MidTerm
Tues. 18 Nov

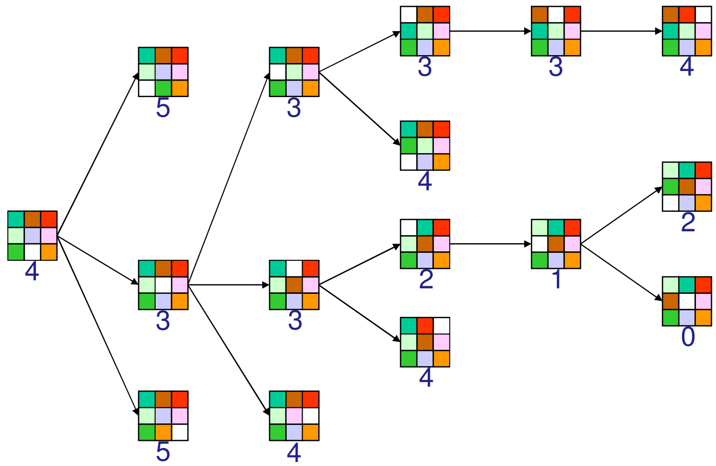
Up to the end of this Lecture

Local Search Algorithms

- In many optimization problems, *path* is irrelevant
- the goal state itself is the solution
- Ex: The 8-queen problem, the final configuration of the queens is the important not the order they were put
- Operates using only single current state, rather than multiple paths.
- Find Optimal Configuration (satisfies the constraints)
- Use *iterative improvement algorithms*
- Good for Optimization problems: find the best state according to some objective function
- A **Complete** local search algorithm finds a goal if exists
- An **Optimal** algorithm finds the global minimum or maximum

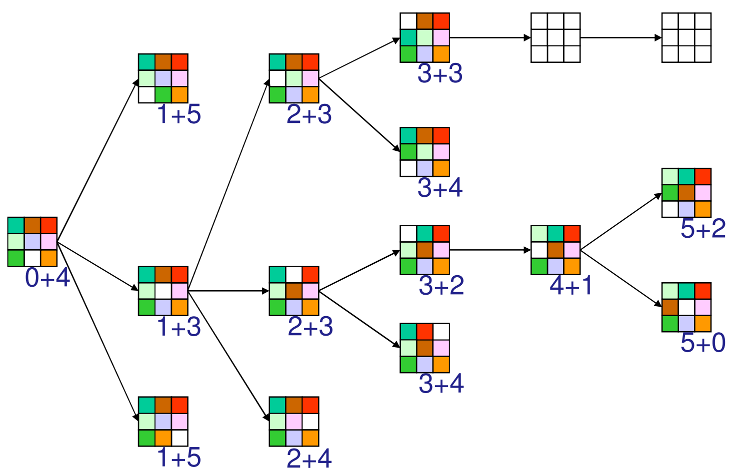
8-Puzzle

f(N) = h(N) = number of misplaced tiles



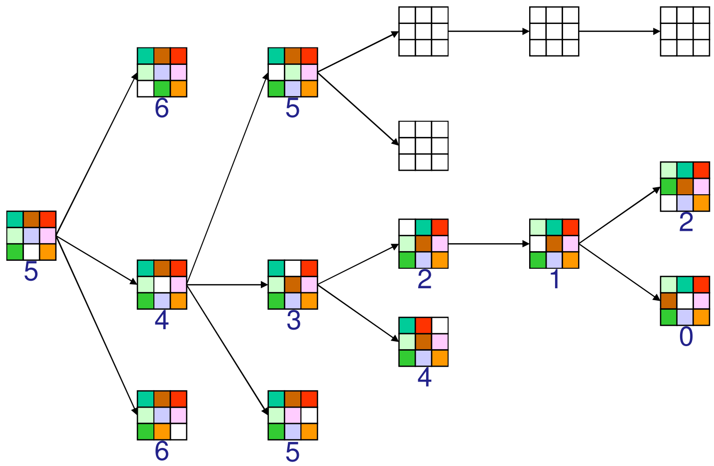
8-Puzzle

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



8-Puzzle

$f(N) = h(N) = \sum$ distances of tiles to goal



Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed 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

The A* procedure

Hill-climbing (and its improved versions) may miss an **optimal** solution. Here is a search method that ensures **optimality** of the solution.

The algorithm

keep a list of partial paths (initially root to root, length 0);

repeat

succeed if the first path P reaches the goal node;

otherwise remove path P from the list;

extend P in all possible ways, add new paths to the list;

sort the list by the sum of two values: the real cost of P till now, and an estimate of the remaining distance;

prune the list by leaving only the shortest path for each node reached so far;

until

success or the list of paths becomes empty;

Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Theorem: If $h(n)$ is admissible, A^* using is optimal

A* Algorithm- Properties

- **Admissibility**: An algorithm is called admissible if it always terminates and terminates in optimal path
- **Theorem**: A* is admissible.
- **Lemma**: Any time before A* terminates there exists on *OL* a node *n* such that $f(n) <= f^*(s)$
- **Observation**: For optimal path $s \rightarrow n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow g$,
  1. $h^*(g) = 0$, $g^*(s)=0$ and
  2. $f^*(s) = f^*(n_1) = f^*(n_2) = f^*(n_3)\ldots = f^*(g)$

Algorithm A*

- $f^*(n) = g^*(n) + h^*(n)$, where,
- $g^*(n)$ = actual cost of the optimal path (s, n)
- $h^*(n)$ = actual cost of optimal path (n, g)
- $g(n) \leq g^*(n)$
- By definition, $h(n) \leq h^*(n)$
- $h(n) <= h^*(n)$ *where* $h^*(n)$ *is the actual cost of optimal path to G(node to be found) from n*

Lemma

Any time before A* terminates there exists in the open list a node n' such that $f(n') <= f^*(S)$

For any node n_i on optimal path,

$f(n_i) = g(n_i) + h(n_i)$

$<= g^*(n_i) + h^*(n_i)$

Also $f^*(n_i) = f^*(S)$

Let n' be the fist node in the optimal path that is in OL. Since all parents of n' have gone to CL,

$g(n') = g^*(n')$ *and* $h(n') <= h^*(n')$

$=> f(n') <= f^*(S)$

A* always terminates

Proof

If A* does not terminate

Let e be the least cost of all arcs in the search graph.

Then $g(n) >= e.l(n)$ where $l(n)$ = # of arcs in the path from S to n found so far. If A* does not terminate, $g(n)$ and hence $f(n) = g(n) + h(n)$ $[h(n) >= 0]$ will become unbounded.

This is not consistent with the lemma. So A* has to terminate.

Admissibility of A*

The path formed by A* is optimal when it has terminated

Proof
Suppose the path formed is not optimal
Let G be expanded in a non-optimal path.
At the point of expansion of G,

$f(G) = g(G) + h(G)$
$= g(G) + 0$
$> g^*(G) = g^*(S) + h^*(S)$
$= f^*(S)$ [$f^*(S)$ = cost of optimal path]

This is a contradiction
So path should be optimal

A list of AI Search Algorithms

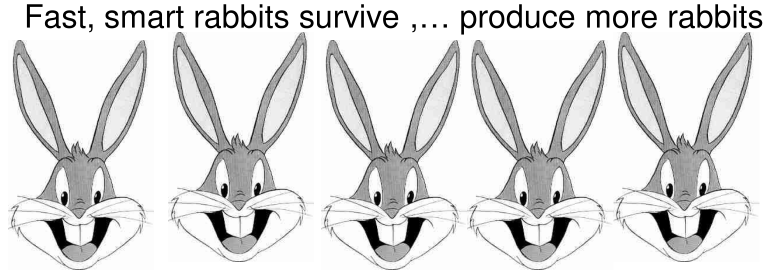
Systematic Search algorithms
- BFS, DFS,...
- A*
  - AO*
  - IDA* (Iterative Deepening)

Local Search Algorithms
- Minimax Search on Game Trees
- Viterbi Search on Probabilistic FSA
- Hill Climbing
- Simulated Annealing
- Gradient Descent
- Stack Based Search
- Genetic Algorithms
- Memetic Algorithms

The Genetic Algorithm (Evolutionary Analogy)

- Consider a population of rabbits:
  - some individuals are faster and smarter than others
  - Slower, dumper rabbits are likely to be caught and eaten by foxes

  - Fast, smart rabbits survive ,… produce more rabbits.



Evolutionary Analogy

- The rabbits that survive generate offspring, which start to mix up their genetic material
- Furthermore, nature occasionally throws in a wild properties because genes can mutate
- In this analogy, an individual rabbit represents a solution to the problem(i.e. Single point in the space)
- The foxes represent the problem constraints (solutions that do more well are likely to survive)

Evolutionary Analogy

- For selection, we use a fitness function to rank individuals of the population
- For reproduction, we define a crossover operator which takes state descriptions of individuals and combine them to create new ones
- For mutation, we can choose individuals in the population and alter part of its state.

The Genetic Algorithm

- Directed search algorithms based on the mechanics of biological evolution
- Developed by John Holland, University of Michigan (1970's)
- To design artificial systems software that retains the robustness of natural systems
- Provide efficient, effective techniques for search problems, optimization and machine learning applications
- Widely-used today in business, scientific and engineering circles

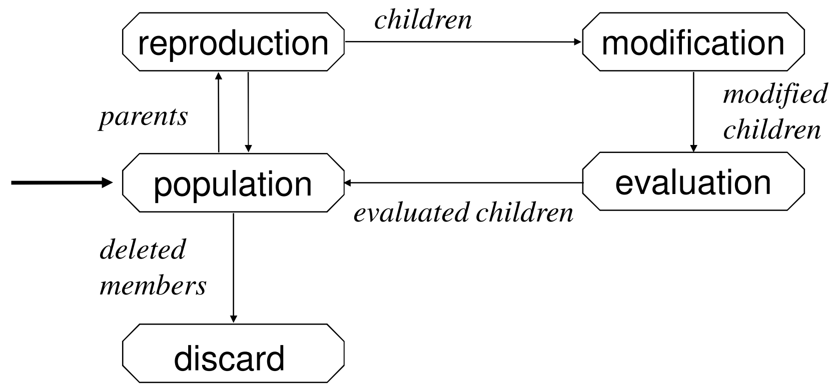
Terminology

- *Evolutionary Computation (EC)* refers to computer-based problem solving systems that use computational models of evolutionary process.
- *Chromosome* – It is an individual representing a candidate solution of the optimization problem.
- *Population* – A set of chromosomes.
- *gene* – It is the fundamental building block of the chromosome, each gene in a chromosome represents each variable to be optimized. It is the smallest unit of information.
- Objective: To find "a" best possible chromosome for a given problem.

Overview of GAs

- GA emulate genetic evolution.
- A GA has distinct features:
  - A string representation of chromosomes.
  - A selection procedure for initial population and for off-spring creation.
  - A cross-over method and a mutation method.
  - A fitness function.
  - A replacement procedure.
- Parameters that affect GA are initial population, size of the population, selection process and fitness function.

The GA Cycle of Reproduction

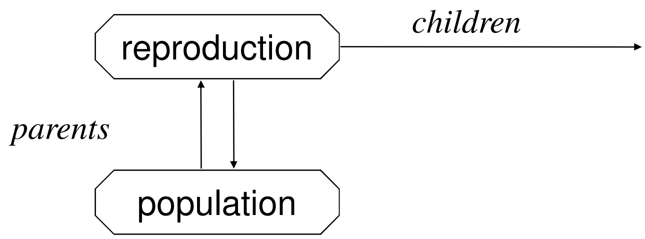


Chromosomes

Chromosomes could be:

Bit strings	(0101 ... 1100)
Real numbers	(43.2 -33.1 ... 0.0 89.2)
Permutations of element	(E11 E3 E7 ... E1 E15)
Lists of rules	(R1 R2 R3 ... R22 R23)
Program elements	(genetic programming)
... any data structure ...	

Reproduction



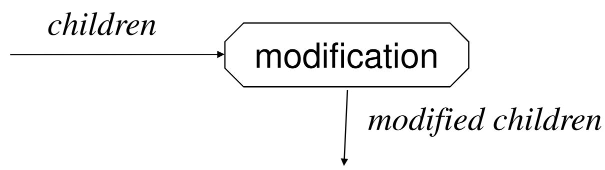
Parents are "selected" at each iteration.

Selection Process

- Selection is a procedure of picking parent chromosome to produce off-spring.
- Types of selection:
  - Random Selection – Parents are selected randomly from the population.
  - Proportional Selection – probabilities for picking each chromosome is calculated as:

$$P(\mathbf{x}_i) = f(\mathbf{x}_i)/\Sigma f(\mathbf{x}_j) \quad \text{for all } j$$

Chromosome Modification



- Operator types are:
  - Mutation
  - Crossover (recombination)

Crossover

P1 (0 1 1 0 1 0 0 0) → (1 1 0 1 1 0 0 0) C1

P2 (1 1 0 1 1 0 1 0) → (0 1 1 0 1 0 1 0) C2

Crossover is a critical feature of genetic algorithms:

- It greatly accelerates search early in evolution of a population
- It leads to effective combination of schemata (subsolutions on different chromosomes)

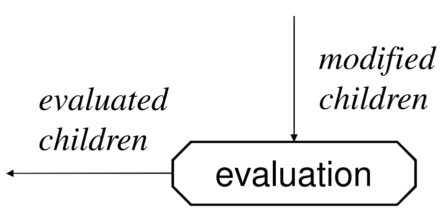
Mutation: Local Modification

Before:	(1 0 1 1 0 1 1 0)
After:	(1 0 1 1 1 1 1 0)

Before:	(1.38 -69.4 326.44 0.1)
After:	(1.38 -67.5 326.44 0.1)

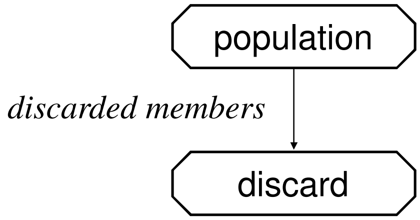
- Causes movement in the search space (local or global)
- Restores lost information to the population

Evaluation

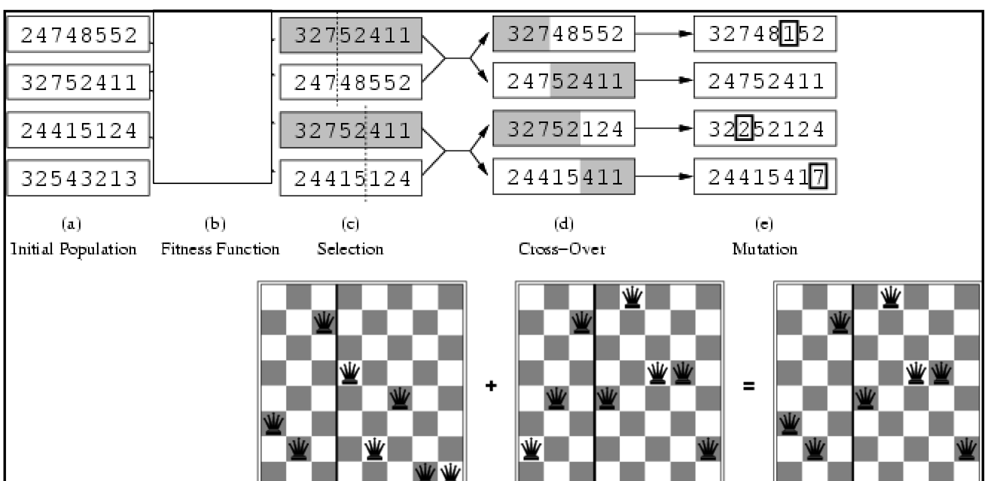


- The evaluator decodes a chromosome and assigns it a fitness measure

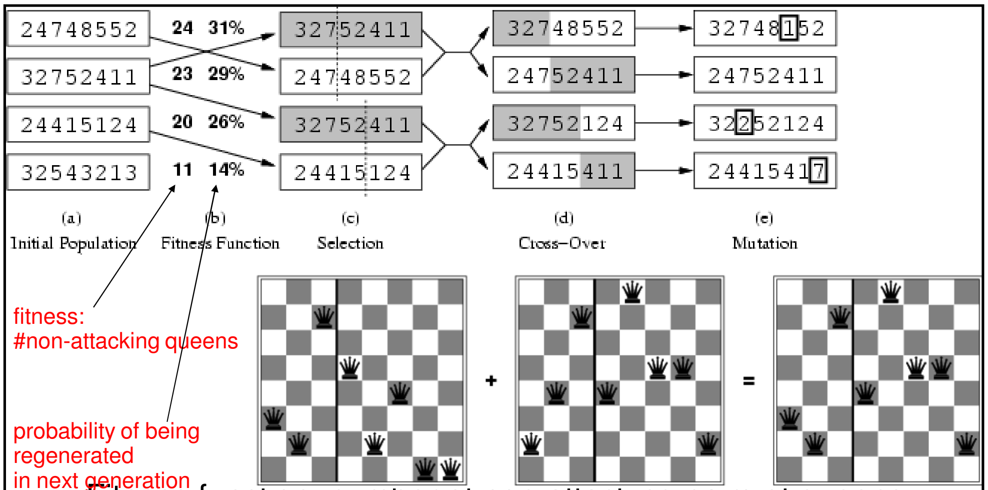
Deletion



- *Generational* GA:
  entire populations replaced with each iteration
- *Steady-state* GA:
  a few members replaced each generation



- Fitness function: number of non-attacking pairs of queens
  (min = 0, max = 8 × 7/2 = 28)



- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- P(child) = 24/(24+23+20+11) = 31%
- P(child) = 23/(24+23+20+11) = 29% etc

Creativity in GA

- ✓ GAs can be thought of as a simultaneous, parallel hill climbing search --- The population as a whole is trying to converge to an optimal solution
- ✓ Because solutions can evolve from a variety of factors, very novel solutions can be discovered