Genetic Algorithms

Patricia J Riddle

Computer Science 760

Motivation

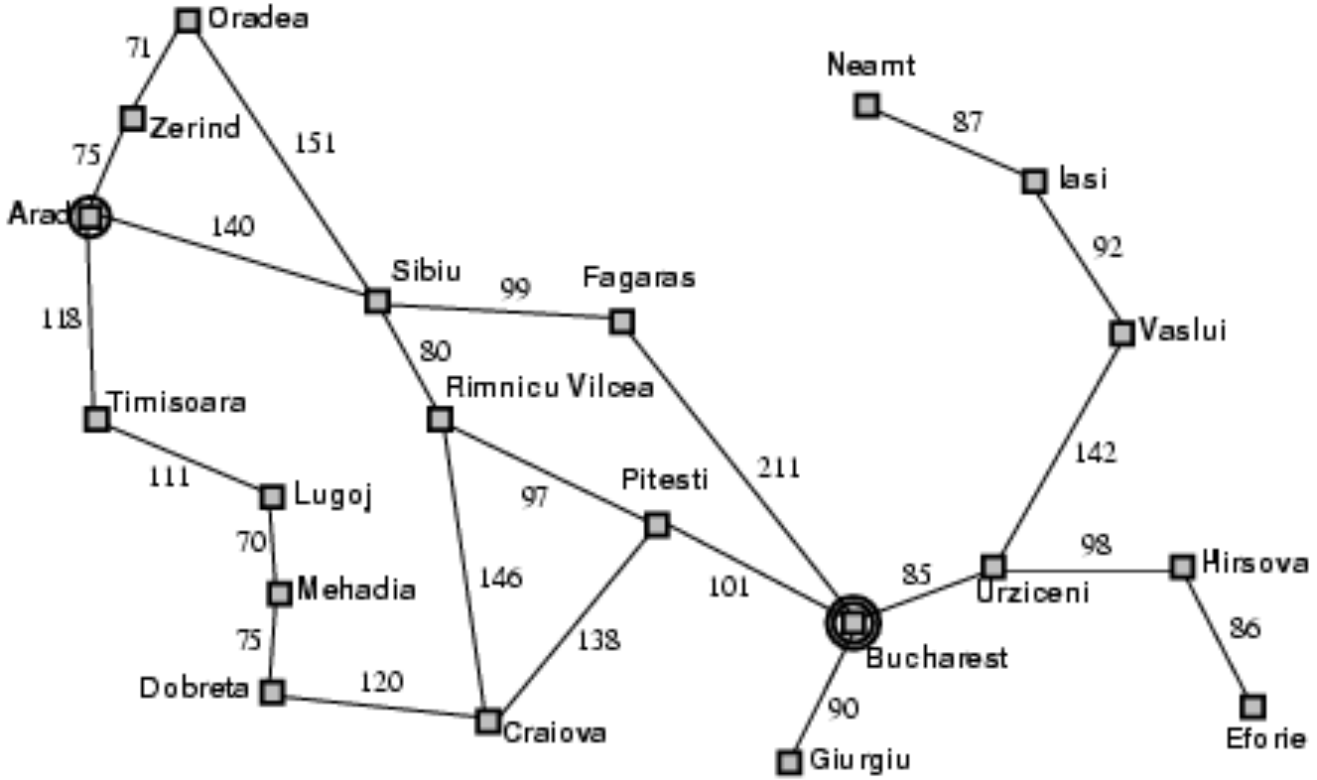
Analogy to biological evolution

GAs generate successor hypotheses by repeatedly **mutating and recombining** parts of the best currently known hypotheses

The collection of hypotheses, **population**, is updated by replacing some fraction of the population by offspring of the fittest current hypotheses

Generate-and-test beam-search of hypotheses in which variants of the fittest current hypotheses are most likely to be considered next

Beam-Search Example



Straight–line distance to Bucharest

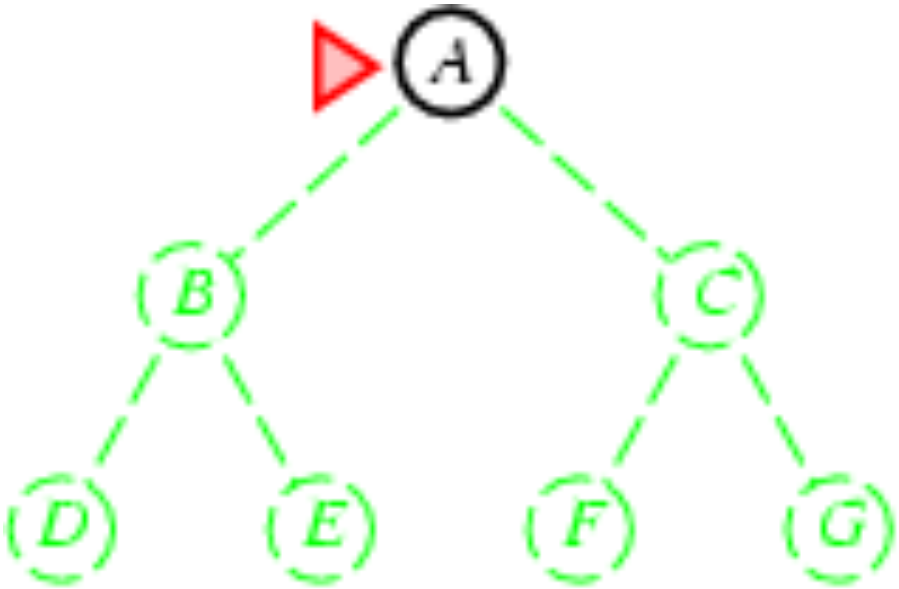
Arad	**366**
Bucharest	**0**
Craiova	**160**
Dobreta	**242**
Eforie	**161**
Fagaras	**178**
Giurgiu	**77**
Hirsova	**151**
Iasi	**226**
Lugoj	**244**
Mehadia	**241**
Neamt	**234**
Oradea	**380**
Pitesti	**98**
Rimnicu Vilcea	**193**
Sibiu	**253**
Timisoara	**329**
Urziceni	**80**
Vaslui	**199**

Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

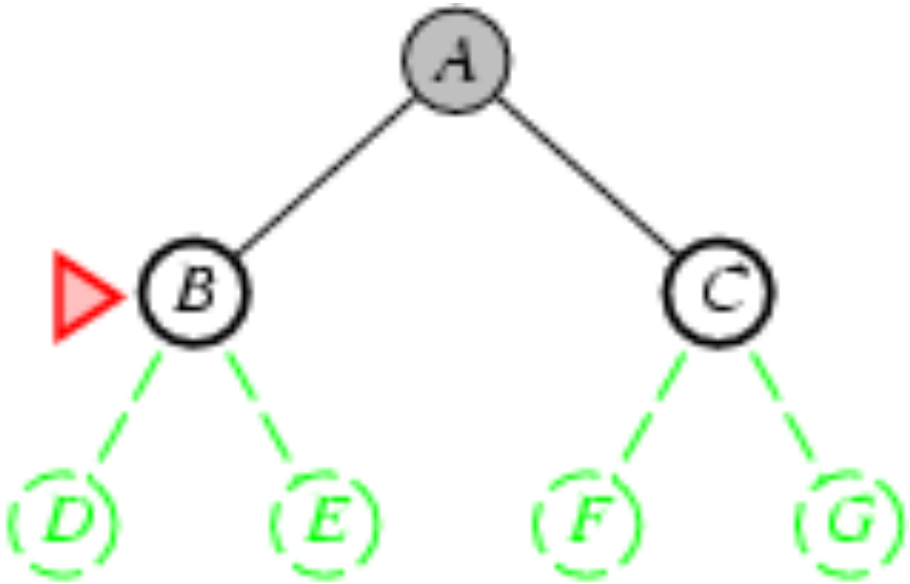


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

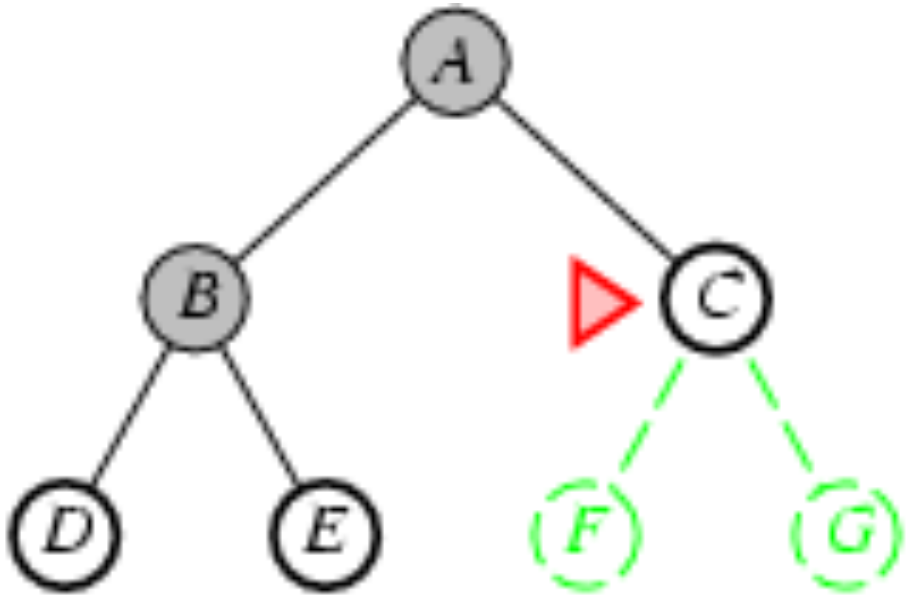


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

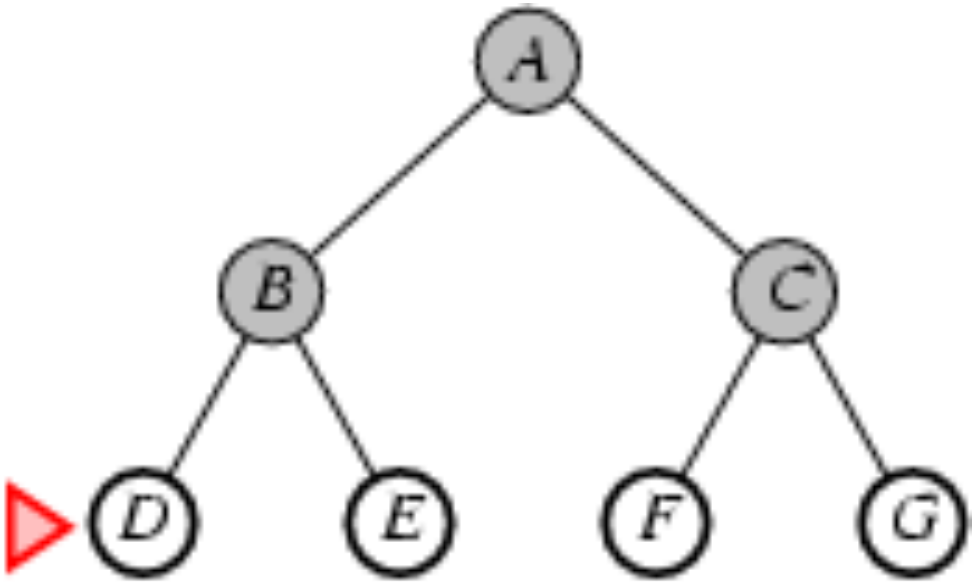


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

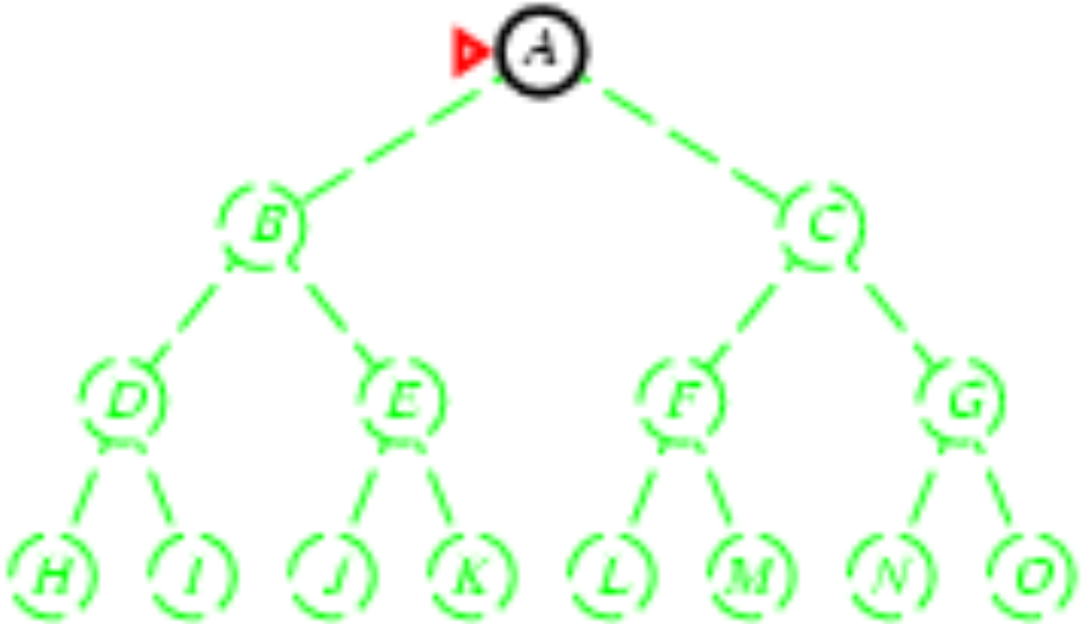


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

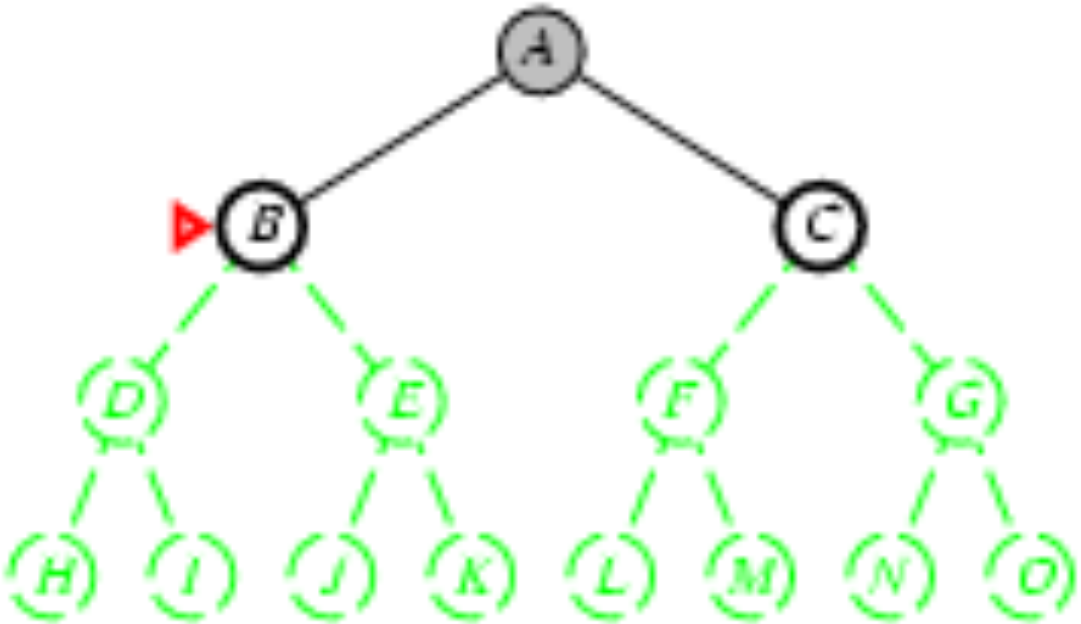


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

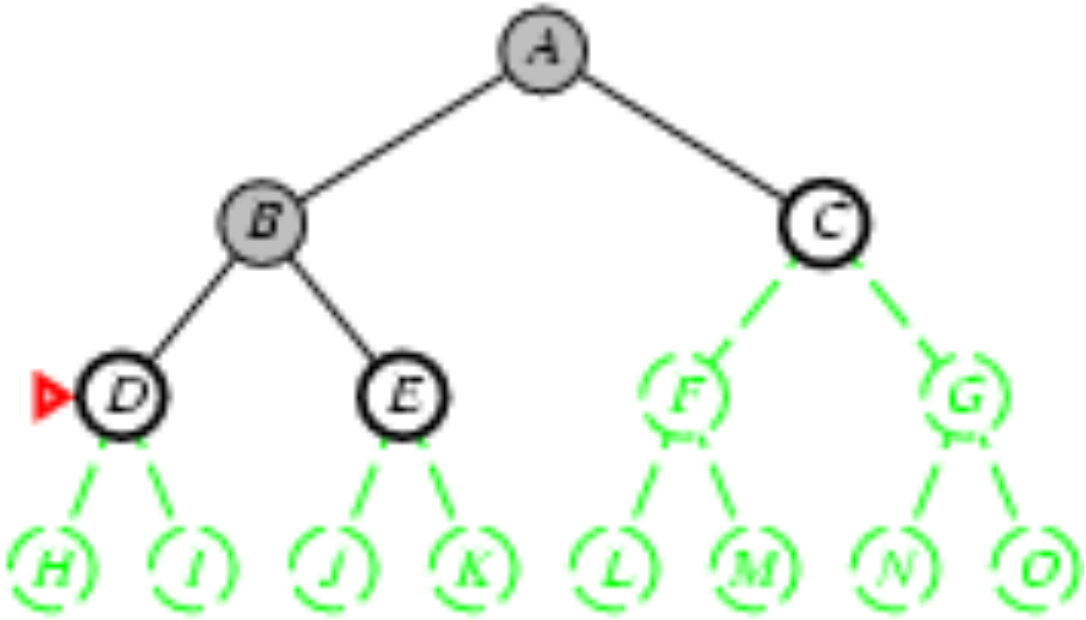


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front



Depth-first search

Expand deepest unexpanded node

Implementation:

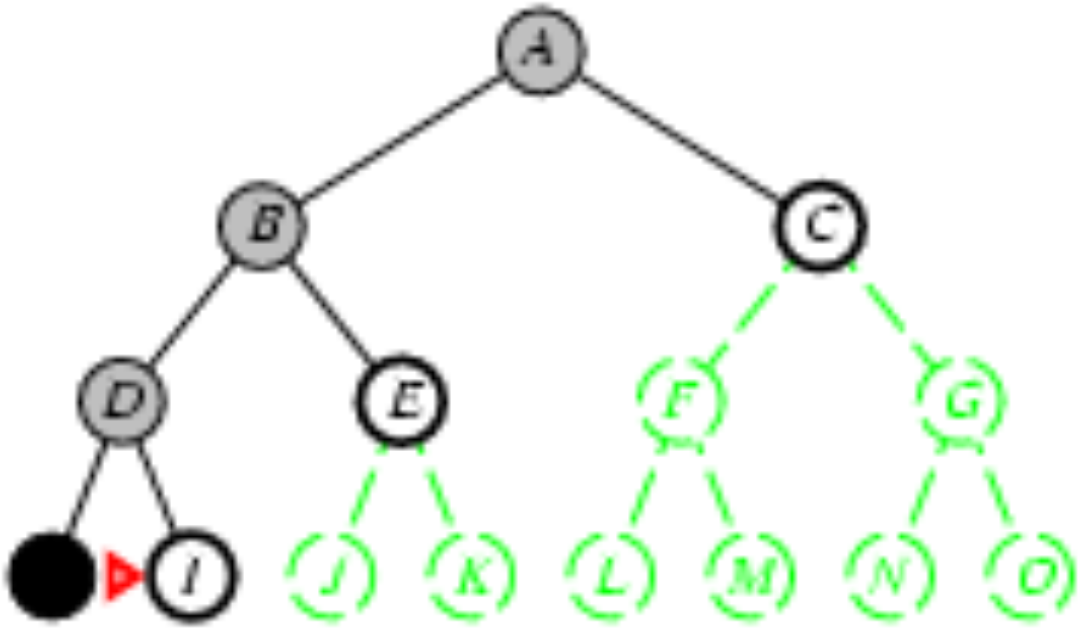
fringe = LIFO queue, i.e., put successors at front

Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

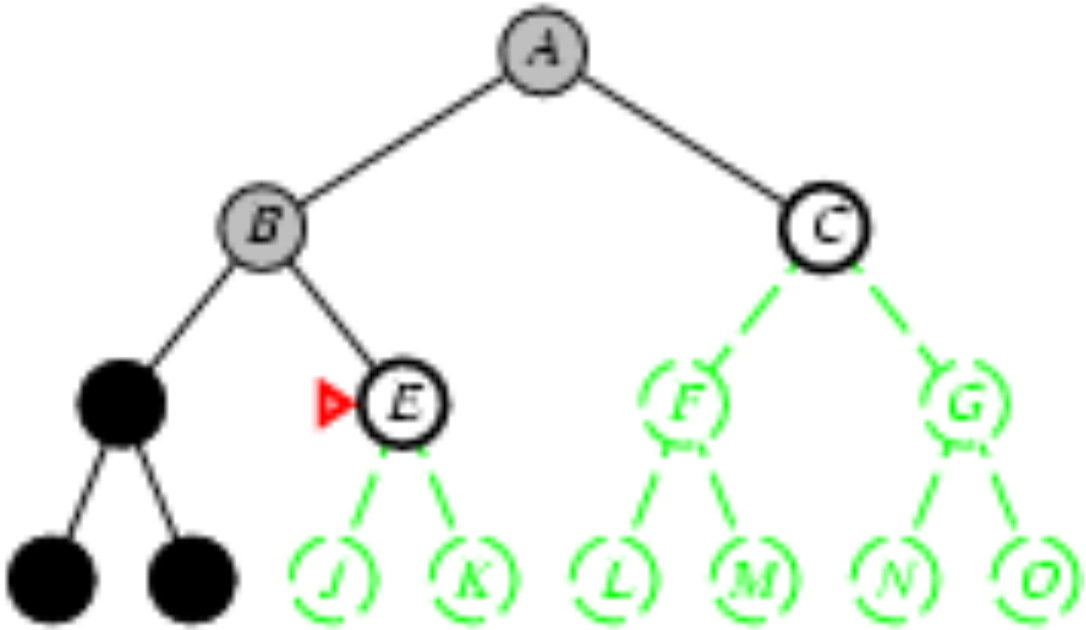


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

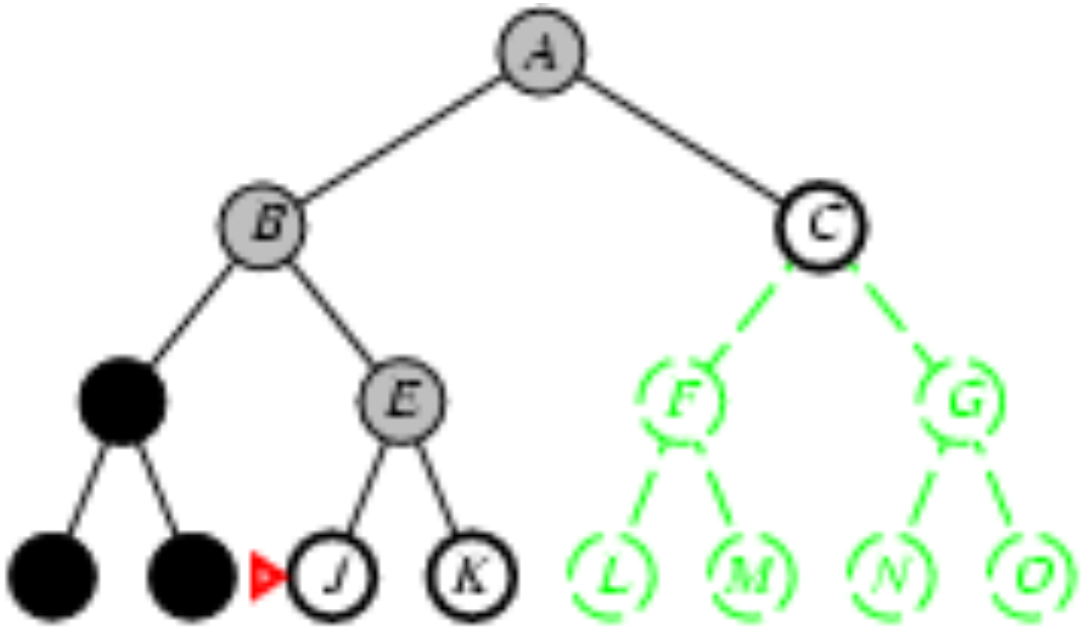


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

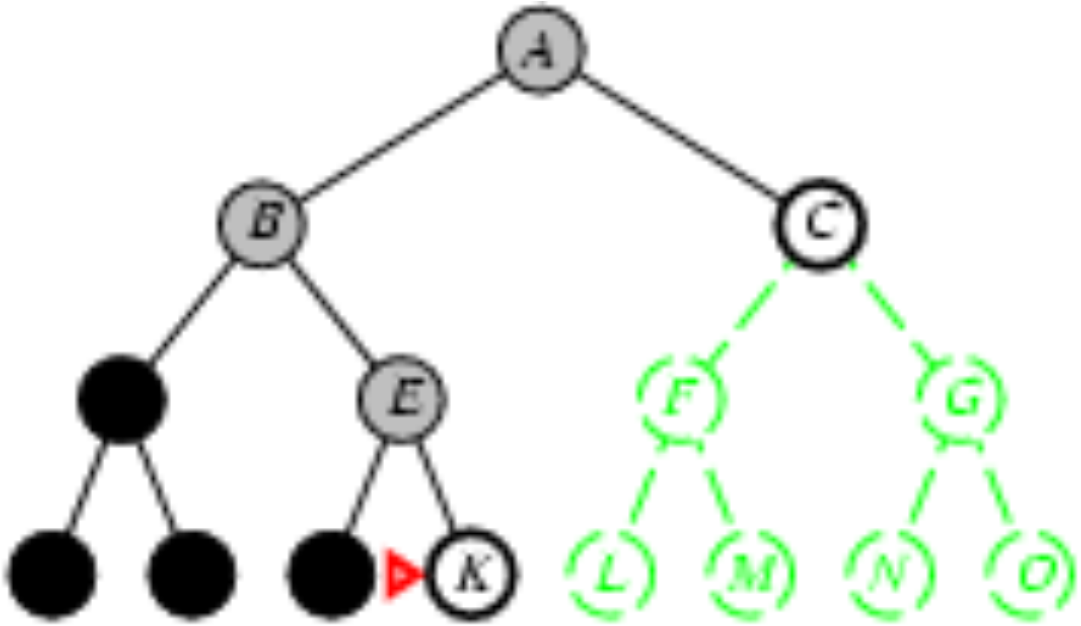


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

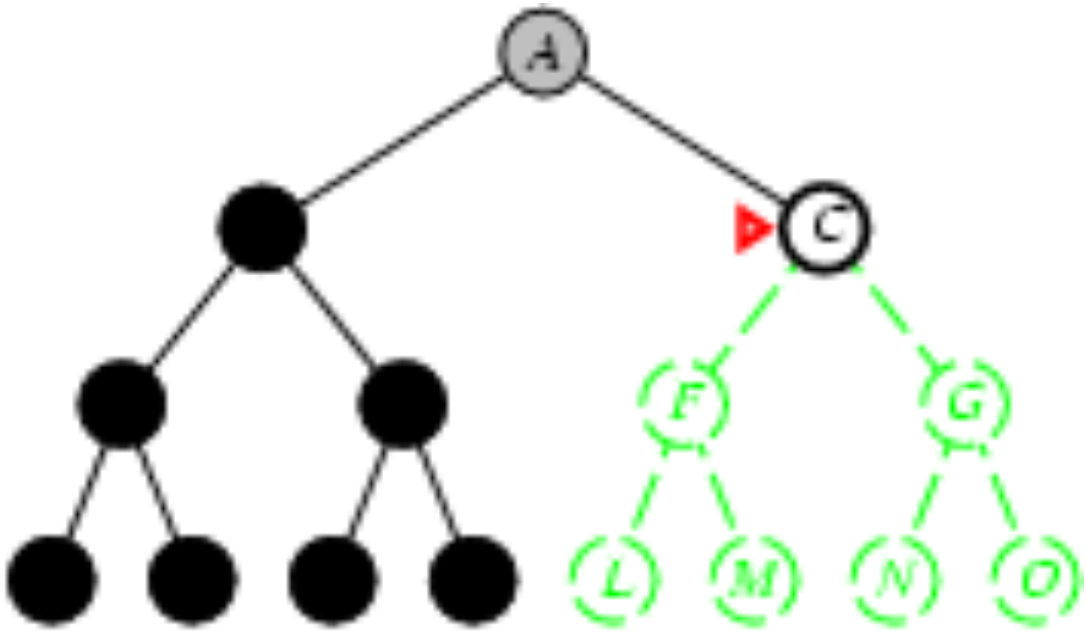


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

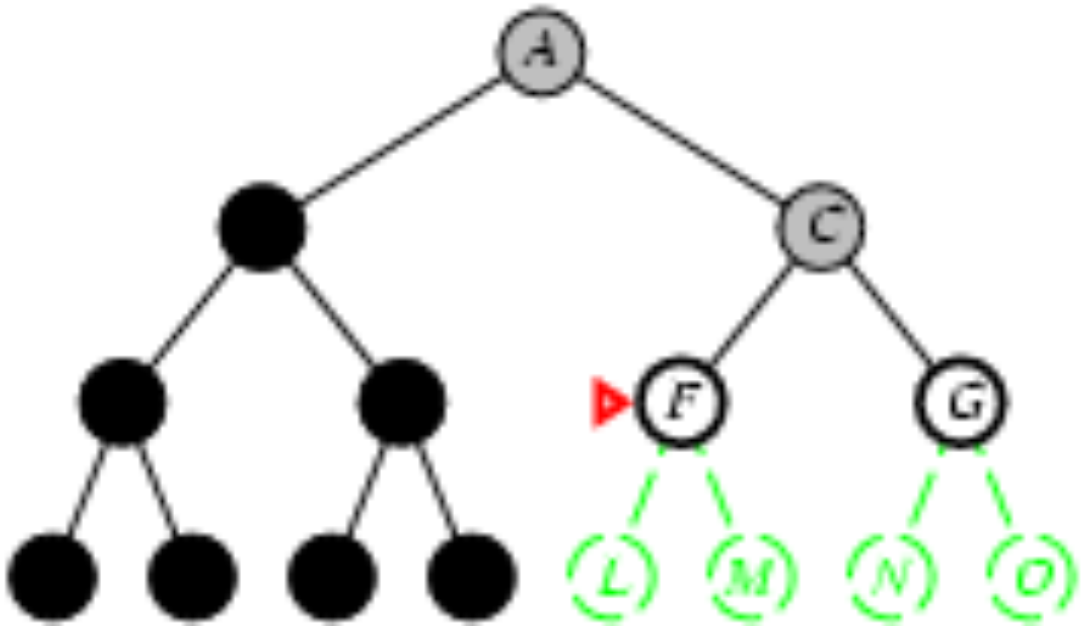


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

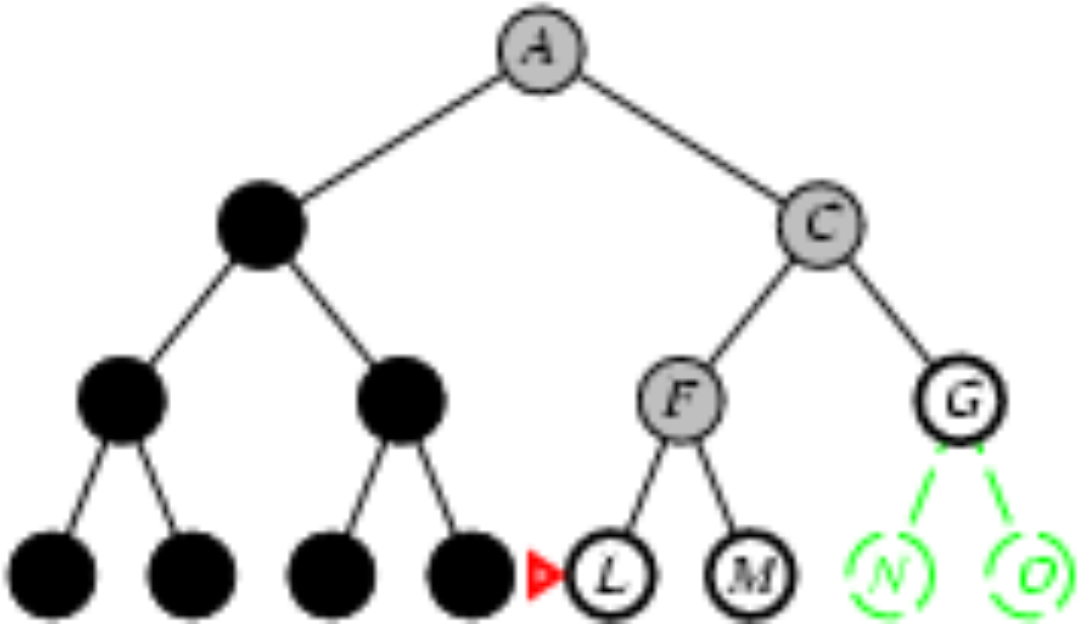


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

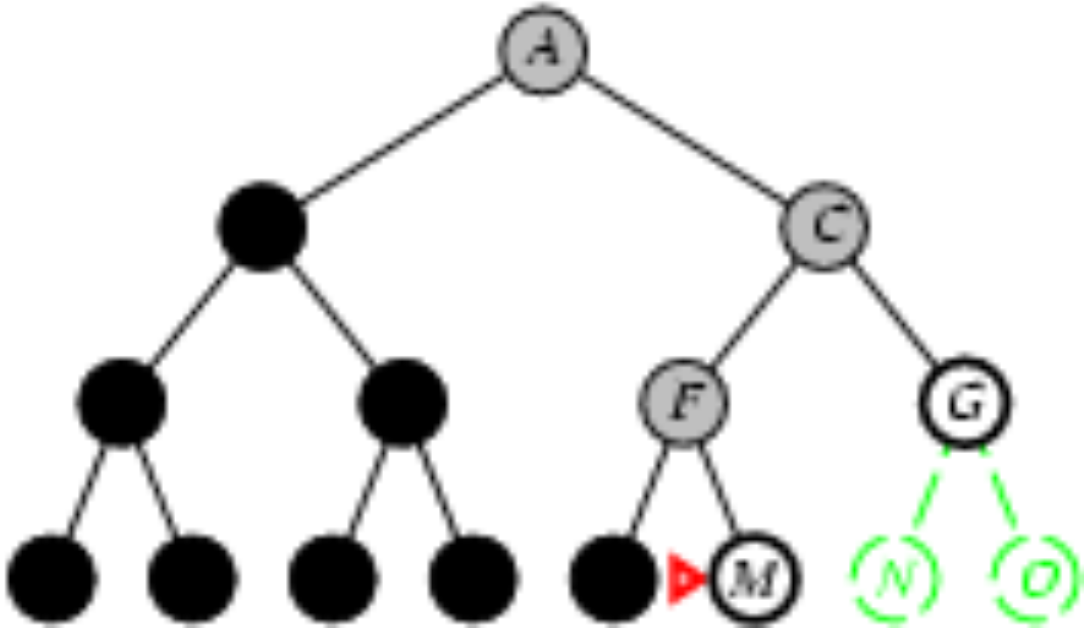


Depth-first search

Expand deepest unexpanded node

Implementation:

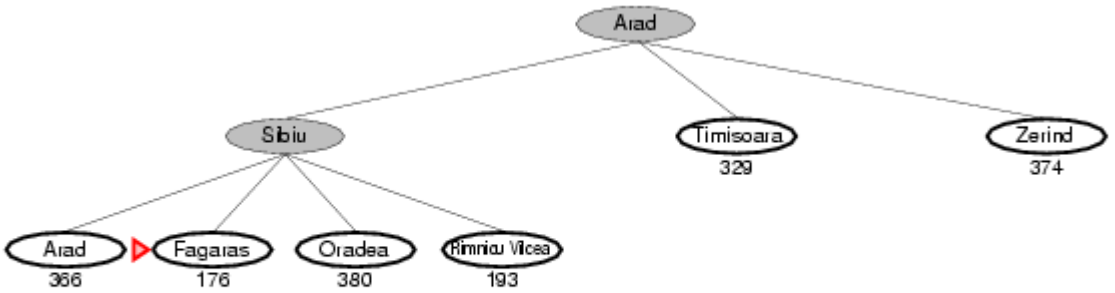
fringe = LIFO queue, i.e., put successors at front



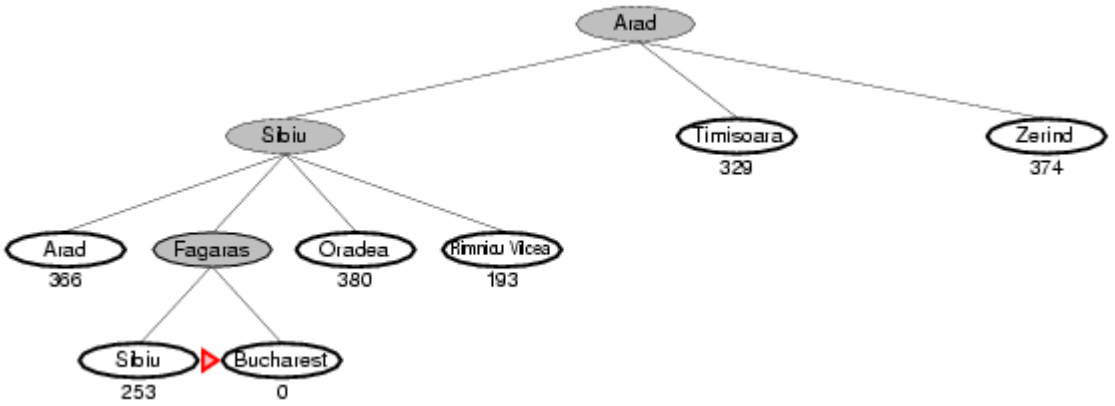
Greedy best-first search example

Greedy best-first search example

Greedy best-first search example



Greedy best-first search example



Beam Search

Beam Search (n=2)

Arad

Sibiu, **Timisoara**, Zernid

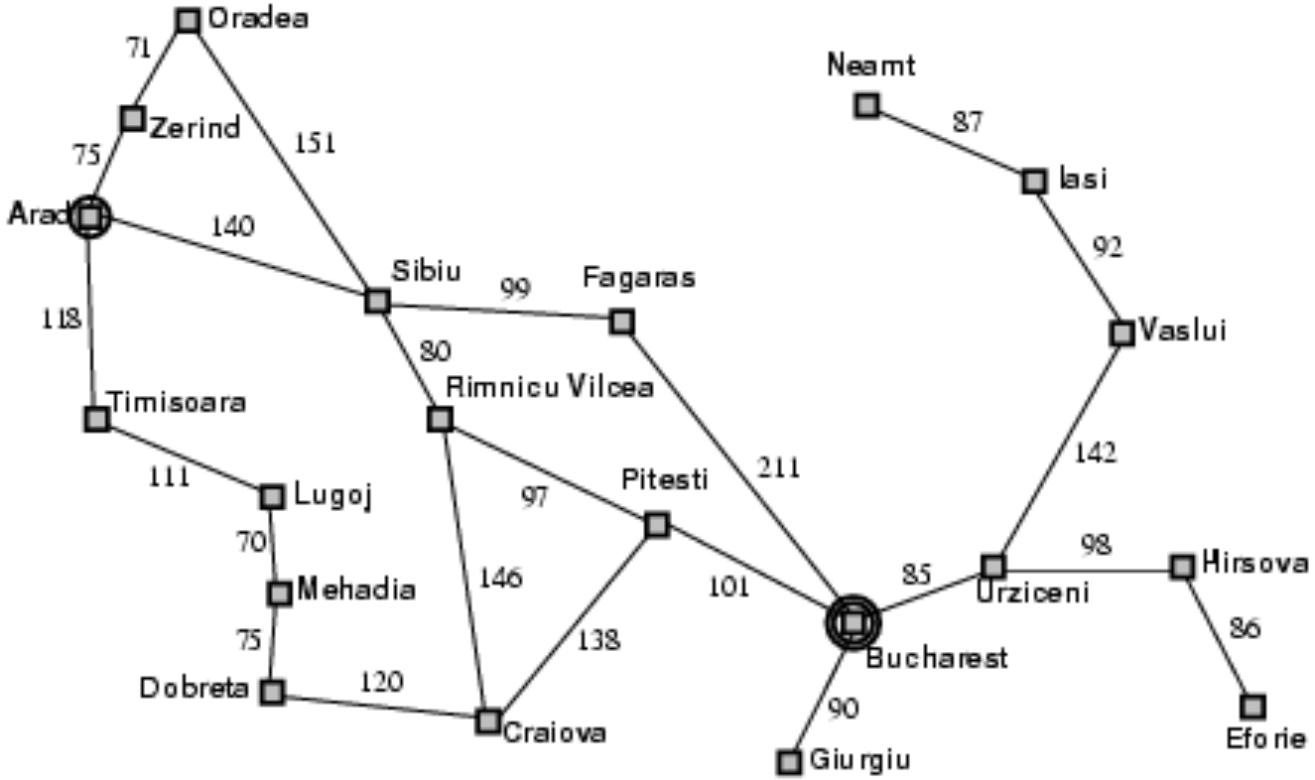
Arad, **Fagaras**, Orades, **Rimnicu Vilcea**

Bucharest

Only stores beam width at each level

When beam width is infinite = breadth first search

Beam-Search Example



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**

Back to GAs

- **Generate-and-test beam-search** of hypotheses in which variants of the fittest current hypotheses are most likely to be considered next

Popularity

Evolution is known to be a successful, robust method of adaptation within biological systems

GAs can search spaces of hypotheses containing complex interacting parts, where the **impact** of each part on overall hypothesis fitness may be **difficult to model**!!!!!!!

Genetic algorithms are **easily parallelized** and can take advantage of the decreasing costs of powerful computer hardware.

Fitness & Population

The ***best*** hypothesis is defined as the one that optimizes a predefined numerical measure called the ***fitness function.***

Fitness could be

- accuracy of the hypothesis over the training data or
- number of games won by the individual when playing against other individuals in the current population

The algorithms iteratively update the pool of hypotheses (i.e., ***population***)

General Method

On each iteration:

1. All members of the population are **evaluated** according to the fitness function.

2. A new population is generated by **probabilistically selecting the most fit individuals** from the current population.

3. Some of these individuals are carried forward into the next generation population intact.

4. Others are used to create new offspring individuals by applying genetic operations such as **crossover and mutation**.

Genetic Algorithm

GA($Fitness$, $Fitness\_threshold$, p, r, m)

Fitness: A function that assigns an evaluation score, given a hypothesis.

$Fitness\_threshold$: A threshold specifying the termination criterion.

p: The number of hypotheses to be included in the population.

r: The fraction of the population to be replaced by Crossover at each step.

m: The mutation rate.

- *Initialize population:* $P \leftarrow$ Generate p hypotheses at random
- *Evaluate:* For each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\_threshold$ do

  Create a new generation, P_S:

  1. *Select:* Probabilistically select $(1 - r)p$ members of P to add to P_S. The probability $\Pr(h_i)$ of selecting hypothesis h_i from P is given by

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

  2. *Crossover:* Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P, according to $\Pr(h_i)$ given above. For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add all offspring to P_s.
  3. *Mutate:* Choose m percent of the members of P_s with uniform probability. For each, invert one randomly selected bit in its representation.
  4. *Update:* $P \leftarrow P_s$.
  5. *Evaluate:* for each h in P, compute $Fitness(h)$
- Return the hypothesis from P that has the highest fitness.

Algorithm Properties

Inputs:

- The **fitness function**,
- The **size of the population,**
- **Threshold** defining an acceptable level of fitness **for terminating** the algorithm
- Parameters that determine how successor populations are generated
  - the **fraction of the population to be replaced** each generation, and
  - the **mutation rate**

Population Models

- Generational Model –GGA
  - Each individual survives 1 generation
- Steady State Model - SSGA
  - One offspring generated per generation – one member replaced
- Generation Gap
  - A proportion of the population replaced
  - 1.0 for GGA and 1/pop_size for SSGA

Probability of Inclusion

Probability of inclusion of hypothesis, h_i , in the next generation,

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

The probability that a hypothesis will be selected is **proportional to its own fitness** and **inversely proportional to the fitness of the other** competing **hypotheses** in the current population

Hypothesis is chosen with replacement!!!!!!

Crossover

Additional members are generated using **crossover**

Crossover takes **two parent hypothesis** from the current population and creates **two offspring hypothesis** by recombining portions of both parents.

The parents are chosen **probabilistically** using the same formula mentioned above.

Now the new generation contains the desired number of members.

Mutation

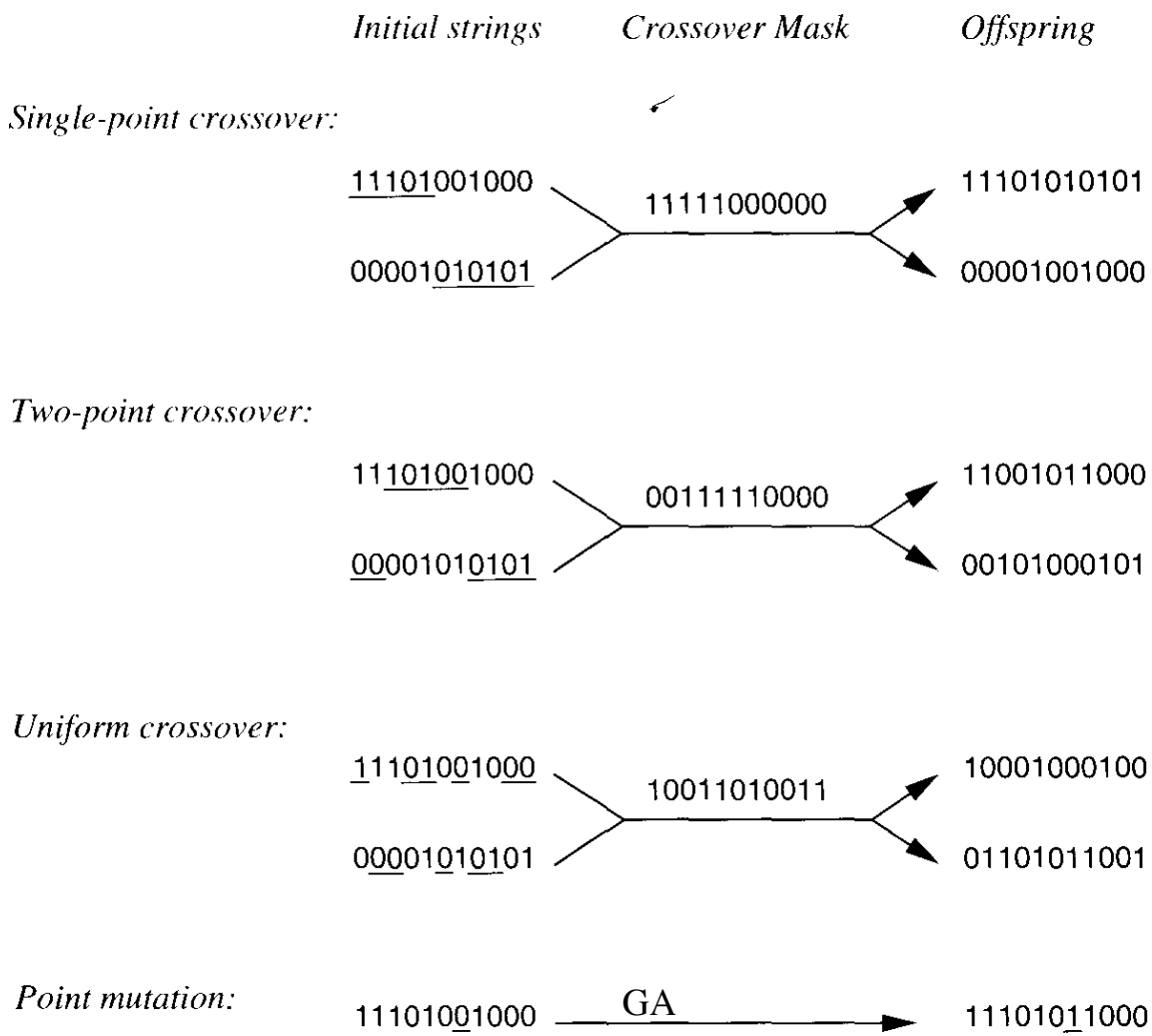
Now a certain fraction ***m*** of these members are chosen at random and random mutations are performed.

Binary Operators

- Binary Mutation
  - Flip Bit
  - Uniform
  - Non-Uniform

- Binary Crossover
  - One Point
  - Two Point
  - Uniform

Genetic Operators



Crossover & Mutation

Single-point crossover: n chosen randomly each time the crossover operator is applied

Two-point crossover: n_0 & n_1 chosen randomly each time applied

Uniform crossover: each bit chosen at random and independent of the others

Mutation: Flip one random bit (sometimes two mutation parameters)

Or chose a value randomly (e.g., chose 0 or 1)

Also can have **bitwise mutation** parameter and **genewise mutation** parameter

Some systems add new operators that do specialization or generalization

Two Mutation Parameters

- **Genewise** - Probability of choosing Gene
- **Bitwise** - Probability of Mutating a Bit
- Causes fewer individuals to be mutated a lot!! – like a macro

Other Representations

- Integer/Categorical
  - same as binary

Real Values

- Mutations
  - Uniform Mutation (Lb_i, Ub_i)
  - Non-uniform – Gaussian Mutation

- Crossovers
  - Discrete
  - Arithmetic $c_i = \alpha x_i + (1-\alpha) y_i \; where \; 0 \le \alpha \le 1$

Permutations

- Like Traveling Salesman
  - Need to make children admissible
  - Must change at least two values

Permutation Mutations

- Mutation {2,4}
  - Insert 12345 => 12435
  - Swap 12345 => 14325
  - Inversion 12345 => 14325
  - Scramble 12345 => 13425

Permutation Crossovers

- Order 1 crossover
- PMX crossover
- Cycle crossover
- Edge Recombination
- Multi-parent Recombination
  - (non Biological)

Order 1 crossover

- Preserve relative order
- Choose random segment from P1
- Copy the rest from P2
  - In order starting after the chosen part and wrapping around
  - 123456789 => 382456719
  - 937826514

PMX (partially Mapped)

- Choose random segment from P1
- Look for elements that have not been copied
- Fill the rest from P2

- 123456789 => 932456718
- 937826514

Cycle Crossover

- Identify Cycles
- Copy Alternate Cycles into Offspring

- 123456789 => 137426589
- 937826514 9238567 14

What is the hypothesis space?

Randomized, parallel, generate-and-test, beam search for an hypothesis that performs well according to the fitness function.

Hypothesis Space Search

Randomized beam search method to seek maximally fit hypothesis

GAs vs, Backpropagation (GD)

- GD moves smoothly from one hypothesis to a new one which is very similar
- GAs move much more abruptly - replacing a parent with an offspring that maybe radically different

GA is therefore less likely to fall into the same kind of local minima that plague GD

GAs have their own problems – early convergence

Early Convergence

Some individual is more fit and so copies of this individual and very similar individuals quickly take over

This lowers the diversity of the population and slows further progress by the GA - in worse case down to mutation

Fitness Function Selection

Fitness proportionate selection: ratio of fitness to the fitness of other members of the current population (**Roulette wheel**), μ – population size

$$E(n_i) = \mu \bullet \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

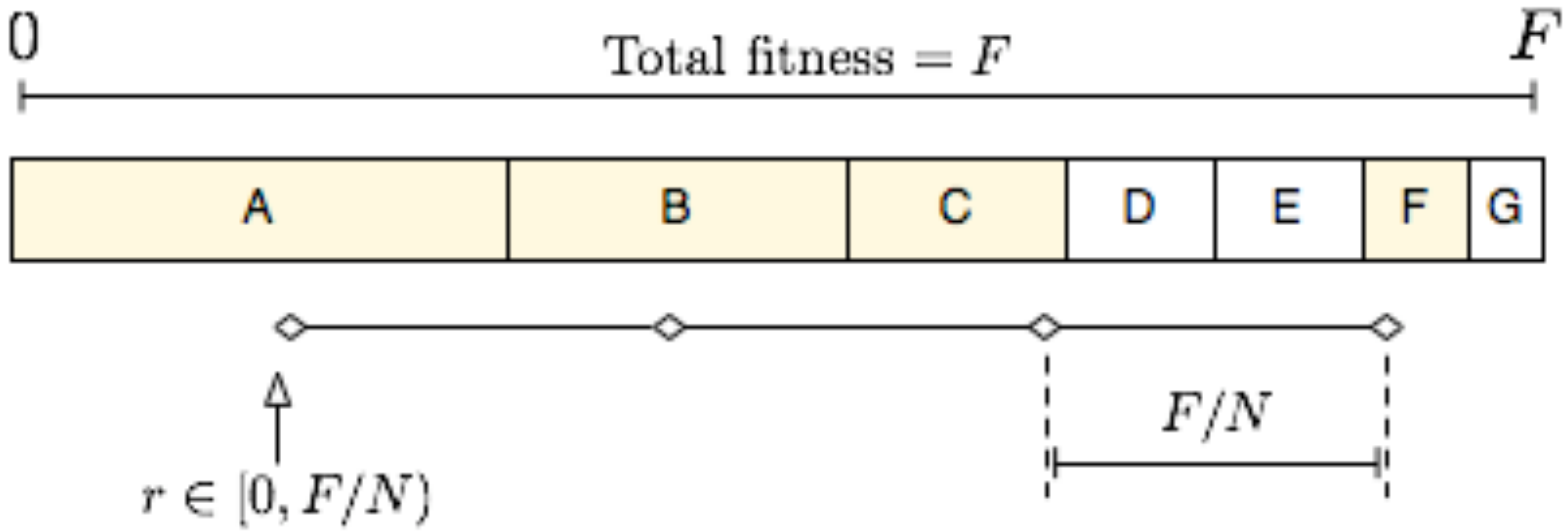
Baker's stochastic universal sampling (SUS)

N evenly spaced arms on wheel

guarantees

$$floor(E(n_i)) \le n_i \le ceil(E(n_i))$$

Baker's stochastic universal sampling



Rank based Fitness

- Sorted by fitness.
- The probability that a hypothesis will be selected is then proportional to its rank.
- Fittest = population size = μ and least-fit = 1
- Relative fitness, not absolute

	Fitness	Rank	P-Fp
A	1	1	0.1
B	5	3	0.5
C	4	2	0.4

Linear Ranking

Parameterized by factor s where $1.0 < s \leq 2.0$

$$P(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

	Fitness	Rank	P-FP	P-L(s=2)	P-L(s=1.5)
A	1	1	0.1	0	0.167
B	5	3	0.5	0.67	0.5
C	4	2	0.4	0.33	0.33

Exponential Ranking

- Linear Ranking has limited selection pressure

$$P(i) = \frac{1 - e^{-i}}{c}$$

- Exponential can allocate more than 2 copies of the fittest individual

Tournament Selection

- Pick k members at random then select best of these
- Can also use ρ to determine whether the fittest gene wins
  - some predefined probability, p, the more fit is selected and with probability (1-p) the less fit is selected
- No longer uses global population statistics (better for parallelization)

Tournament Probabilities

- Probability of selecting i will depend on
  - Rank of i
  - Size of sample k
    - Higher k increases selection pressure
  - Whether contestants are picked with replacement
    - Without replacement increases selection pressure – Why??

- For $k = 2$, time for fittest individual to take over population is the same as linear ranking with

$$s = 2 \bullet p$$

Survivor Selection

- Survivor selection can be divided into two approaches:
  - Age-Based Selection
    - e.g. SGA
    - In SSGA can implement as “delete-random” (not recommended) or as first-in-first-out (a.k.a. delete-oldest)
  - Fitness-Based Selection
    - Using one of the methods above

Two Special Cases

- Elitism
  - Always keep at least one copy of the fittest solution so far
  - Widely used in both population models (GGA, SSGA)
- GENITOR: a.k.a. “delete-worst”
  - From Whitley’s original Steady-State algorithm (he also used linear ranking for parent selection)
  - Rapid takeover : use with large populations or “no duplicates” policy

Elitism

The best chromosome (or a few best chromosomes) is copied to the population in the next generation. The rest are chosen in the classical way.

Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution to date.

A variation is to eliminate an equal number of the worst solutions, i.e. for each "best chromosome" carried over a "worst chromosome" is deleted.

Niching Solutions (to avoid early convergence)

Use tournament or rank selection

No incest – (not used much)

Restrict the kinds of individuals allowed to recombine - multiple clans with festivals

- related approach is to spatially distribute individuals and allow only nearby individuals to recombine

Crowding

Crowding is a steady-state technique where the current population is sampled to find an individual that is "close" to the new offspring.

This closest individual is then replaced.

De Jong reported a noticeable reduction in allele loss even when the number of individuals sampled was 2, the minimum sample size that distinguishes crowding from random replacement.

Clearing Procedure

- Minimal distance between all members of the population

- If a new child is added within the radius, the dominating child is kept.

Sharing

Sharing is similar to crowding in that individuals that are "close" compete with each other, but it is used for generational GAs.

Fitness sharing causes individuals that are similar to many others to have their fitness reduced, making them less likely to be selected.

This preserves diversity by favouring unusual individuals, however Ursam has shown that the effectiveness of sharing is "sensitive to the range of fitness values" and can lead to adverse results simply by adding a constant to the fitness function.

Bloat - The Other Problem

Problem with variable length representations

Longer individuals usually have a better chance of a higher fitness

What happens when there is no selection pressure? – absorbing boundaries

Crossover or Mutation?

- Which one is better or necessary?
  - it depends on the problem, but
  - in general, it is good to have both
  - both have another role
  - mutation-only-EA is possible, xover-only-EA would not work
- Why?

Old Thoughts

Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

Old Thoughts Continued

There is co-operation AND competition between them

• Crossover is explorative, it makes a *big* jump to an area somewhere “in between” two (parent) areas

• Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of) the parent

More Thoughts

- Only crossover can combine information from two parents

- Only mutation can introduce new information (alleles)

  (ONLY AT BIT LEVEL)

More Thoughts Continued

- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0's on first bit in the population, ?% after performing *n* crossovers)
  - YES or NO?

- To hit the optimum you often need a 'lucky' mutation

Cameron Skinner's thesis

- Clearly the accepted wisdom that recombination is the dominant component of the genetic algorithm is incorrect

Cameron Skinner's Thesis

Discovery & Retention

Crossover & Mutation???

The more disruptive an operator, the better it is at discovery

All operators have higher destruction probabilities than discovery probabilities

Random search outperforms all genetic operators, once they are biased towards a correct solution.

Cameron Skinner's Thesis II

Better Theoretical Analysis Than Schema Theorem

Seed Pool instead of Mutation!!!
(best idea since sliced bread)

Seed Pool Idea

Creature Demo

Population Evolution

Can we mathematically characterise the evolution over time of the population within a GA

Schema theorem of Holland

Schema is a string composed of 1s 0s and *s

* is "don't care"

Schema 0*10 represents the set of bit strings 0010, 0110

The bit string 0010 represents 2^4 different schemas

Schemas

Population of bit strings can be viewed by the set of schemas it represents and the number of individuals associated with each schema

m(s,t) is the number of instances of schema s at a time t

Schema Theorem

- Determine the expected value of schema

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)(1 - p_c \frac{d(s)}{l-1})(1 - p_m)^{o(s)}$$

- Average fitness of individuals of schema s

$$\hat{u}(s,t) = \frac{\sum_{h \in s \cap p_t} f(h)}{m(s,t)}$$

Terms

$\bar{f}(t)$ Average fitness of all individuals in the population at time t

$h \in s \cap p_t$ indicates an individual h is both a representative of schema s and in the population at time t

P_c, probability that the single-point crossover will be applied

Terms 2

P_m, probability that the mutation operator will be applied

o(s) is the number of defined bits

d(s) is the distance between the leftmost and rightmost defined bits

l is the length of the bitstrings

Schema Theorem Intuition

Whether an individual representing schema **s** at time **t** will be selected for time **t+1**, or still represent **s** after crossover, or still represent **s** after mutation

Effects of mutation increase with the number of defined bits

Effects of crossover increase with the distance between defined bits

More fit schemas will grow in influence, especially schemas with a small number of defined bits and especially when these defined bits are near each other in the bit string

Problems with Schema Theorem

Incomplete because it **fails to take into** account the **positive effects** of crossover and mutation (i.e., only gives lower bound)
- numerous more recent theoretical analyses

That is why Cam's analysis is better.

Representing Hypotheses

Hypothesis in GAs are often represented by bit strings, which are easily manipulated by crossover and mutation.

These can be quite complex

a set of if-then rules

Representation Example

Attribute: ***Outlook***

- Values: ***Sunny***, ***Rainy***, ***Overcast***

Use a bit string of length 3, where each position corresponds to one of the values. Placing a 1 in some position indicates that the attribute is allowed to take on the corresponding value.

- 010 represents ***Outlook = Overcast***
- 011 represents ***Outlook = Overcast or Rainy***

More Representations

Conjunctions of constraints can be represented by concatenation.

So ***011 10*** represents ***Outlook = Overcast or Rainy and Wind = Strong***

Postconditions can be represented in the same way.

111 10 10 represents ***If Wind = Strong then PlayTennis = Yes***.

Notice that ***111*** represents the "don't care" condition on Outlook

Fixed length bit strings of rules

Sets of rules can be represented by concatenating single rules, but then are not fixed length!

Representations III

It is best if every syntactically legal bit string represents a well-defined hypothesis

So 111 10 11 represents a rule whose postcondition does not constrain the target PlayTennis. To avoid this

- allocated just one bit to PlayTennis to indicate Yes or No, or
- alter the genetic operators so they explicitly avoid constructing such bit strings, or
- assign them a very low fitness (only works if there are only a few).

Some GAs represent hypothesis as symbolic descriptions rather than bit strings (more later).

GABIL System

GABIL uses a GA to learn boolean concepts represented by a disjunctive set of propositional rules.

Comparable in generalization accuracy to C4.5 (C4.5 is state of the art decision tree algorithm)

r, fraction replaced by crossover, was 0.6

m, mutation rate, was 0.001

p, population size varied from 100 to 1000, depending on the task

GABIL Representation

Each hypothesis is a disjunctive set of propositional rules

Conjunction of constraints on fixed set of attributes - bit string representations of individual rules

So the hypothesis consisting of the two rules follows:

- IF a_1= T $\wedge$ a_2 = F then c = T $\vee$ IF a_2=T then c=F
- 10 01 1 11 10 0

The length of the bit string grows with the number of rules. This causes a modification to the crossover operator.

GABIL Genetic Operators

Same mutation operator

Crossover occurs only between like sections of the bit strings

Standard extension to two-point crossover

- Two crossover points are chosen at random in the first parent string
- Calculate d_1 (d_2), the distance from the leftmost (rightmost) of the crossover points to the rule boundary immediately to its left
- Crossover points are randomly chosen in the second parent with the constraint that they must have the same d_1 and d_2 values

Genetic Operators Example

H_1: 10 01 1 11 10 0

H_2: 01 11 0 10 01 0

If the crossover points for the 1st parent are <1,8> then the allowable crossover points for the second parent are <1,3>, <1,8> and <6,8>

If happen to choose <1,3> then the two offspring would be:

H_3: 11 10 0 and

H_4: 00 01 1 11 11 0 10 01 0

All bit strings generated in this fashion represent well-defined rule sets

GABIL Fitness Function

Fitness(h) = $(\text{correct}(h))^2$,

Where ***correct(h)*** is the percent of all training examples correctly classified by hypothesis ***h*** (i.e., accuracy)

GABIL Extensions

Two new genetic operators

AddAlternative - generalises constraints by changing a 0 to a 1

In an attribute substring 10010 becomes 10110

This operator was applied with probability .01

DropCondition

DropCondition - performs more drastic generalisation step by replacing all bits for a particular attribute by a 1

In an attribute substring 10010 becomes 11111

This operator was applied with probability .60 **(this is very high!)**

The addition of these operators increased accuracy from 92.1% to 95.2% on a range of datasets

Evolving Search Methods

Even tried new attributes AA and DC specifying whether these operators can apply to these hypothesis

Worked better on some datasets and worse on others

In this way GAs can be used to evolve their own hypothesis search methods – **meta-learning**

Genetic Programming

Form of evolutionary computation where the individuals are computer programs instead of bit strings

Typically represented by trees corresponding to **parse trees** of the program

User must define primitive functions

Fitness is determined by **executing the program** on the training data

Crossover replaces a randomly chosen subtree from one parent with one from the other

Program Tree

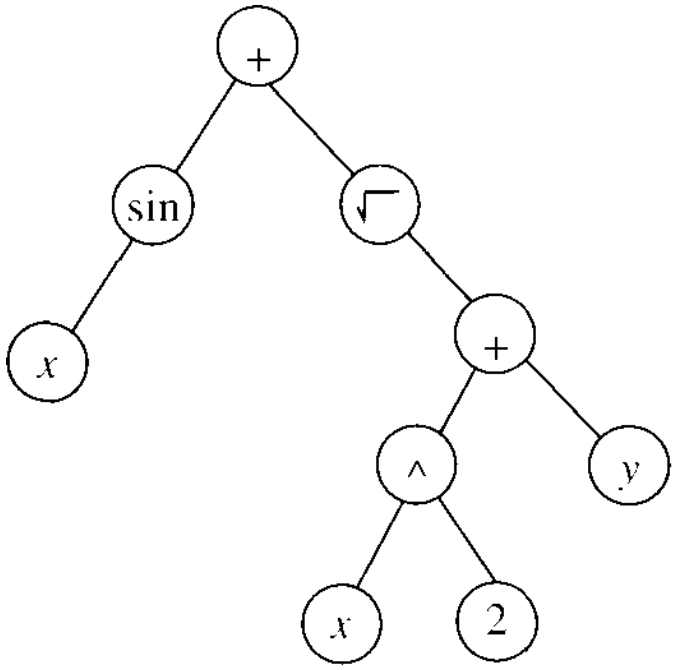


FIGURE 9.1
Program tree representation in genetic programming. Arbitrary programs are represented by their parse trees.

Crossover of Program Trees

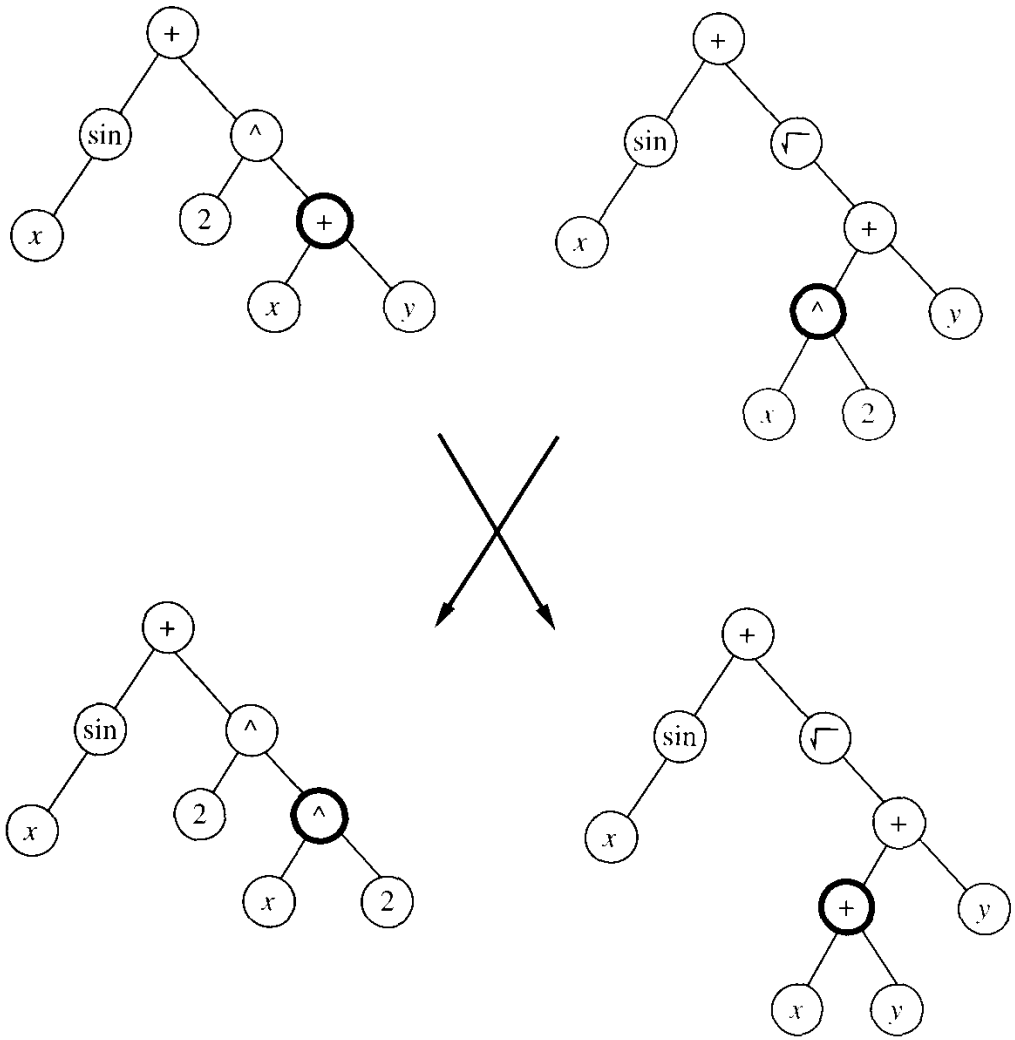


FIGURE 9.2
Crossover operation applied to two parent program trees (top). Crossover points (nodes shown in bold at top) are chosen at random. The subtrees rooted at these crossover points are then exchanged to create children trees (bottom).

Genetic Programming Example

Develop an algorithm for stacking blocks in a single stack that spells the word "**universal**" independent of the initial configuration of the blocks

In GP applications, problem representation has a significant impact on the ease of solving the problem, 3 terminal arguments:

- CS returns the name of the top block on the stack or F if there is no current stack
- TB returns the top block on the stack that is in the correct order
- NN returns the name of the next block needed or F if no blocks are needed

Imagine the difficulty if the terminal arguments returned x,y coordinates of the blocks!!!!

Primitive Functions!!!

(MS x) moves x to the top of the stack and returns T otherwise does nothing and returns F,

(MT x) if block x is in the stack it moves the top block off the stack and puts it on the table and returns T otherwise it does nothing and returns F

(EQ x y) returns T if x=y and returns F otherwise

(Not x) returns T if x=F and F if x=T

(Du x y) executes the expression x repeatedly until y returns the value T

Experiment Results

106 training examples representing a broad variety of initial block configurations and degrees of difficulty - **must contain all boundary conditions**!

Fitness of a program was the number of these examples solved, population was initialized to 300 random programs

Experiment Results II

After 10 generations -

(EQ (DU (MT CS)(Not CS))(DU (MS NN)(Not NN)))

solves all 166 problems

unstack loop followed by a stack loop

EQ used for sequencing only

was 10 runs an average or just a lucky try?

what was the variance?

Experiment Results III

GAs have been used to design electronic filter circuits and classify segments of protein molecules

- But circuit example used a population of 640,000!!!

Models of Evolution

Lamarkian Evolution - evolution over many generations was directly influenced by the experiences of individual organisms - if an individual learned during its lifetime to avoid some toxic food it could pass the trait on genetically to its offspring.

Repudiated in biological systems but this can still be used to improve the effectiveness of GAs

Like GA on ANN with ANN learning between each generation

Baldwin Effect

Evolutionary pressure to favor individuals who can learn

An individual can perform a small local search during its lifetime to maximize its fitness

It can support a more diverse gene pool and therefore more rapid evolutionary adaptation

Baldwin Effect Example

Evolving population of neural networks

- genes determined which weights could change
- weights changed during lifetime
- over generations more weights became fixed as the population optimized
- what would happen if the fitness function kept moving??

If the new weights are used by the GA it is Lamarkian, if they are not it is Baldwin

How one might parallelize a GA?

- The GA calculations are minimal. An optimization might require 1000 generations, and each generation is dominated by the cost to evaluate the fitness
  - If your fitness is related to an engineering design, this might run a simulation for each individual.

- Standard serial GA programs can handle the GA routines with ~1 ms of cpu time, while your fitness routine can parallelize the fitness evaluations.

Parallel GAs

- MPI is always a good choice if you're already familiar the language. This option would also enable GA algorithms with "islands" on heterogeneous clusters.

- Fork/wait would be very easy
  - Can be done in several languages

Parallelizing Genetic Algorithms

Coarse grain - subdivide population into **demes**, each deme is assigned a computational node, GA search performed at each node, communication and cross-fertilization across demes occurs less frequently by migration, also reduces early convergence problem

Fine grain - assign one processor per individual - recombination occurs among neighbors - neighborhood could be planar or torus

Summary

GAs **randomized parallel generate-and-test beam** search for hypothesis that optimize a predefined fitness function

Based on analogy to **biological evolution**

Diverse population of competing hypotheses, at each iteration most fit members of the population are **selected**, combined by **crossover** and subjected to random **mutation**

Summary II

GAs show how **learning** can be seen as a special case of **optimization**

learning task is finding optimal hypothesis

this suggests other optimization techniques - like **simulated annealing** - can be applied to machine learning

Summary III

GAs most commonly been applied to optimization problems outside machine learning

especially suited to learning tasks where **hypotheses are complex** and the objective to be optimized may be an **indirect function of the hypothesis** (e.g., the acquired rules to successfully control a robot).

Genetic Programming is a variant of GAs where the hypotheses are programs. Demonstrated to learn programs to simulate robot control and recognize objects in visual scenes.

References

- Introduction to Evolutionary Computing
  - A E Eiben and J E Smith
- http://www.cs.vu.nl/~gusz/ecbook/ecbook.html

- Genetics Algorithm Chapter in Tom Mitchell Book

Questions you should be able to answer

- What is the general way GAs search the space?
- What is the GA hypothesis space?
- What is the difference between crossover and mutation?
- What is early convergence and how do you stop it?
- What is the difference between Lamarkian evolution and the Baldwin effect?