# Genetic Algorithms and Local Search

Darrell Whitley
Computer Science Department
Colorado State University
Fort Collins, CO

# GENETIC ALGORITHMS AND LOCAL SEARCH

Darrell Whitley
Computer Science Department
Colorado State University
Fort Collins, CO 80523
whitley@cs.colostate.edu
(970) 491-5373

## ABSTRACT

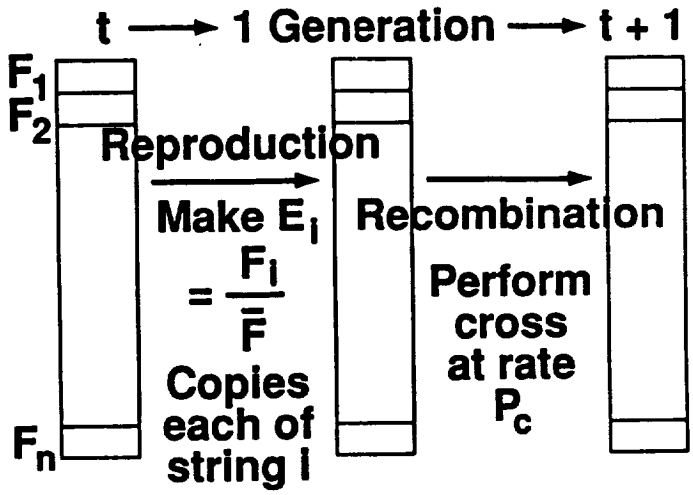
The first part of this presentation is a tutorial level introduction to the principles of genetic search and models of simple genetic algorithms. The second half covers the combination of genetic algorithms with local search methods to produce hybrid genetic algorithms. Hybrid algorithms can be modeled within the existing theoretical framework developed for simple genetic algorithms.

An application of a hybrid to geometric model matching is given. The hybrid algorithm yields results that improve on the current state-of-the-art for this problem.[1]

[1]Ross Beveridge collaborated in the geometric matching application. This work was supported by NSF grants IRI-9312748 and IRI-9503366 and by the Advanced Research Projects Agency (ARPA) under grant DAAH04-93-G-422 monitored by the U.S. Army Research Office.

## THE CANONICAL GENETIC ALGORITHM

In the canonical genetic algorithm, fitness is defined by: $f_i / \bar{f}$ where $f_i$ is the evaluation associated with string $i$, and $\bar{f}$ is the average evaluation of all the strings in the population. It is helpful to view the execution of the genetic algorithm as a two-stage process. It starts with the *current population*. Selection is applied to the current population to create an *intermediate population*. Then recombination and mutation are applied to the intermediate population to create the *next population*.




1. Generate a random population of strings.
2. Evaluate all strings in the population.
3. Reproduce (duplicate) strings according to fitness.
4. Recombine strings according to the probability of crossover.
5. Repeat steps 2 to 4.

## AN EXAMPLE OF HYPERPLANE PARTITIONING

A function over a single variable is plotted as a one-dimensional space, with function maximization as a goal. The hyperplane 0****...** spans the first half of the space and 1****...** spans the second half of the space. Since the strings in the 0****...** partition are on average better than those in the 1****...** partition, we would like the search to be proportionally biased toward this partition. In the second graph the portion of the space corresponding to **1**...** is shaded, which also highlights the intersection of 0****...** and **1**...**, namely, 0*1*...**. Finally, in the third graph, 0*10**...** is highlighted.

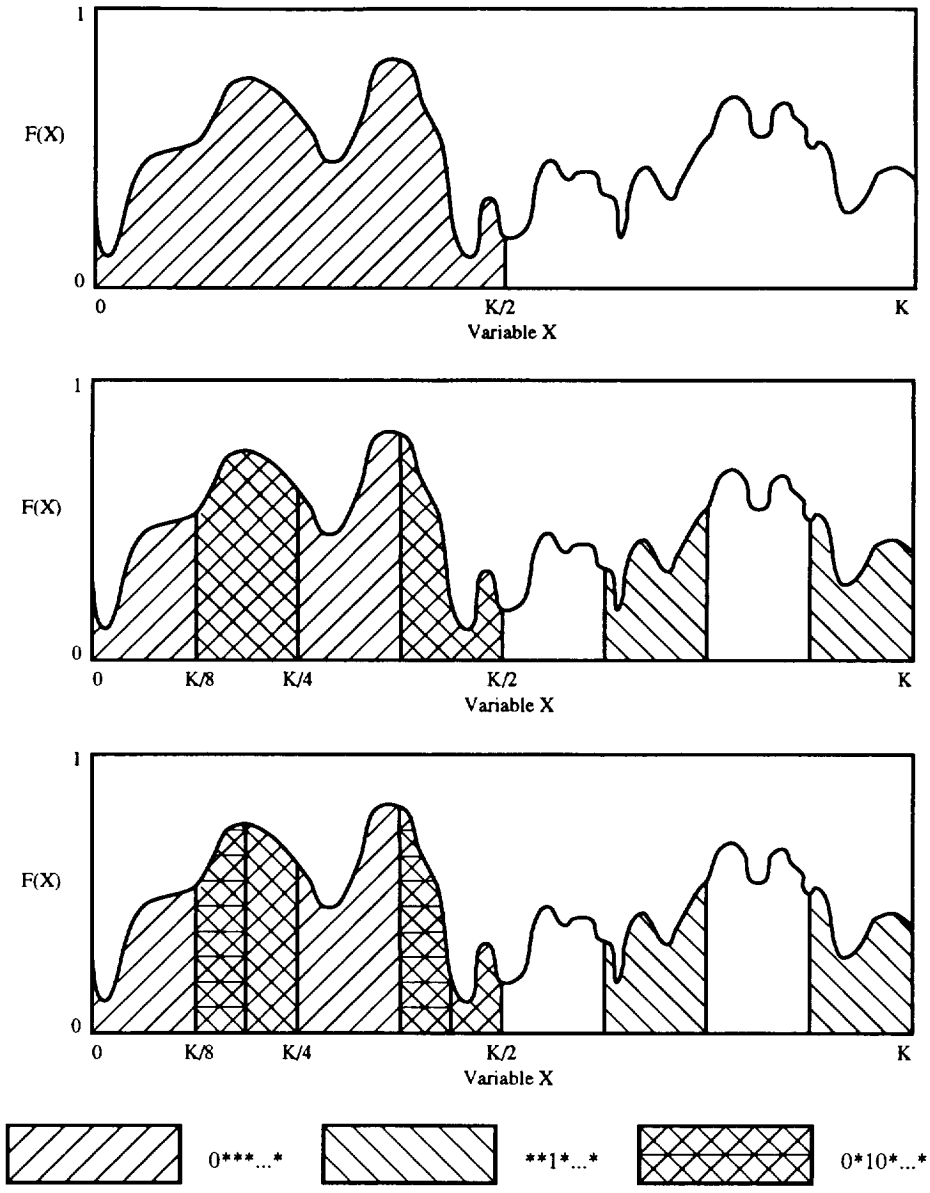



Figure 3 - A function and various partitions of hyperspace. Fitness is scaled to a 0 to 1 range in this diagram.

## SELECTION IN A SAMPLE POPULATION

This example population contains only 21 (partially specified) strings. The table includes information on the fitness of each string and the number of copies to be placed in the intermediate population. This table enumerates the 27 hyperplanes ($3^3$) that can be defined over the first three bits of the strings in the population and *explicitly* calculates the fitness associated with the corresponding hyperplane partition. The true fitness of the hyperplane partition corresponds to the average fitness of all strings that lie in that hyperplane partition.

| Strings and Fitness Values | | | | |
|---|---|---|---|---|
| String-ID | String | Fitness | Random | copies |
| 1 | 001#...# | 2.0 | - | 2 |
| 2 | 101#...# | 1.9 | 0.93 | 2 |
| 3 | 111#...# | 1.8 | 0.65 | 2 |
| 4 | 010#...# | 1.7 | 0.02 | 1 |
| 5 | 111#...# | 1.6 | 0.51 | 2 |
| 6 | 101#...# | 1.5 | 0.20 | 1 |
| 7 | 011#...# | 1.4 | 0.93 | 2 |
| 8 | 001#...# | 1.3 | 0.20 | 1 |
| 9 | 000#...# | 1.2 | 0.37 | 1 |
| 10 | 100#...# | 1.1 | 0.79 | 1 |
| 11 | 010#...# | 1.0 | - | 1 |
| 12 | 011#...# | 0.9 | 0.28 | 1 |
| 13 | 000#...# | 0.8 | 0.13 | 0 |
| 14 | 110#...# | 0.7 | 0.70 | 1 |
| 15 | 110#...# | 0.6 | 0.80 | 1 |
| 16 | 100#...# | 0.5 | 0.51 | 1 |
| 17 | 011#...# | 0.4 | 0.76 | 1 |
| 18 | 000#...# | 0.3 | 0.45 | 0 |
| 19 | 001#...# | 0.2 | 0.61 | 0 |
| 20 | 100#...# | 0.1 | 0.07 | 0 |
| 21 | 010#...# | 0 0 | - | 0 |

Table 1 - A population with fitness assigned to strings according to rank.

## PROCESSING OF HYPERPLANES

The *expected* number of strings sampling a hyperplane partition after selection can be calculated by multiplying the number of hyperplane samples in the current population before selection by the average fitness of the strings in the population that fall in that partition. The *observed* number of copies actually allocated by selection is also given. In most cases, the match between expected and observed sampling rate is fairly good: the error is a result of sampling error due to the small population size.

| Schemata and Fitness Values | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Schema | Mean | Count | Expect | Obs | | Schema | Mean | Count | Expect | Obs |
| 101*...* | 1.70 | 2 | 3.4 | 3 | | *0**...* | 0.991 | 11 | 10.9 | 9 |
| 111*...* | 1.70 | 2 | 3.4 | 4 | | 00**...* | 0.967 | 6 | 5.8 | 4 |
| 1*1*...* | 1.70 | 4 | 6.8 | 7 | | 0***...* | 0.933 | 12 | 11.2 | 10 |
| *01*...* | 1.38 | 5 | 6.9 | 6 | | 011*...* | 0.900 | 3 | 2.7 | 4 |
| **1*...* | 1.30 | 10 | 13.0 | 14 | | 010*...* | 0.900 | 3 | 2.7 | 2 |
| *11*...* | 1.22 | 5 | 6.1 | 8 | | 01**...* | 0.900 | 6 | 5.4 | 6 |
| 11**...* | 1.175 | 4 | 4.7 | 6 | | 0*0*...* | 0.833 | 6 | 5.0 | 3 |
| 001*...* | 1.136 | 3 | 3.5 | 3 | | *10*...* | 0.800 | 5 | 4.0 | 4 |
| 1***...* | 1.089 | 9 | 9.8 | 11 | | 000*...* | 0.767 | 3 | 2.3 | 1 |
| 0*1*...* | 1.033 | 6 | 6.2 | 7 | | **0*...* | 0.727 | 11 | 8.0 | 7 |
| 10**...* | 1.020 | 5 | 5.1 | 5 | | *00*...* | 0.667 | 6 | 4.0 | 3 |
| *1**...* | 1.010 | 10 | 10.1 | 12 | | 110*...* | 0.650 | 2 | 1.3 | 2 |
| ****...* | 1.000 | 21 | 21.0 | 21 | | 1*0*...* | 0.600 | 5 | 3.0 | 4 |
| | | | | | | 100*...* | 0.566 | 3 | 1.70 | 2 |

Table 2 - The average fitnesses (Mean) associated with the samples from the 27 hyperplanes defined over the first three bit positions are explicitly calculated. The Expected representation (Expect) and Observed representation (Obs) are shown. Count refers to the number of strings in hyperplane H before selection.

In the Vose and Liepins model, the vector $s^t \in \Re$ represents the $t$ th generation of the genetic algorithm and the $i$ th component of $s^t$ is the probability that the string $i$ is selected for the gene pool. The standard form of the executable equations corresponds to the following portion of the Liepins and Vose model (T denotes transpose):

$$s^T M\, s$$

A permutation function, $\sigma$, is defined as follows:

$$\sigma_j < s_0, \ldots, s_{V-1} >^T = < s_{j \oplus 0}, \ldots, s_{j \oplus (V-1)} >^T$$

where the vectors are treated as columns and $V = 2^L$, the size of the search space.

## The Vose and Liepins Model

The $i$ th component of vector $s^t$ is the probability that the string $i$ is selected for the gene pool.

$$s_i^t = P\left(S_i, t\right) f\left(S_i\right) / \bar{f}$$

The function $r_{i,j}(k)$ is used to construct a mixing matrix $M$ where the $i,j$ th entry $m_{i,j} = r_{i,j}(0)$. This matrix gives the probabilities that crossing strings $i$ and $j$ will produce $S_0$.

The proportional representation for string 0 at time t+1 is given by:

$$s^T M\, s$$

where $T$ denotes transpose.

If recombination is a combination of crossover and mutation then

$$r_{i,j}\left(k \oplus q\right) = r_{i \oplus k,\, j \oplus k}(q)$$

A general operator $\mathcal{M}$ can be defined over $s$ which remaps $s^T M\, s$ to cover all strings in the space.

$$\mathcal{M}(s) = < (\sigma_0\, s)^T M\, \sigma_0\, s, \ldots, (\sigma_{V-1}\, s)^T M\, \sigma_{V-1}\, s >^T$$

Fitness information is explicitly introduced to transform the population at the beginning of iteration $t + 1$ to the next intermediate population. A fitness matrix $F$ is defined such that fitness information is stored along the diagonal; the $i,i\ th$ element is given by $f(i)$ where $f$ is the evaluation function.

The transformation from the vector $p^{t+1}$ to the next intermediate population represented by $s^{t+1}$ is given as follows:

$$s^{t+1} = \frac{F\, p^{t+1}}{1^T F p^{t+1}}$$

## The Vose and Liepins Model

A general operator $\mathcal{M}$ can now be defined over $s$ which remaps $s^T M\, s$ to cover all strings in the search space.

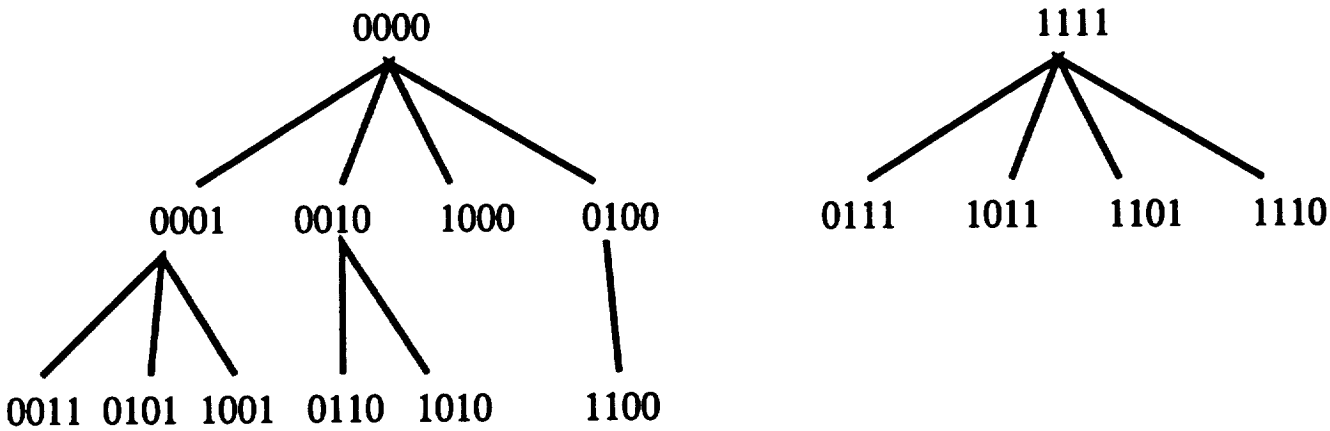
$$\mathcal{M}(s) = < (\sigma_0\, s)^T M\, \sigma_0\, s, \ldots, (\sigma_{V-1}\, s)^T M\, \sigma_{V-1}\, s >^T$$

A fitness matrix $F$ is defined such that fitness information is stored along the diagonal; following Vose and Wright, the $(i,i)^{th}$ element is given by $f(i)$ where $f$ is the fitness function.

$$s^{t+1} = \frac{F\, p^{t+1}}{1^T F p^{t+1}}$$

## BASINS OF ATTRACTION

This slide illustrates the basins of attraction for a four-bit space under steepest ascent local search. Note that equivalence classes of functions are created when one considers the set of functions that have the same basins of attraction. These equivalence classes carry over into hybrid genetic algorithms that combine genetic search with local search, also referred to as Lamarckian search.



## UPDATES TO REPRESENTATION UNDER LOCAL SEARCH

Local search can be added to the existing models of genetic algorithms. This slide shows how one step of steepest ascent changes the distribution of samples in the population.

How Population Distributions Change Under Lamarckian Search

```
P'(0000,t) = P(0000,t) + P(0001,t) + P(0010,t) + P(0100,t) + P(1000,t)
P'(0001,t) = P(0011,t) + P(0101,t) + P(1001,t)
P'(0010,t) = P(0110,t) + P(1010,t)
P'(0100,t) = P(1100,t)
P'(1111,t) = P(1111,t) + P(1110,t) + P(1101,t) + P(1011,t) + P(0111,t)
P'(0011,t) = 0
P'(0101,t) = 0
P'(0110,t) = 0
P'(0111,t) = 0
P'(1000,t) = 0
P'(1001,t) = 0
P'(1010,t) = 0
P'(1011,t) = 0
P'(1100,t) = 0
P'(1101,t) = 0
P'(1110,t) = 0
```

Function 1:

| | | | |
|---|---|---|---|
| e(0000) = 28 | e(0100) = 20 | e(1000) = 12 | e(1100) = 4 |
| e(0001) = 26 | e(0101) = 18 | e(1001) = 10 | e(1101) = 2 |
| e(0010) = 24 | e(0110) = 16 | e(1010) = 8 | e(1110) = 0 |
| e(0011) = 22 | e(0111) = 14 | e(1011) = 6 | e(1111) = 30 |

The following Function is Equivalent to Function 1 Under Hybrid Search.

| | | | |
|---|---|---|---|
| e(0000) = 28 | e(0100) = 20 | e(1000) = 18 | e(1100) = 18 |
| e(0001) = 26 | e(0101) = 24 | e(1001) = 24 | e(1101) = 18 |
| e(0010) = 24 | e(0110) = 22 | e(1010) = 22 | e(1110) = 18 |
| e(0011) = 24 | e(0111) = 22 | e(1011) = 22 | e(1111) = 30 |

## A MATRIX REPRESENTATION OF LOCAL UPDATES

The same redistribution of samples seen below can be captured in matrix form and used to model multiple steps of local search.

### Lamarckian Updates to the p Vector

$$p^T \begin{bmatrix} 1\,0\,0\,0\,0 & 0 \\ 1\,0\,0\,0\,0 & 0 \\ 1\,0\,0\,0\,0 & 0 \\ 0\,1\,0\,0\,0 & 0 \\ 1\,0\,0\,0\,0 & 0 \\ 0\,1\,0\,0\,0 & 0 \\ 0\,0\,1\,0\,0 & 0 \\ 0\,0\,0\,0\,0 & \dots\ 0 \\ 1\,0\,0\,0\,0 & 0 \\ 0\,1\,0\,0\,0 & 0 \\ 0\,0\,1\,0\,0 & 0 \\ 0\,0\,0\,0\,0 & 0 \\ 0\,0\,0\,0\,1 & 0 \\ 0\,0\,0\,0\,0 & 0 \\ 0\,0\,0\,0\,0 & 0 \\ 0\,0\,0\,0\,0 & 0 \end{bmatrix}$$

For one step of local search: $p^T L$

For 2 steps of local search: $(p^T L)L = p^T L^2$

For k steps of local search: $p^T L^k$

## THE GEOMETRIC MATCHING PROBLEM

The geometric matching problem matches a model to a set of data. For example, the model may be a line drawing and the data might be line segments extracted from a photograph. Problems of this type arise in a number of areas and are specifically relevant to problems associated with computer vision. Variations of local search are currently being used to solve geometric matching problems in the context of semi-autonomous photo-interpretation and robot navigation.

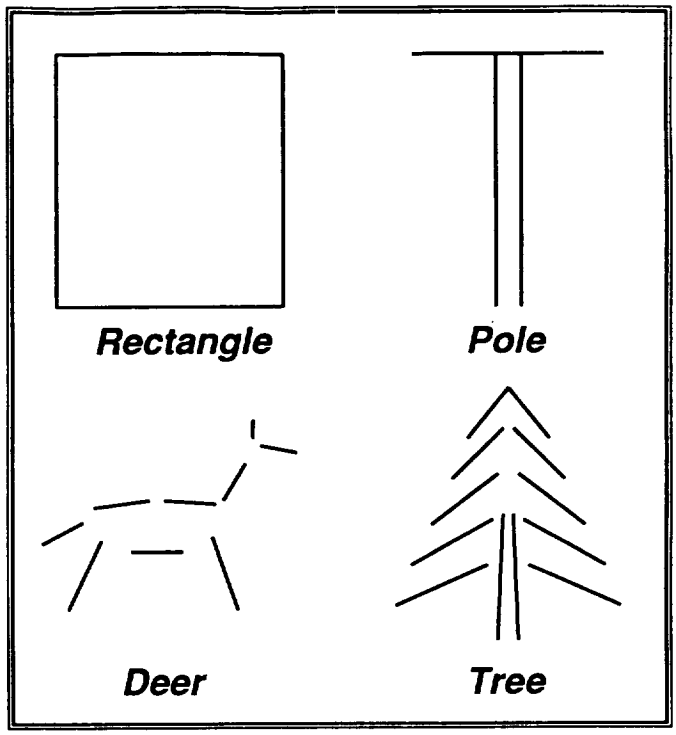



Figure 1 - Four stick figure models used in geometric matching tests.

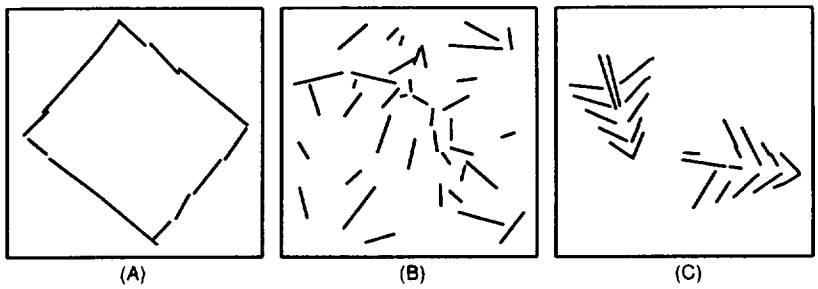



Figure 2 - Sample of synthetic test data for matching: a) rectangle model randomly placed, scaled and corrputed, b) corrupted deer model with random clutter, c) multiple corrupted instances of the tree model.

## AN EXAMPLE MATCH

An example of a best match is shown for the tree model. The line segments making up the model are labeled with letters and are shown on the left. The data line segments include three instances of the tree and are shown to the right. The model is overlaid on top of the data in the best match position. The correspondence matrix indicates which pairs of model and data segments are part of this best match. Each square in the table may be thought of as representing one bit in the bit string encoding of the match. The filled in squares correspond to 1s in the bit string. Details for this application can be found in Whitley, Beveridge, Graves and Mathias (1995), "Testing Driving Three 1995 Genetic Algorithms," Journal of Heuristics.

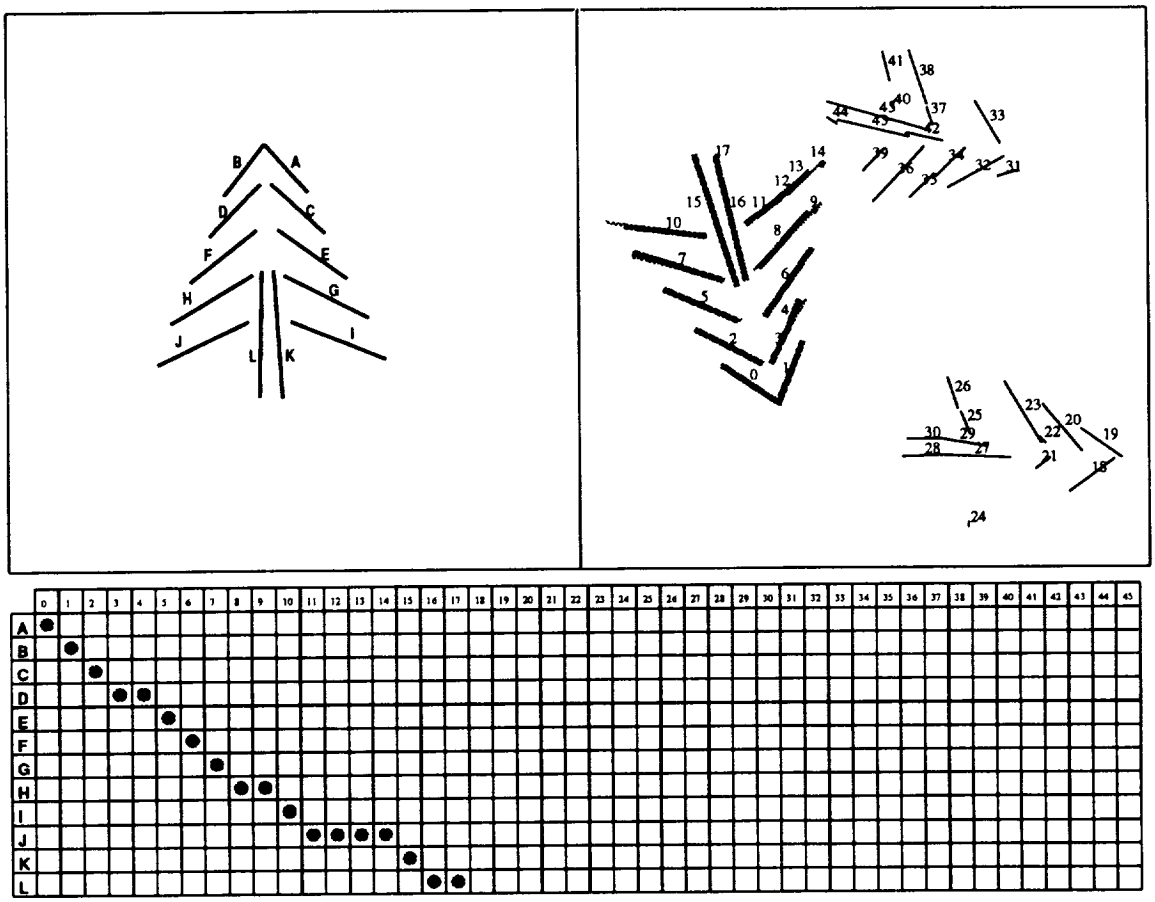
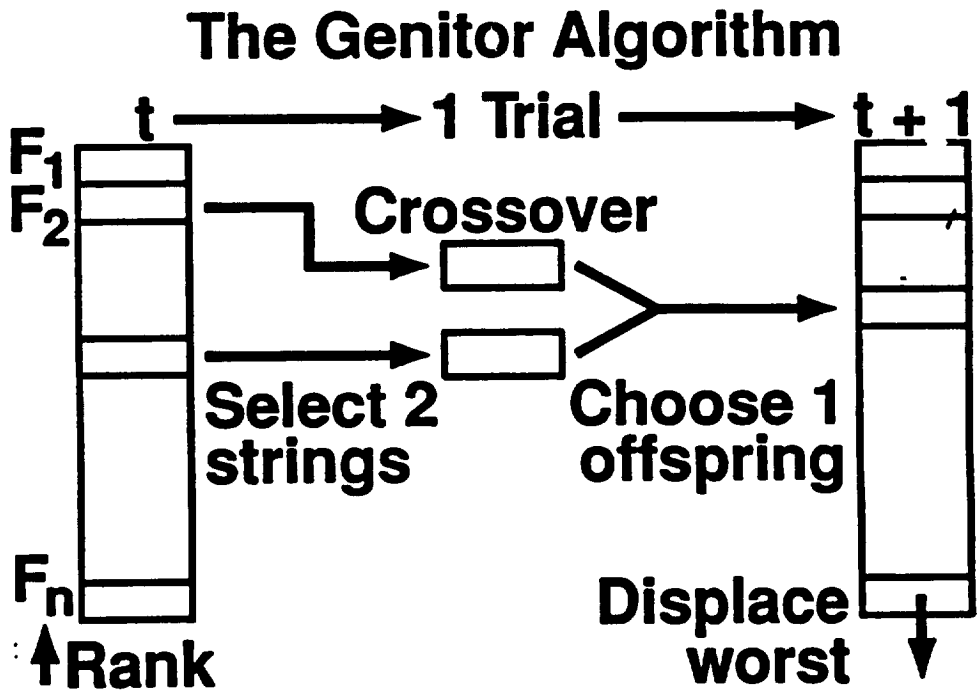



Figure 3 - Example of a best match for one of the 32 matching problems.

The Genitor Algorithm. Offspring are produced one at a time and replace the worst member of the population.

## Genitor

- Ranking used to determine fitness
- One-at-a-time recombination and replacement



## COMPARISON OF LOCAL SEARCH AND GENITOR

Neither the local search algorithms nor the genetic algorithms converge upon the globally optimal match in all cases. Therefore, in comparing algorithms for model matching, two things must be taken into consideration. First is the likelihood that the algorithm once initiated will converge on the optimal solution. The second is how much work must be performed in order to reach convergence. LS Inc uses a fast incremental evaluation while LS Full uses a full evaluation.

| | | Trials Required: $t_{95}$ | | | Estimated Run Time | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Model | Params | LS Inc. | LS Full | Genitor+ | LS Inc. | | LS Full | | Genitor+ | |
| Pole | 12C | 6 | 5 | 1 | 0.2 | ⋆ | 0.2 | ⋆ | 0.3 | |
| Pole | 24M | 8 | 7 | 2 | 0.8 | o | 0.7 | ⋆ | 1.4 | |
| Rectangle | 28C | 6 | 6 | 3 | 0.6 | ⋆ | 0.6 | ⋆ | 3.6 | |
| Pole | 42C | 26 | 20 | 5 | 2.6 | o | 2.0 | ⋆ | 4.0 | |
| Pole | 42M | 12 | 10 | 7 | 1.2 | o | 1.0 | ⋆ | 9.1 | |
| Rectangle | 52M | 5 | 5 | 4 | 1.0 | ⋆ | 1.5 | o | 7.2 | |
| Rectangle | 68C | 11 | 9 | 4 | 1.1 | ⋆ | 2.7 | o | 8.4 | |
| Pole | 72C | 35 | 39 | 20 | 3.5 | ⋆ | 7.8 | o | 20.0 | |
| Deer | 81C | 13 | 11 | 2 | 3.9 | ⋆ | 8.8 | o | 12.2 | |
| Pole | 81M | 37 | 29 | 11 | 3.7 | ⋆ | 8.7 | o | 20.9 | |
| Pole | 96M | 45 | 24 | 29 | 9.0 | ⋆ | 9.6 | o | 52.2 | |
| Deer | 99M | 19 | 15 | 3 | 7.6 | ⋆ | 18.0 | o | 25.5 | |
| Pole | 102C | 96 | 72 | 13 | 9.6 | ⋆ | 21.6 | | 20.8 | o |
| Rectangle | 108C | 47 | 17 | 8 | 9.4 | ⋆ | 10.2 | o | 41.6 | |
| Rectangle | 108M | 12 | 13 | 18 | 3.6 | ⋆ | 10.4 | o | 138.6 | |
| Rectangle | 124M | 14 | 15 | 49 | 5.6 | ⋆ | 13.5 | o | 240.1 | |
| Rectangle | 148C | 42 | 26 | 8 | 12.6 | ⋆ | 23.4 | o | 67.2 | |
| Tree | 156C | 42 | 32 | 9 | 33.6 | ⋆ | 118.4 | o | 361.5 | |
| Rectangle | 168M | 20 | 24 | 24 | 10.0 | ⋆ | 72.0 | o | 206.4 | |
| Deer | 171C | 29 | 34 | 4 | 17.4 | ⋆ | 85.0 | o | 137.6 | |
| Deer | 180M | 31 | 33 | 6 | 21.7 | ⋆ | 99.0 | o | 152.4 | |
| Tree | 216M | 51 | 52 | 8 | 66.3 | ⋆ | 291.2 | o | 331.5 | |
| Deer | 261C | 43 | 67 | 6 | 43.0 | ⋆ | 341.7 | o | 414.0 | |
| Deer | 261M | 59 | 46 | 6 | 59.0 | ⋆ | 239.2 | o | 372.0 | |
| Tree | 276C | 51 | 52 | 5 | 76.5 | ⋆ | 473.2 | o | 617.5 | |
| Deer | 342M | 106 | 67 | 7 | 159.0 | ⋆ | 576.2 | o | 984.9 | |
| Deer | 351C | 64 | 66 | 6 | 96.0 | ⋆ | 607.2 | | 517.2 | o |
| Tree | 396C | 52 | 70 | 3 | 114.4 | ⋆ | 1,113.0 | | 829.2 | o |
| Tree | 432M | 135 | 99 | 7 | 310.5 | ⋆ | 1,871.1 | o | 2,405.2 | |
| Tree | 516C | 99 | 96 | 5 | 297.0 | ⋆ | 2,438.4 | | 2,156.5 | o |
| Tree | 552M | 149 | 106 | 75 | 461.9 | ⋆ | 3,307.2 | o | 35,317.5 | |
| Tree | 780M | 299 | 175 | - | 1,554.8 | ⋆ | 10,202.5 | o | - | |

Table 8 - Comparison of steepest ascent local search using partial incremental evaluation (LS-Inc), steepest ascent local search using full evaluation (LS-Full), and Genitor+algorithm on 32 test problems.

## COMPARISON OF A HYBRID ALGORITHM

A "Hybrid Genetic Algorithm" uses local search as an operator to improve the strings in the population. The comparison in this slide suggests there is little difference in the hybrid genetic algorithm compared to local search using incremental evaluation. Aside from concerns about the execution speed of the implementations of the different algorithms, there is another factor which very much influences the results shown here: What criterion should be used to terminate genetic search when evaluation of the global optimum is unknown? This issue typically does not arise for artifically constructed test functions and this issue has not received a great deal of attention in the genetic algorithm literature.

| | | Success Prob.: $P_s$ | | Trials: $t_{95}$ | | Estimated Run Time | | |
|---|---|---|---|---|---|---|---|---|
| Model | Params | LS Inc. | Hybrid | LS Inc. | Hybrid | LS Inc. | Hybrid | % Diff. |
| Pole | 12C | 0.42 | 0.94 | 6 | 2 | 0.2 | 0.2 | 83% |
| Pole | 24M | 0.32 | 0.98 | 8 | 1 | 0.8 | 0.3 | 38% |
| Rectangle | 28C | 0.44 | 1.00 | 6 | 1 | 0.6 | 0.5 | 83% |
| Pole | 42C | 0.11 | 0.52 | 26 | 5 | 2.6 | 2.0 | 77% |
| Pole | 42M | 0.23 | 0.80 | 12 | 2 | 1.2 | 1.2 | 100% |
| Rectangle | 52M | 0.47 | 0.98 | 5 | 1 | 1.0 | 1.4 | 140% |
| Rectangle | 68C | 0.24 | 1.00 | 11 | 1 | 1.1 | 1.1 | 100% |
| Pole | 72C | 0.08 | 0.42 | 35 | 6 | 3.5 | 4.8 | 137% |
| Deer | 81C | 0.21 | 0.66 | 13 | 3 | 3.9 | 7.2 | 185% |
| Pole | 81M | 0.08 | 0.62 | 37 | 4 | 3.7 | 6.4 | 173% |
| Pole | 96M | 0.07 | 0.40 | 45 | 6 | 9.0 | 9.6 | 107% |
| Deer | 99M | 0.15 | 0.84 | 19 | 2 | 7.6 | 6.0 | 79% |
| Pole | 102C | 0.03 | 0.40 | 96 | 6 | 9.6 | 8.4 | 88% |
| Rectangle | 108C | 0.06 | 0.64 | 47 | 3 | 9.4 | 6.0 | 64% |
| Rectangle | 108M | 0.23 | 0.94 | 12 | 2 | 3.6 | 8.0 | 222% |
| Rectangle | 124M | 0.19 | 0.92 | 14 | 2 | 5.6 | 7.6 | 136% |
| Rectangle | 148C | 0.07 | 0.60 | 42 | 4 | 12.6 | 14.0 | 111% |
| Tree | 156C | 0.07 | 0.58 | 42 | 4 | 33.6 | 36.8 | 110% |
| Rectangle | 168M | 0.14 | 0.76 | 20 | 3 | 10.0 | 18.3 | 183% |
| Deer | 171C | 0.10 | 0.60 | 29 | 4 | 17.4 | 29.2 | 168% |
| Deer | 180M | 0.10 | 0.90 | 31 | 2 | 21.7 | 16.8 | 77% |
| Tree | 216M | 0.06 | 0.70 | 51 | 3 | 66.3 | 47.7 | 72% |
| Deer | 261C | 0.07 | 0.52 | 43 | 5 | 43.0 | 80.5 | 187% |
| Deer | 261M | 0.05 | 0.82 | 59 | 2 | 59.0 | 35.4 | 60% |
| Tree | 276C | 0.06 | 0.64 | 51 | 3 | 76.5 | 78.3 | 102% |
| Deer | 342M | 0.03 | 0.64 | 106 | 3 | 159.0 | 105.6 | 66% |
| Deer | 351C | 0.05 | 0.54 | 64 | 4 | 96.0 | 145.2 | 151% |
| Tree | 396C | 0.06 | 0.80 | 52 | 2 | 114.4 | 125.2 | 109% |
| Tree | 432M | 0.02 | 0.64 | 135 | 3 | 310.5 | 235.5 | 76% |
| Tree | 516C | 0.03 | 0.74 | 99 | 3 | 297.0 | 390.6 | 132% |
| Tree | 552M | 0.02 | 0.74 | 149 | 3 | 461.9 | 482.4 | 104% |
| Tree | 780M | 0.01 | 0.46 | 299 | 5 | 1,554.8 | 1,735.0 | 112% |

Table 9 - Comparison of steepest ascent local search using partial evaluation (LS-Inc) and the hybrid genetic algorithm. % Diff refers to the amount of time required by the hybrid to find a solution with 95% reliability compared to local search.

## EVALUATION TIME FOR THE HYBRID GA

The data presented in this slide indicates that the hybrid genetic algorithm is finding the solution to the problems long before the stopping criterion is satisfied. The data in Table 10 is broken down into the number of evaluations used by the local search algorithm (which can exploit incremental evaluation) as well as the full-evaluation calls used by the genetic algorithm after mating. The number of evaluations required to find the best solution is less than half the number of evaluations required to satisfy the stopping criterion. Cutting the number of evaluations in half can be achieved easily by picking a better stopping criteria. This would then represent a new state-of-the-art in general geometric matching.

| | | | Best Match Found | | Convergence | | |
|---|---|---|---|---|---|---|---|
| Model | Params | $P_s$ | Inc-Evals | Full-Evals | Inc-Evals | Full-Evals | Secs |
| Pole | 12C | 94 | 161.4 | 22.1 | 727.6 | 129.2 | 0.2 |
| Pole | 24M | 96 | 452.0 | 16.3 | 2540.8 | 138.0 | 0.5 |
| Rectangle | 28C | 100 | 438.8 | 12.2 | 3693.9 | 164.7 | 0.7 |
| Pole | 42C | 54 | 1311.2 | 47.6 | 3264.2 | 141.8 | 0.9 |
| Pole | 42M | 90 | 1284.0 | 30.2 | 4748.0 | 148.8 | 0.9 |
| Rectangle | 52M | 100 | 1462.4 | 13.1 | 11597.6 | 178.9 | 1.9 |
| Rectangle | 68C | 100 | 1920.9 | 22.1 | 10663.4 | 184.7 | 1.8 |
| Pole | 72C | 42 | 2324.5 | 49.8 | 5949.4 | 150.8 | 1.2 |
| Deer | 81C | 84 | 6175.3 | 40.6 | 23095.7 | 252.9 | 3.6 |
| Pole | 81M | 72 | 4288.2 | 45.8 | 12057.3 | 162.4 | 2.4 |
| Pole | 96M | 56 | 3999.0 | 38.6 | 12707.8 | 163.3 | 2.5 |
| Deer | 99M | 92 | 9740.3 | 60.3 | 30829.1 | 280.7 | 4.7 |
| Pole | 102C | 48 | 4161.0 | 58.3 | 10026.1 | 163.7 | 2.1 |
| Rectangle | 108C | 80 | 4888.9 | 37.9 | 18074.7 | 197.9 | 3.4 |
| Rectangle | 108M | 96 | 6251.5 | 28.7 | 28152.4 | 205.8 | 5.0 |
| Rectangle | 124M | 94 | 6286.8 | 33.8 | 29511.1 | 233.6 | 5.7 |
| Rectangle | 148C | 62 | 7684.1 | 74.2 | 19792.1 | 227.0 | 4.3 |
| Tree | 156C | 68 | 37024.8 | 154.9 | 83272.4 | 505.6 | 13.4 |
| Rectangle | 168M | 90 | 8978.0 | 48.0 | 41066.9 | 291.4 | 8.4 |
| Deer | 171C | 78 | 24980.9 | 151.2 | 62196.5 | 520.5 | 10.6 |
| Deer | 180M | 88 | 21017.0 | 116.8 | 73928.3 | 602.3 | 12.0 |
| Tree | 216M | 74 | 52856.6 | 271.0 | 129904.2 | 897.4 | 21.1 |
| Deer | 261C | 62 | 50067.9 | 285.2 | 113533.1 | 796.8 | 20.7 |
| Deer | 261M | 84 | 37051.1 | 206.2 | 122304.8 | 925.7 | 22.7 |
| Tree | 276C | 72 | 89913.4 | 370.7 | 202536.3 | 1082.0 | 33.8 |
| Deer | 342M | 66 | 65857.8 | 274.4 | 184221.0 | 966.9 | 35.3 |
| Deer | 351C | 64 | 85626.9 | 371.5 | 179823.7 | 941.0 | 35.2 |
| Tree | 396C | 70 | 148512.5 | 504.8 | 332537.4 | 1392.8 | 62.9 |
| Tree | 432M | 76 | 175521.8 | 693.4 | 387632.3 | 1864.1 | 82.9 |
| Tree | 516C | 56 | 245209.8 | 539.8 | 571884.2 | 1590.3 | 113.7 |
| Tree | 552M | 66 | 263373.7 | 613.2 | 643095.3 | 1937.2 | 130.7 |
| Tree | 780M | 42 | 429647.0 | 701.4 | 1032966.4 | 2191.5 | 245.9 |

Table 10 - Comparison of number of evaluations required to find solutions in contrast to time required to reach the stopping criterion. This is further broken down into the incremental evaluations (Inc-Evals) used by the local search component of the hybrid algorithm and full evaluations (Full-Evals) of strings generated by recombination and mutlation.

## OTHER APPLICATIONS

- **Navy Air-Plane Simulator Scheduling**
  Developed by BBN using CSU algorithms

- **Poor Prototype Warehouse Scheduler**

  A fast evaluation function is used that approximates warehouse operations which is orders of magnitude faster than a corresponding detailed warehouse simulator.

  Local methods yield poor results when the evaluation function is noisy or approximate, while genetic algorithms yield very good solutions.

- **Artificial Genetically Grown Neural Networks**

  Neural networks are grown using cell development processes for control applications. The grammar controlling the growth process is optimized rather than the neural network itself. The resulting networks have outperformed temporal difference methods on several applications.

- **Seismic Data Interpretation**