Consider the optimal path.
Every node on this path has what property
What about the start state. Therefore A* has what property wrt node expansion? . If A* is optimal what other property wrt node expansion must hold.
Proof of Optimality of A*

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 $G_1$.

![Diagram showing nodes and paths]

\[ f(G_2) = g(G_2) \quad \text{since} \quad h(G_2) = 0 \]
\[ > g(G_1) \quad \text{since} \quad G_2 \text{ is suboptimal} \]
\[ \geq f(n) \quad \text{since} \quad h \text{ is admissible} \]

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

- For a given heuristic function, no optimal algorithm is guaranteed to do less work.
- Aside from ties in \( f() \), A* expands every node necessary for the proof that we’ve found the shortest path, and no unnecessary nodes.
A* graph search

1. Start with OPEN containing just the initial state.
2. Until a goal is found or there are no nodes on OPEN do:
   (a) Select the node on OPEN w/ the lowest f-value.
   (b) Generate its successors.
   (c) For each successor do:
       i. If it hasn’t been generated before (i.e., it’s not in CLOSED) evaluate it, add it to OPEN, and record its parent.
       ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have. Then add node to CLOSED list.
Iterative Deepening A* (IDA*)

- Use $f(N) = g(N) + h(N)$ with admissible $h$
- Each iteration is depth-first with cutoff on the value of $f$ of expanded nodes
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) = \) number of misplaced tiles

Cutoff=4
f(N) = g(N) + h(N) with h(N) = number of misplaced tiles
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) = \) number of misplaced tiles
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) \) = number of misplaced tiles
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) \) = number of misplaced tiles
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles

Cutoff=5
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) = \) number of misplaced tiles

Cutoff = 5
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles

8-Puzzle

Cutoff=5
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles
f(N) = g(N) + h(N) with h(N) = number of misplaced tiles
f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles
Examples

• For each, specify the state space, operators, constraints of the problem and admissible heuristics
  – Towers of Hanoi
  – Robot path planning
  – TSP
  – Rubik’s Cube
What is the admissible Heuristic?

Consider the constraints of the problem

Well-known example: travelling sale
Find the shortest tour visiting all cities
Well-known example: travelling salesman problem (TSP)
Find the shortest tour visiting all cities exactly once.
Iterative Improvement Techniques

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

Then state space = set of “complete” configurations;
  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, try to improve it.

Constant space, suitable for online as well as offline search.

Two key issues: how to perturb, how to measure “goodness” of solution.
Simple Problem

Find the shortest tour that visits each city exactly once
N-Queens (Again)

Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
Gradient Descent/Ascent

“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
Simulated Annealing

Idea: escape local maxima by allowing some “bad” moves but gradually decrease their size and frequency

\[
\textbf{function} \ \text{SIMULATED-ANNEALING}(\ \text{problem, schedule}) \ \textbf{returns} \ \text{a solution state}
\]
\[
\textbf{inputs:} \ \text{problem}, \ \text{a problem}
\]
\[
\text{schedule}, \ \text{a mapping from time to “temperature”}
\]
\[
\textbf{local variables:} \ \text{current, a node}
\]
\[
\text{next, a node}
\]
\[
T, \ \text{a “temperature” controlling the probability of downward steps}
\]
\[
current \leftarrow \text{MAKE-NODE(INITIAL-STATE[problem])}
\]
\[
\text{for } t \leftarrow 1 \ \text{to } \infty \ \text{do}
\]
\[
T \leftarrow \text{schedule}[t]
\]
\[
\text{if } T=0 \text{ then return current}
\]
\[
next \leftarrow \text{a randomly selected successor of current}
\]
\[
\Delta E \leftarrow \text{VALUE[next] - VALUE[current]}
\]
\[
\text{if } \Delta E > 0 \text{ then } current \leftarrow next
\]
\[
\text{else } current \leftarrow next \text{ only with probability } e^{\Delta E/T}
\]
What’s in a Proof? Proof of Asymptotic Convergence

since \( \lim_{x \downarrow 0} a^{bx} = 1 \) if \( b = 0 \) and \( 0 \) if \( b < 0 \)

\[
\lim_{c \downarrow 0} P(X = i) = \lim_{c \downarrow 0} \frac{\exp\left(-\frac{f(i)}{c}\right)}{\sum_{j \in S} \exp\left(-\frac{f(j)}{c}\right)} \\
= \lim_{c \downarrow 0} \frac{\exp\left(\frac{f_{opt} - f(i)}{c}\right)}{\sum_{j \in S} \exp\left(\frac{f_{opt} - f(j)}{c}\right)} \\
= \lim_{c \downarrow 0} \frac{1}{\sum_{j \in S} \exp\left(\frac{f_{opt} - f(j)}{c}\right)} \chi(S_{opt})(i) \\
+ \lim_{c \downarrow 0} \frac{\exp\left(\frac{f_{opt} - f(i)}{c}\right)}{\sum_{j \in S} \exp\left(\frac{f_{opt} - f(j)}{c}\right)} \chi(S \setminus S_{opt})(i) \\
= \frac{1}{|S_{opt}|} \chi(S_{opt})(i) + 0

hence:

\[
\lim_{c \downarrow 0} P(X \in S_{opt}) = 1
\]
Simulated Annealing Properties

At fixed “temperature” $T$, state occupation probability reaches Boltzmann distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

$T$ decreased slowly enough $\implies$ always reach best state

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.
Genetic Algorithms - Briefly

- invented in mid-1970s by John Holland

- GAs used the principle of natural selection to solve complicated optimization problems
  - transportation scheduling
  - car dealership location
  - pipeline sizing
  - VLSI chip layout

- However GAs are not a panacea ...
  - their efficiency is critically dependant on finding a good representation and good operators
An Overview of GA’s

A genetic algorithm is a computer simulation in which a population of abstract representations of candidate solutions to an optimization problem are stochastically selected, recombined, mutated, and then either eliminated or retained, based on their relative fitness.

- from an initial population
- select individuals
- breed them
- mutate the offspring
- insert them into the next generation
Biology Terminology

- **a gene is a set of possible alleles, each of which codes for a different variation of some characteristic**
  - gene for eye colour
  - alleles: blue, green, brown, etc.
  - a gene pool is the collection of all alleles in a particular population (the set of available options for the next generation)

- **a genome is the set of all the genes defining a species**
  - a particular set of genes describing an individual is called a genotype
  - many living organisms store their genome as several chromosomes
  - in GAs, chromosome and genome often used as synonyms

- **a population for a GA is a set of chromosomes**
Basic GA Representation

The original Holland's GA is used as a starting point for almost all new work.

A simple GA represents individuals solutions using string of bits:

```
1 0 1 1 0 1 0 0 1 1 1 0
```

- This bits may encode integers, real numbers, sets, or whatever ...
- Original analysis of GAs assumed binary representation
- Makes operators simple
- Most practical GAs use problem-specific representation
Simple Cross Over

- **1-point cross-over**

  father
  ![father diagram]

  mother
  ![mother diagram]

  child
  ![child diagram]

- several different forms of crossover are now used, and inversion is no more use

- **mutation**
Creation of the Next Generation

- selection based directly on fitness

- generational update
  - create entirely a new population and replace all of the old one
  - no breeding possible between individuals from different generations

- single population
  - anyone could potentially breed with any other

Representation, cross-over, mutation, fitness for n-Queens problem, TSP
CSI535 — Introduction to A.I. Assignment #1: Direction Finding Using A*

Due: Friday 03/03/04
Worth: 20% of Final Grade
Late Policy: You lose one full grade for each week (including partial weeks) you are late.

Read the instructions carefully, ask questions if you have any doubts.

The A* algorithm is a very useful A.I. technique for direction finding in applications such as robot navigation and computer game character movement (many advancements to A* have been created by the gaming industry). Direction finding involves making the optimal series of actions to move from a start to goal point.

You will be using an environment that consists of a 20 x 20 cell universe where your agent takes up exactly one cell. Your agent can make only one of four movements (back, forward, right, left one block). You will begin by implementing A* to navigate this universe in its most basic form and then make changes to handle progressively difficult versions of the environment.
Question 1. Basic version of environment (20 points)

Assume that your agent can see the entire map. Implement a program to determine the optimum series of actions (shortest distance) for the following map using A*. The heart represents the goal state, the smiley face the start state, the black blocks are borders that cannot be crossed, all non-filled blocks can be passed through.

![Map Diagram]

Define the state space, constraints, actions and goal test. Carefully define your admissible heuristic under all conditions and your $f$ and $g$ functions. Draw the first two levels of your data structure (nominate if you intend to use a graph or tree) showing $f, g$ and $h$ values. Present your results by showing the $f(n) = g(n) + h(n)$ values for each entry in the map. Present the pseudo code to your algorithm.
Question 2. Environment with elevation (25 points)

Now imagine the environment has a topography to it. In this question you will be using the map below. Some cells are “higher” than others, your agent can climb from a lower to a higher elevated cell at a fixed cost dependent only on the elevation of the cell but not on the elevation of the cell the agent is moving from (this is a simplification). As before, the black cells are impossible to go through, the dark gray cells can be traversed through at cost 10, the light gray cells the cost is 5, traversing through all remaining cells cost 1.

Define the changes (if any) to your space state space, constraints, actions and goal test. Carefully define your admissible heuristic under all conditions and your $f$ and $g$ functions. Present your pseudo code. Present your results by showing the $f(n) = g(n) + h(n)$ values for each entry in the following map. Note the non-uniform cost aspect of $A^*$ can be used to handle direction planning to avoid oppositions, pits, collect as much reward as possible etc.
Question 3. Environment with topography and partial access to the environment (35 points)

Now imagine that your agent does not have a map, but does have 360 degree vision but the cells with non-zero height block his/her entire view beyond the cell. The agent perceives two predicates for each cell: isBlock() and isGoal(). Discuss changes to your basic algorithm that allow handling an environment with topography and partial access. Define the state space, constraints, actions and goal test. Carefully define your admissible heuristic under all conditions and your $f$ and $g$ functions. Present your pseudo code. Present your results by showing the $f(n) = g(n) + h(n)$ values for each entry in the above map. Does your agent find the same trail as before?

Question 4. Is close enough good enough? (20 points)

Much has been made of the need for A* to use an admissible heuristic. It is known that if $h(n)$ underestimates the value $f(OPTIMAL\_GOAL) - f(n)$ for all $n$ then A* returns the optimal route. However, what if $h(n)$ occasionally overestimates the correct value by at most some small number $\varepsilon$. Clearly the proof of optimality for A* no longer holds. But does the algorithm now return hopelessly sub-optimal results? Empirically (Ph.D. students can tackle this formally) investigate the cost of above optimality if your heuristic overestimates the correct cost occasionally. You can introduce over-estimation by adding a stochastic (RNG) aspect to the calculation of $h(n)$.

For the interested student

Question 3 and Question 4 introduce two simplifications. Firstly, that the cost to go to a cell of a specific height only depends on that cell. If the cost of going between two cells is $|Cell_A - Cell_B|$ what would be an admissible heuristic? Secondly, if blocks of non-zero height only block out what can be seen behind according to the distance the agent is from the block and the height of the cell, what is an admissible heuristic?