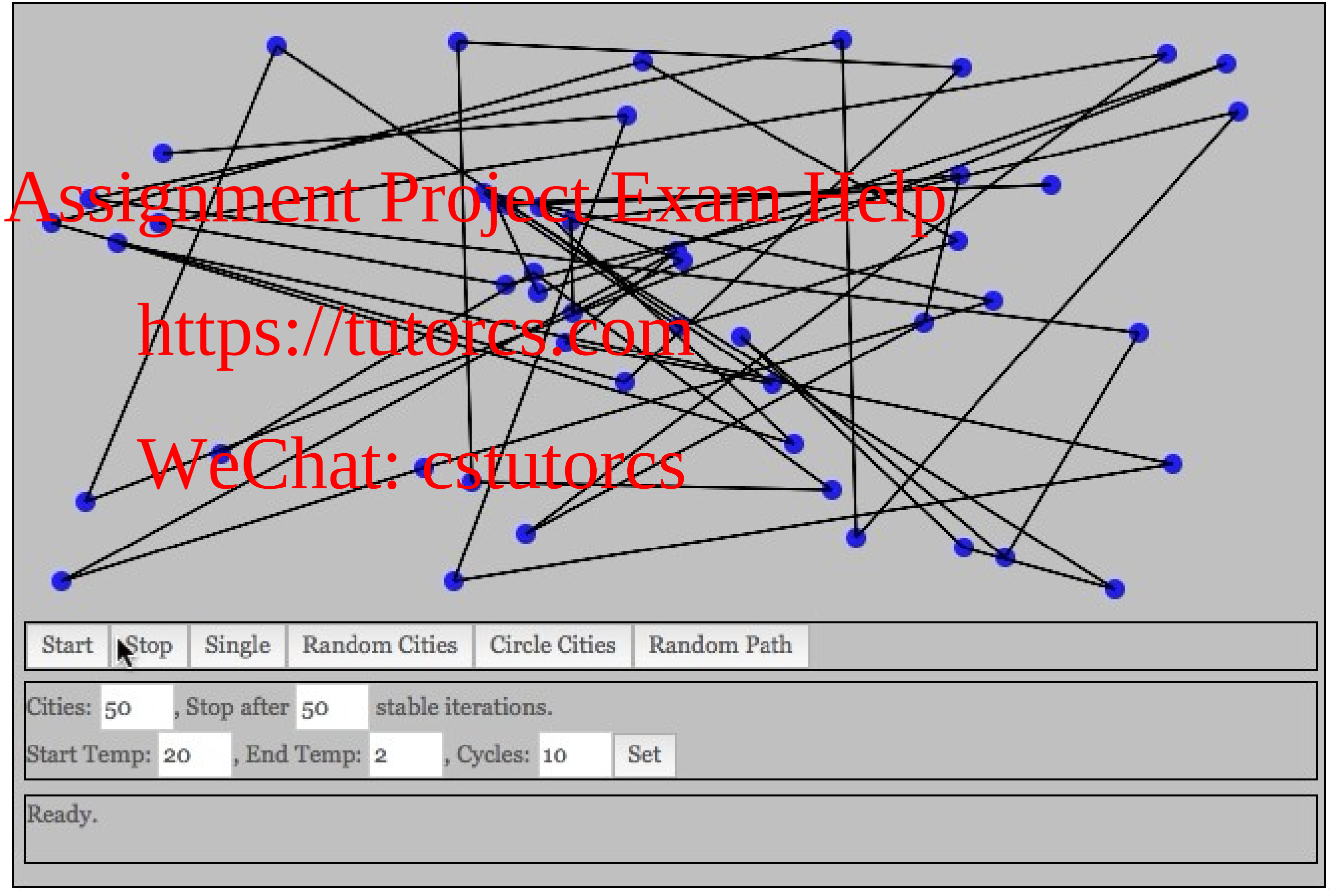
- If it has better value than current state, then "accept the transition," that is, use successor state as current state;

- Otherwise, do not give up, but instead flip a coin and accept the transition with a given probability (that is lower as the successor is worse).

- So we accept to sometimes "un-optimize" the value function a little with a non-zero probability.

Demo

AIFH Volume 1, Chapter 9: Traveling Salesman (TSP): Simulated Annealing



Boltzmann's statistical theory of gases

- In the statistical theory of gases, the gas is described not by a deterministic dynamics, but rather by the probability that it will be in different states.

- The 19th century physicist Ludwig Boltzmann developed a theory that included a probability distribution of temperature (i.e., every small region of the gas had the same kinetic energy).

- Hinton, Sejnowski and Ackley's idea was that this distribution might also be used to describe neural interactions, where low temperature T is replaced by a small noise term T (the neural analog of random thermal motion of molecules). While their results primarily concern optimization using neural networks, the idea is more general.

Boltzmann distribution

- At thermal equilibrium at temperature T, the Boltzmann distribution gives the relative probability that the system will occupy state A vs. state B as:

$$\frac{P(A)}{P(B)} = \exp\left(-\frac{E(A)-E(B)}{T}\right) = \frac{\exp(E(B)/T)}{\exp(E(A)/T)}$$

- where E(A) and E(B) are the energies associated with states A and B.

Simulated annealing

Kirkpatrick et al. 1983:

- Simulated annealing is a general method for making likely the escape from local minima by allowing jumps to higher energy states.
- The analogy here is with the process of annealing used by a craftsman in forging a sword from an alloy.
- He heats the metal, then slowly cools it as he hammers the blade into shape.
  - If he cools the blade too quickly the metal will form patches of different composition;
  - If the metal is cooled slowly while it is shaped, the constituent metals will form a uniform alloy.

Simulated annealing in practice

- set T
- optimize for given T
- lower T (see Geman & Geman, 1984)
- repeat

- Geman & Geman (1984): if T is lowered sufficiently slowly (with respect to the number of iterations used to optimize at a given T), simulated annealing is guaranteed to find the global minimum.

- Caveat: this algorithm has no end (Geman & Geman's T decrease schedule is in the 1/log of the number of iterations, so, T will never reach zero), so it may take an infinite amount of time for it to find the global minimum.

Simulated annealing algorithm

- Idea: Escape local extrema by allowing "bad moves," but gradually decrease their size and frequency.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling the probability of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T=0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next]   VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(ΔE/T)
```

Simulated annealing algorithm

- Idea: Escape local extrema by allowing "bad moves," but gradually decrease their size and frequency.


```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling the probability of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T=0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next]   VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(ΔE/T)
```

Note on simulated annealing: limit cases

- Boltzmann distribution: accept “bad move” with $\Delta E<0$ (goal is to maximize E) with probability $P(\Delta E) = \exp(\Delta E/T)$

- If T is large:
  $\Delta E < 0$
  $\Delta E/T < 0$ and very small
  $\exp(\Delta E/T)$ close to 1
  accept bad move with high probability

- If T is near 0:
  $\Delta E < 0$
  $\Delta E/T < 0$ and very large
  $\exp(\Delta E/T)$ close to 0
  accept bad move with low probability

Note on simulated annealing: limit cases

- Boltzmann distribution: accept "bad move" with $\Delta E<0$ (goal is to maximize E) with probability $P(\Delta E) = \exp(\Delta E/T)$

- If T is large:

  $\Delta E < 0$

  $\Delta E/T < 0$ and very small

  $\exp(\Delta E/T)$ close to 1

  accept bad move with high probability

Random walk

- If T is near 0:

  $\Delta E < 0$

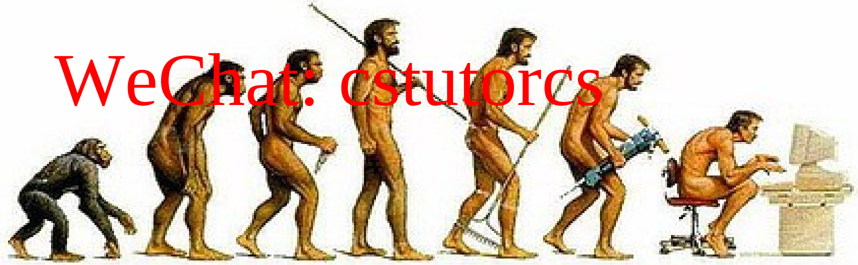
  $\Delta E/T < 0$ and very large

  $\exp(\Delta E/T)$ close to 0

  accept bad move with low probability

Deterministic up-hill

Genetic Algorithms

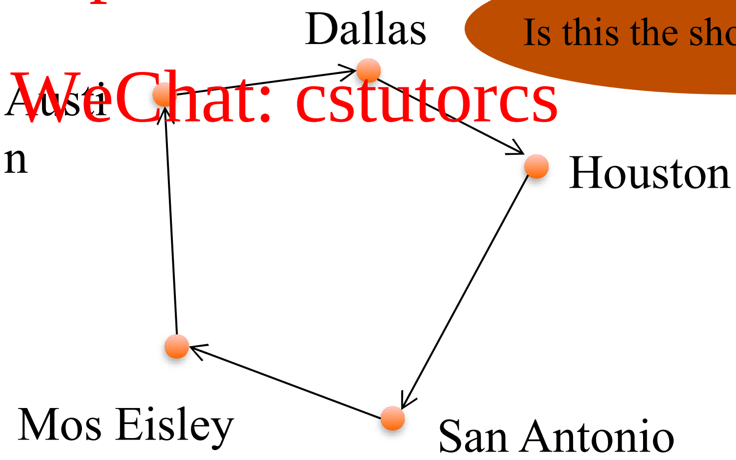


How do you find a solution in a large complex space?

- Ask an expert?
- Adapt existing designs?
- Trial and error?

Example: Traveling Sales Person (TSP)

- Classic Example: You have N cities, find the shortest route such that your salesperson will visit each city once and return.
- This problem is known to be NP-Hard
  - As a new city is added to the problem, computation time in the classic solution increases exponentially $O(2^n)$ … *(as far as we know)*



Is this the shortest path???

A Texas Sales Person

What if………

- Lets create a whole bunch of random sales people and see how well they do and pick the best one(s).
  - Salesperson A
    - Houston -> Dallas -> Austin -> San Antonio -> Mos Eisely
    - Distance Traveled 780 Km

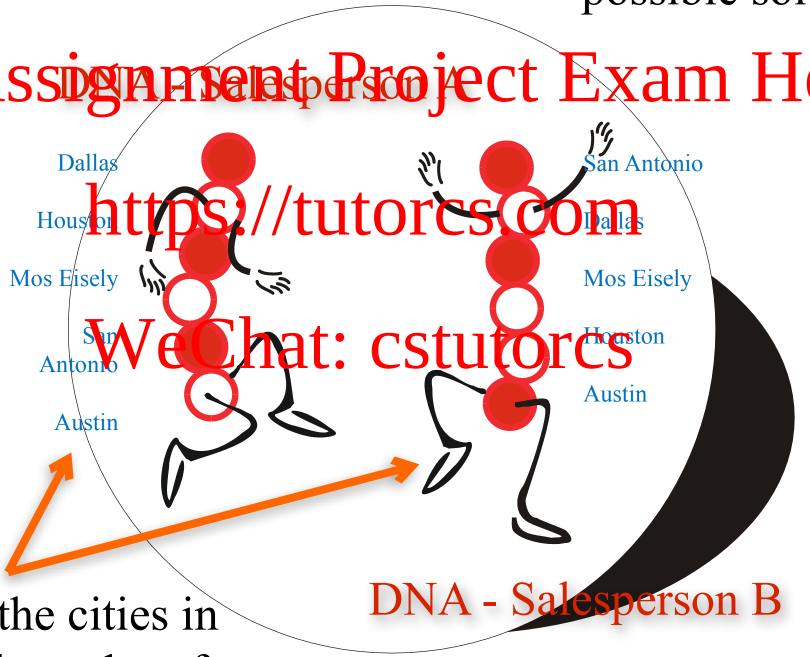
  - Salesperson B
    - Houston -> Mos Eisley -> Austin -> San Antonio -> Dallas
    - Distance Traveled 820 Km

  - Salesperson A is better (more fit) than salesperson B
  - Perhaps we would like sales people to be more like **A** and less like **B**

- Question:
  - do we want to just keep picking random sales people like this and keep testing them?

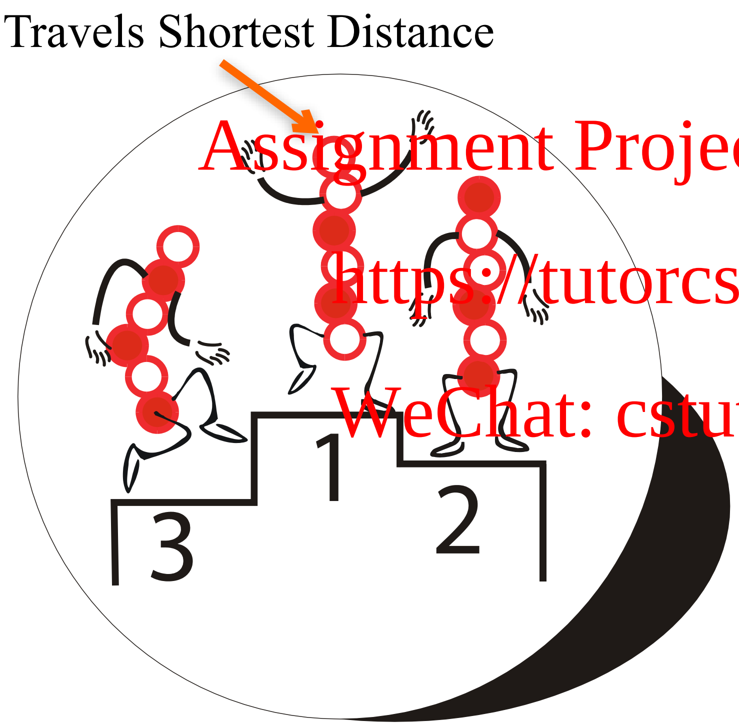
Represent problem like a DNA sequence

Each DNA sequence is an encoding of a possible solution to the problem.



The order of the cities in the genes is the order of the cities the TSP will take.

Ranking by Fitness:



Here we've created three different salespeople. We then checked to see how far each one has to travel. This gives us a measure of "Fitness"

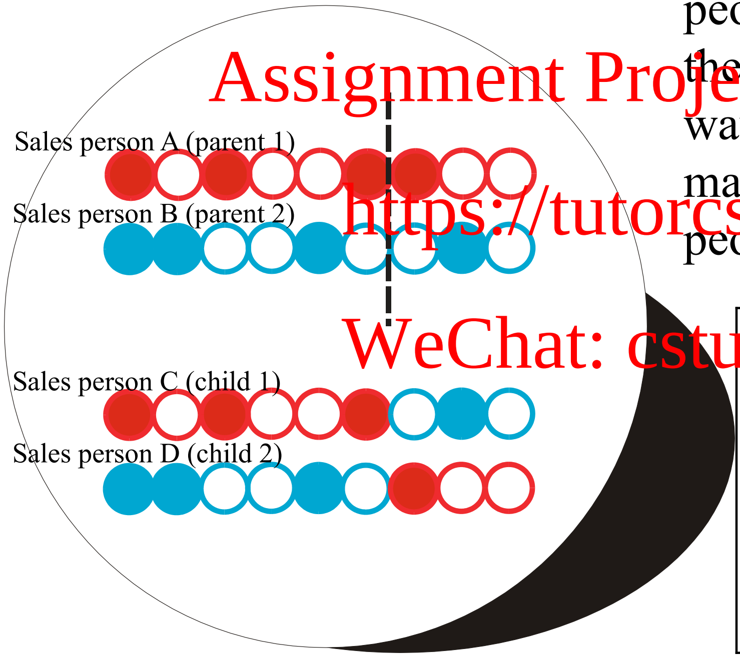
Note: we need to be able to measure fitness in polynomial time, otherwise we are in trouble.

Let’s breed them!

- We have a population of traveling sales people. We also know their fitness based on how long their trip is. We want to create more, but we don’t want to create too many.
- We take the notion that the salespeople who perform better are closer to the optimal salesperson than the ones which performed more poorly. Could the optimal sales person be a “combination” of the better sales people?
- We create a population of sales people as solutions to the problem.
- *How* do we actually mate a population of data???

Crossover:

Exchanging information through some part of information (representation)

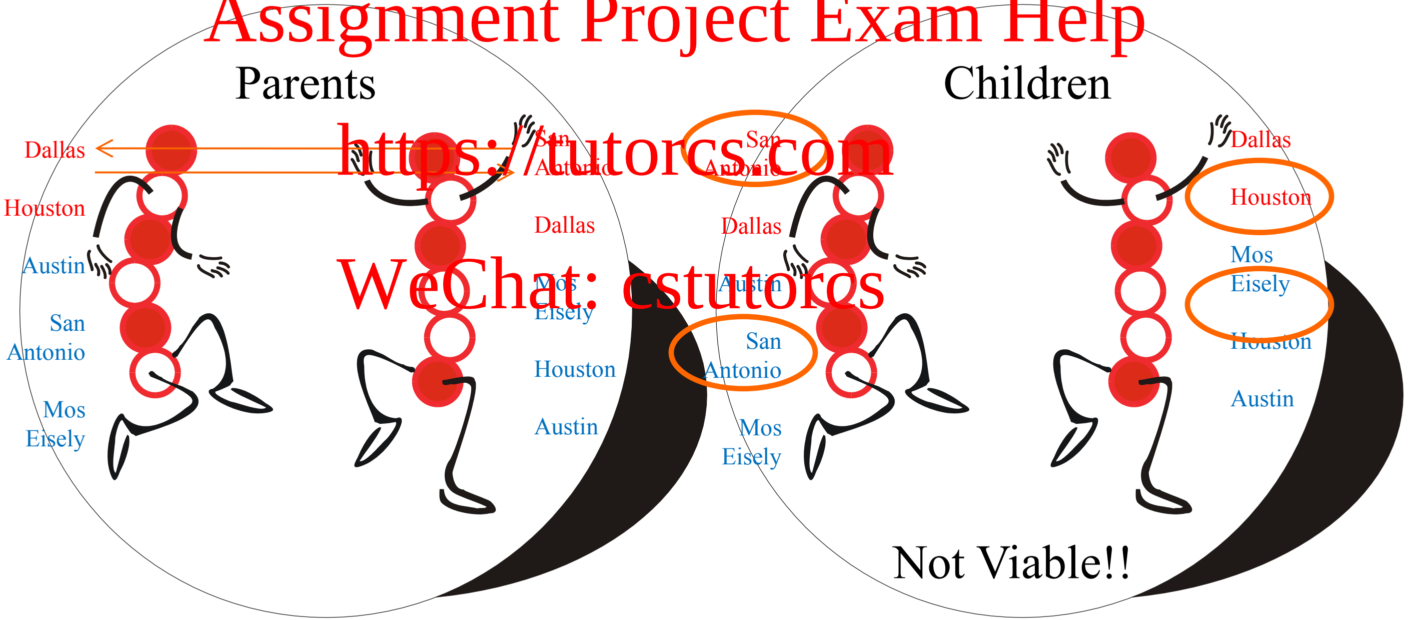


Once we have found the best sales people we will in a sense mate them. We can do this in several ways. Better sales people should mate more often and poor sales people should mate lest often.

Sales People	City DNA
Parent 1	F A B | E C G D
Parent 2	D E A | C G B F
Child 1	F A B | C G B F
Child 2	D E A | E C G D

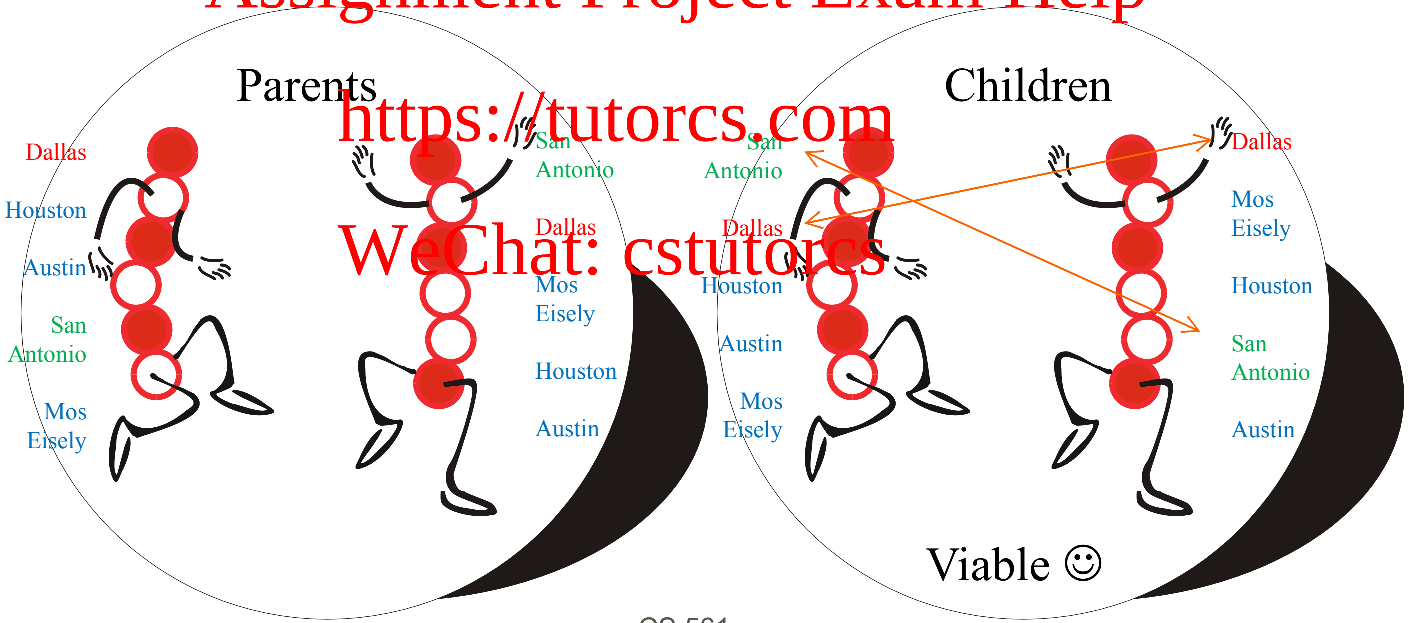
Crossover Bounds (Houston we have a problem)

- Not all crossed pairs are viable. We can only visit a city once.
- Different GA problems may have different bounds.



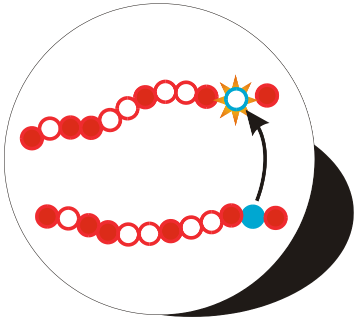
TSP needs some special rules for crossover

- Many GA problems also need special crossover rules.
- Since each genetic sequence contains all the cities in the travel, crossover is a swapping of travel order.
- Remember that crossover also needs to be efficient.

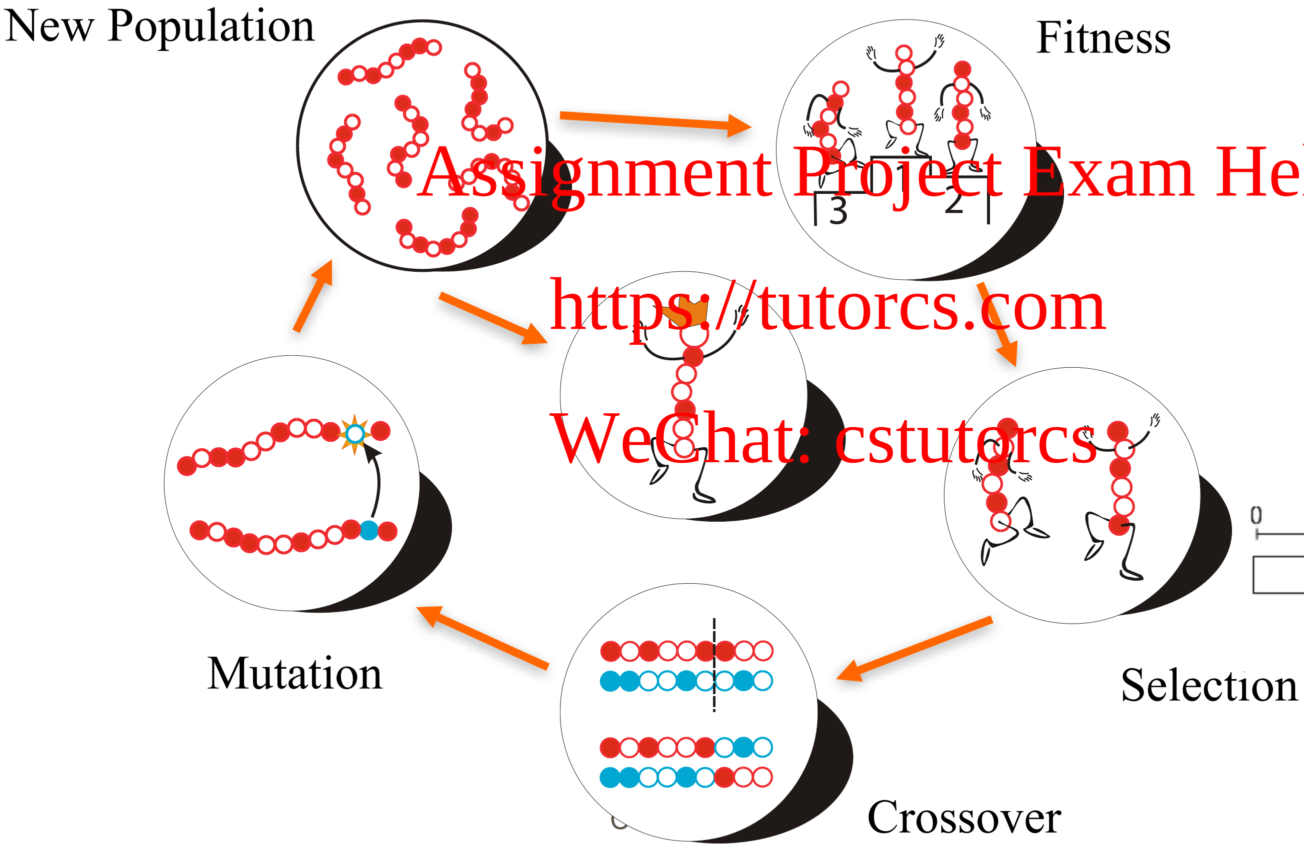


What about local extrema?

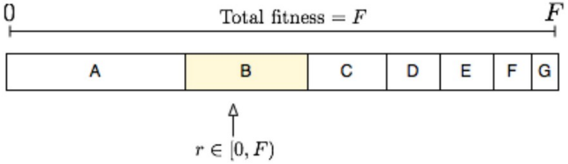
- With just crossover breading, we are constrained to gene sequences which are a cross product of our current population.
- Introduce random effects into our population.
  - Mutation – Randomly twiddle the genes with some probability.
  - Cataclysm – Kill off n% of your population and create fresh new salespeople if it looks like you are reaching a local minimum.
  - Annealing of Mating Pairs – Accept the mating of suboptimal pairs with some probability.
  - Etc…



In summation: The GA Cycle

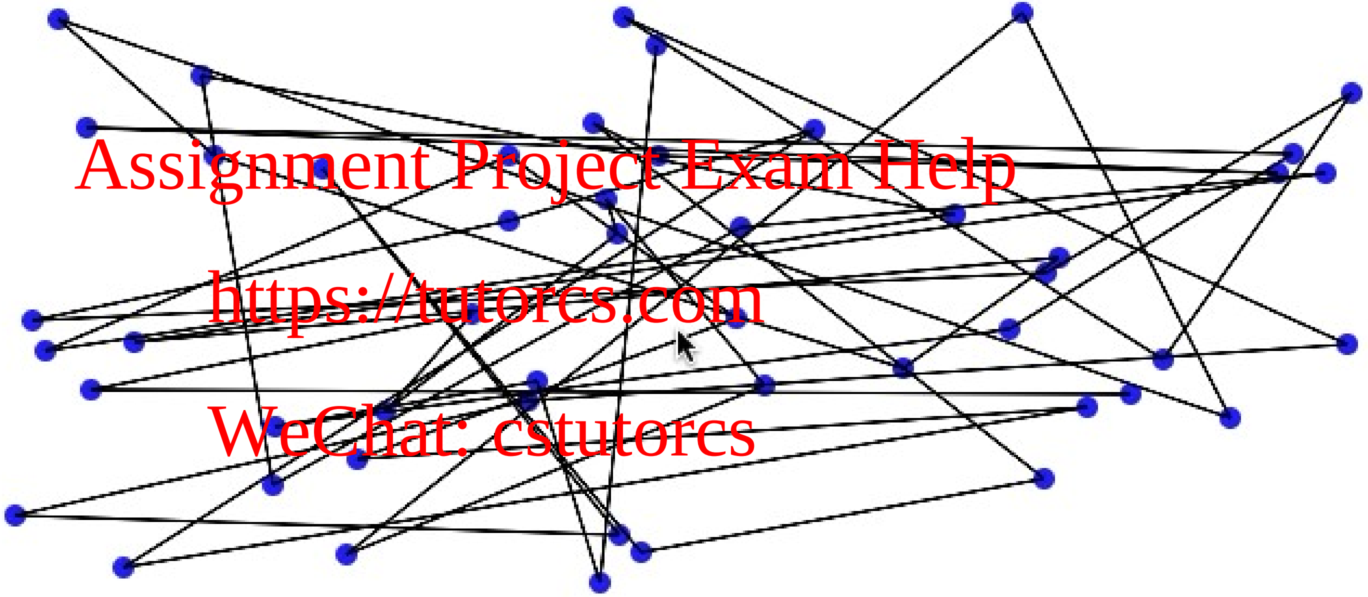


(see, e.g., Wikipedia on Fitness proportionate selection)



Demo

AIFH Volume 2, Chapter 9: Traveling Salesman (TSP): Genetic Algorithm



Start | Stop | Single | Random Cities | Circle Cities | New Population

Cities: 50 , Stop after 50 stable iterations.

Population: 1000 , Mutation %: 0.2 , % to Mate: 0.2 , Eligible Pop %: 0.5 Set

Ready.

GA and TSP: the claims

- Can solve for over 3500 cities (still took over 1 CPU years).
  - Maybe holds the record.
- Will get within 2% of the optimal solution.
  - This means that it's not a solution per se, but is an approximation.

GA Discussion

- We can apply the GA solution to any problem where the we can represent the problems solution (even very abstractly) as a string.

- We can create strings of:
  - Digits
  - Labels
  - Pointers
  - Code Blocks – This creates new programs from strung together blocks of code. The key is to make sure the code can run.
  - Whole Programs – Modules or complete programs can be strung together in a series. We can also re-arrange the linkages between programs.

- The last two are examples of Genetic Programming

Things to consider

- How large is your population?
  - A large population will take more time to run (you have to test each member for fitness!).
  - A large population will cover more bases at once.

- How do you select your initial population?
  - You might create a population of approximate solutions. However, some approximations might start you in the wrong position with too much bias.

- How will you cross bread your population?
  - You want to cross bread and select for your best specimens.

    - Too strict: You will tend towards local minima
    - Too lax: Your problem will converge slower
- How will you mutate your population?
  - Too little: your problem will tend to get stuck in local minima
  - Too much: your population will fill with noise and not settle.

GA is a good *no clue* approach to problem solving

- GA is superb if:
  - Your space is loaded with lots of weird bumps and local minima.
    - GA tends to spread out and test a larger subset of your space than many other types of learning/optimization algorithms.
  - You don't quite understand the underlying *process* of your problem space.
    - NO I DONT: What makes the stock market work??? Don't know? Me neither! Stock market prediction might thus be good for a GA.
    - YES I DO: Want to make a program to predict people's height from personality factors? This might be a Gaussian process and a good candidate for statistical methods which are more efficient.
  - You have lots of processors
    - GA's parallelize very easily!

Why not use GA?

- Creating generations of samples and cross breading them can be resource intensive.
  - Some problems may be better solved by a general gradient descent method which uses less resource.
  - However, resource-wise, GA is still quite efficient (no computation of derivatives, etc).

- In general if you know the mathematics, shape or underlying process of your problem space, there may be a better solution designed for your specific need.
  - Consider Kernel Based Learning and Support Vector Machines?
  - Consider Neural Networks?
  - Consider Traditional Polynomial Time Algorithms?
  - Etc.

More demos: motorcycle design

Summary

- Best-first search = general search, where the minimum-cost nodes (according to some measure) are expanded first.

- Greedy search = best-first with the estimated cost to reach the goal as a heuristic measure.
  - Generally faster than uninformed search
  - not optimal
  - not complete.

- A* search = best-first with measure = path cost so far + estimated path cost to goal.
  - combines advantages of uniform-cost and greedy searches
  - complete, optimal and optimally efficient
  - space complexity still exponential

- Hill climbing and simulated annealing: iteratively improve on current state
  - lowest space complexity, just O(1)
  - risk of getting stuck in local extrema (unless following proper simulated annealing schedule)

- Genetic algorithms: parallelize the search problem