Genetic Algorithms Applied to the Scheduling of The Hubble Space Telescope

Jeffrey L. Sponsler
*Space Telescope Science Institute**
3700 San Martin, Baltimore, MD 21218 USA

Abstract

A prototype system employing a genetic algorithm (GA) has been developed to support the scheduling of the Hubble Space Telescope. A non-standard knowledge structure is used and appropriate genetic operators have been created. Several different crossover styles (random point selection, evolving points, and smart point selection) are tested and the best GA is compared with a neural network (NN) based optimizer. The smart crossover operator produces the best results and the GA system is able to evolve complete schedules using it. The GA is not as time-efficient as the NN system and the NN solutions tend to be better. Work is proposed to create a classifier system which can draw more effectively on the knowledge that is available in the scheduling domain.

1. Introduction

Genetic algorithms (Holland, 1975) are modeled from organic evolutionary systems and have been applied to search problems. Schaffer *et al* (1988) examined the evolution of crossover points (a technique used to minimize disruption of high performance schemata in chromosomes). Greffenstette (1988) has examined genetic algorithms with respect to rule discovery. De Jong (1980) applied GA technology to the problem of adaptive system design.

Genetic algorithm technology has been applied to real world problems such as scheduling. Hilliard *et al* (1988) have designed a competition-based system to discover scheduling heuristics. Baffes *et al* (1988) used greedy double-crossover techniques to generate optimal solutions to the space station mobile transporter scheduling problem.

This report describes on-going research wherein genetic algorithm technology has been applied to the problem of searching for feasible schedules for the Hubble Space Telescope and is organized as follows: First, a description of the Space Telescope and the constraint propagation system is given. Second, an informal analysis of the problem complexity is provided. Next, the genetic algorithm, crossover operators, and the results of experimentation that has been done are reported. Last, the results are discussed.

* Operated by the Association of Universities for Research in Astronomy for the National Aeronautics and Space Administration

2. The Hubble Space Telescope and SPIKE

The Hubble Space Telescope (HST) is an astronomical observatory that is to be placed into orbit in the near future by a space shuttle mission. The Space Telescope Science Institute is responsible for software ground support for HST. This involves, among other tasks, the processing of proposals (specifications for scientific experiments) submitted by astronomers in the international community. A proposal generally contains requests for activities (target acquisitions, exposures, and calibrations) and may include constraints upon these activities (e.g., "A before B"). The AI group at STScI has developed a system called **Spike**, which is a long-term scheduling utility. This system is described briefly below; for a more complete discussion of the Spike system, see Miller *et al* (1988).

2.1. Description of the Constraint-Based Scheduler

The Spike system supports and processes the following data objects: **targets** (stars, etc.), **activities** (exposures, etc.), **absolute constraints** (e.g., moon exclusion), and **relative constraints** (e.g., "A before B"). Each activity has an associated **suitability** that is a function of time and which provides knowledge about when it is legal to schedule an activity. A suitability is represented internally by a **piecewise constant function** (PCF) and is a list of time/value pairs (e.g., (minus-infinity 0 100 0.5 150 1 200 0.5 300 0)). Each absolute constraint is derived from appropriate astronomical models (e.g., the moon-exclusion constraint suitability is a function of target position and has value of 1 when the moon does not block the target and value of 0 otherwise). The suitability of a relative constraint is determined by looking at the activities that are linked via the constraint. For example, the suitability of the constraint "A before B" is calculated by looking at the legal times for A and doing the arithmetic to determine what times are legal for B such that B will follow A.

A group of activities linked via relative constraints is called a **dependency cluster**. Constraints may indirectly interact via common activity connections and so a **constraint propagation** technique is used. For example, if the explicit constraints "A before B" and "B before C" are specified, then "A before C" is an implicit constraint that is inferred via this propagation.

Spike is responsible for producing coarse year-long schedules consisting of time **segments** that are roughly week-long. Generating a finer schedule far in advance of execution is not reasonable (as the difference between present and future time increases the accuracy of HST orbit models decreases). The assignment of an activity to a time segment is called a **commitment**. This action causes the PCF of the activity to be zeroed over all time points outside of the selected time segment.

2.2. Complexity of Scheduling Problem

The scheduling of HST is an NP-complete problem. The generation and testing of all possible schedules is therefore practically impossible in cases where the numbers of things to be scheduled is large. In order to examine the complexity, let A be the number of activities to be scheduled and let S be the number of time segments in a schedule. The number of ways that one can assign A activities to S time segments is S^A.

The order in which the commitments in a possible schedule are made is important in judging the merits of a schedule. First, not all possible schedules can be instantiated; doing some subset of commitments may cause some other subset of commitments to be illegal. Second, in Spike the summed suitability of a schedule is used to rate the overall goodness; different orderings of the same set of activity/segment commitments will yield different results. Consider Figure 1 which illustrates how the ordering of commitments is important. On the basis of this, the complexity of the HST scheduling problem is $O(A!\ S^A)$.

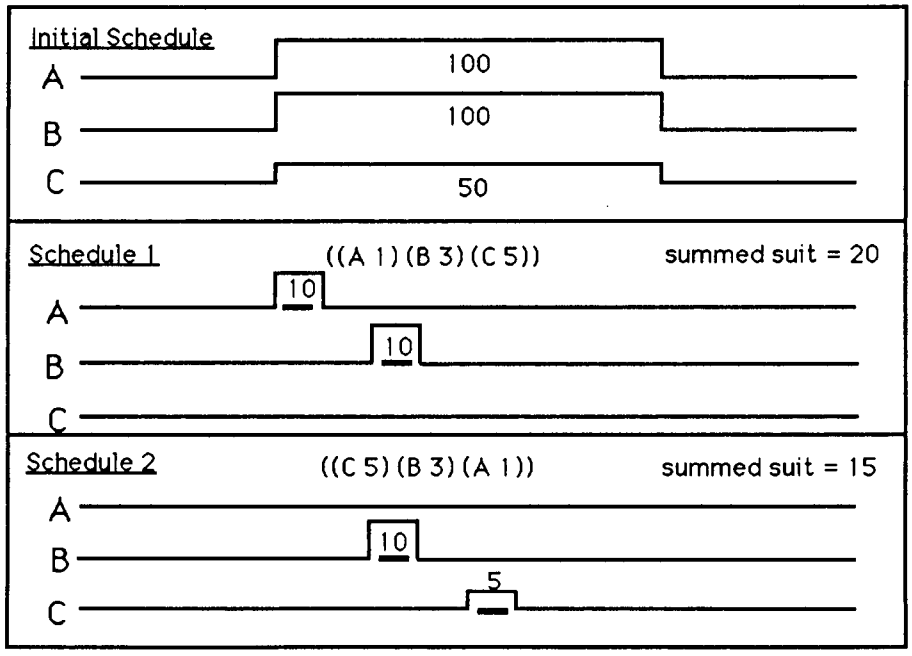

Figure 1. The ordering of commitments may yield different schedule suitabilities. In this contrived example, the relative constraint *within-2-time-units(A, B, C)* is implicit; this means that it is only legal to commit the activities so constrained such that no two are more than 2 time units apart. Committing activities A and B first in Schedule 1 disallows the (C 5) commitment (due to constraint violation) and produces suitability of 20; committing C and B first in Schedule 2 disallows the (A 1) commitment and produces suitability of 15. (A commitment is represented here as a small dark bar on a time line.)

2.3. Meta-Scheduling Techniques

The core Spike system has been implemented in a Common Lisp/Flavors/Common Windows environment. A user can manually create schedules or several procedural algorithms can be applied to automatically create schedules. The goal of the planning group is to use a meta-scheduling system to search for "optimal" solutions. A rule-based system and a neural network (NN) system (Johnston, 1989) have produced very promising results.

3. A Genetic Algorithm Applied to HST Scheduling

Genetic Algorithm technology has been applied to the HST scheduling problem in the form of a prototype system coded in Common Lisp that interacts with Spike and the neural network representation of commitments and constraints.

3.1. Representation of Knowledge Structures

The classic bit-string representation was not used in the GA prototype due to these factors: Crossover must yield complete schedules (one in which all activities are included) and the order of the commitments is important. Instead, the following representation was used. Each knowledge structure (**chromosome**) is a list consisting of sublists. Each sublist contains an activity id and a time segment number. An example chromosome illustrates: ((a 5) (b 7) (c 10) (d 10)). The ordering from left to right specifies the ordering of the commitment attempts. The interpretation of the example chromosomes is this: First attempt to commit activity a to time segment 5, then attempt to commit b to 7, and so on. All commitment pairs are attempted. The success of a given commitment attempt may have an affect on successive attempts. This is due to changes in suitabilities resulting from constraint propagation (i.e., commiting **a** to 5 may remove 10 as a valid time for **c**).

3.2. Crossover Techniques

In order to support exploration (via recombination) through the search space, two types of crossover are used. They have been called *horizontal* and *vertical.*

Vertical crossover is a binary operator and works in the following way. Let C_i and C_k be chromosomes. Select a site S_a in C_i where the desired cross is to occur. Then find the activity A_a at that site, and the corresponding site S_b in C_k where A_a resides. Swap the segments T_a at site S_a and T_b at site S_b. For example, if C_i is ((a 1) (b 2)(c 3)) and C_k is ((b 8) (a 7) (c 6)), swapping at site 2 in C_i will result in two new chromosomes: ((a 1) (b 8) (c 3)) and ((b 2) (a 7) (c 6)). This technique bears some resemblance to Goldberg's *partially matched crossover* (Goldberg, 1988). Vertical crossover can be applied iteratively over a range from one site to another.

Horizontal crossover is a unary operator and works as follows. Select two sites on the chromosome to be manipulated. Swap the two activity/segment pairs at those sites. For example, if the chromosome is ((a 1) (b 2) (c 3) (d 4)) and the sites are 2 and 4, the result will be ((a 1) (d 4) (c 3) (b 2)).

3.3. The Mutation Operator

The mutation operator works as follows: For a given chromosome iterate over each activity/segment pair. Flip a biased coin and if *true* results randomly select a new segment for the activity from the Table of Legal Segments.

3.4. The Genetic Algorithm

The genetic algorithm developed is not a general purpose parameter optimizer due to the unusual chromosome form. It does fit, however, into the general purpose design of the Spike scheduling system. Certain modifications to the algorithm form the basis for the experimentation reported here (details are found in later sections). Chromosomes are haploid. The pseudo-code algorithm and functional descriptions follow:

```
initialize-spike-system
create-table-of-legal-segments
setup-initial-population
process-chromosomes
loop-while (no-solution OR generations < max-gen)
   reproduce-chromosomes
   process-chromosomes
end-of-loop
```

To execute **initialize-spike-system,** proposals are selected from a pool and Spike data structures are instantiated as usual. In order to execute **create-table-of-legal-segments,** for each activity, a list of the legal segments where (at least initially) the activity could be committed is collected and stored in the Table with the activity.

The **setup-initial-generation** function creates a set of chromosomes of a specified size. To generate one chromosome, a complete set of randomly ordered activities is created. For each activity a segment number is selected randomly from the Table of Legal Segments.

The function **process-chromosomes** is responsible for interpreting (determining the fitness of) each chromosome (C_i) in a generation as well as calculating the average fitness of the generation and the relative fitness of each chromosome. The formula used to calculate a chromosome's fitness follows:

$$\text{fitness}(C_i) = \text{expt}(10 * \text{commitments/possible-commitments}) * \text{summed-suitability}(C_i)$$

The fitness function is heavily biased such that schedules with high commitment ratios are favored (the adjusted ratio is exponentiated with base of *e*). The motive for this was to highly reward the system for completed schedules. The function factors in the suitability of the schedule; this acts in a more subtle manner to differentiate schedules based on how good the commitments were.

Baker's Stochastic Universal Selection Process (Baker, 1988) is used to select offspring (surviving chromosomes) from the individuals in the generation. This technique selects offspring from a population based on the relative fitness of the individuals and is linear in complexity.

The step labeled **reproduce-chromosomes** executes crossover and mutation on each offspring. These actions are triggered with some specified probability.

3.5. Neural Network Used for Rapid Fitness Analysis

During chromosome interpretation, the Spike system is commanded to make a series of commitments. With each commitment, expensive processing is done (mostly to propagate relative constraints). In order to minimize this processing the following approach has been taken. A neural network is created that stores (as connection strengths) what changes occur to suitabilities when any given commitment is made. This network is then used to execute fitness evaluation. This approach has decreased GA processing time by at least one order of magnitude; the number of disposable *cons cells* generated is also much lower.

4. Description of Tested Hypotheses

4.1. A Smart Horizontal Crossover Technique

A modification on the basic random crossover technique has been developed because the random crossover was inefficient in achieving goal states. The *smart horizontal crossover* operator is knowledge-based and works as follows: Given two chromosomes C_i and C_k that have had fitness evaluations and have been selected as a mating pair, randomly select one chromosome (e.g., C_i). Consider each site (activity/segment pair) on C_i. If a site coded for a legal commitment (within the context of the ordered set of sites) then do nothing. If the site did not code for a legal commitment (as a result of previous commitments), execute a single vertical crossover with the appropriate site on the other chromosome (e.g., C_k). This technique will tend to preserve good regions regardless of length for one of the two chromosomes. Figure 2 illustrates.

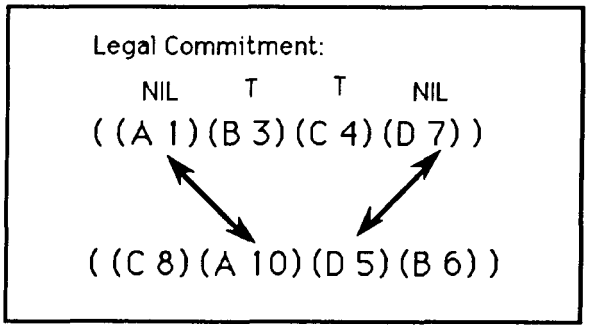

Figure 2. Smart horizontal crossover preserves high quality schemata. A T indicates a successful commitment and a NIL indicates an unsuccessful one.

The following hypotheses have been tested and the results will be presented in a later section.

H_0: A smart crossover will cause the GA to approach a solution state in roughly the same number of generations as that of a GA using a Random Crossover.

H_1: The GA employing Smart Crossover will reach a solution state in fewer generations than a GA using Random Crossover.

4.2. Evolution of Crossover Points

An algorithm that supports the evolution of crossover points has been implemented. Let C1 and C2 be the parent chromosomes. Associated with each are two crossover points. During crossover, one of the parents is randomly selected as the source of the two crossover points used for vertical crossover (which operates as usual). Let C3 and C4 be the offspring. C3 will inherit the crossover points from C1 and C4 will inherit from C2. Crossover points are subjected to mutation with frequency that is based on the specified probability of that operator. A crossover point mutation is merely the random selection of a new chromosome site number.

H_0: An evolving crossover will cause the GA to approach a solution state in roughly the same number of generations as that of a GA using a random crossover.

H_1: The GA employing the evolving crossover algorithm will reach a solution state in fewer generations than a GA using Random Crossover.

4.3. GA technology vs Neural Network

Since a neural network-based search algorithm is available, it seemed only reasonable to compare the two technologies. It was not expected that the GAs would perform as well as the NNs with respect to time. The summed suitability of solutions however was compared as a means to determine which technology produced more optimal solutions.

5. Experimental Results

Figures 3 and 4 illustrate the results from the experiments. The number of activities was 10 and each activity was constrained in several ways. From ten trials, an "average" run was selected from the Random and Smart results; the change in number of commitments over time is seen in Figure 3. Also from ten trials, the best run has been selected from Random, Evolving, and Smart Crossover trials; these results are seen in Figure 4.

In each trial, the probability of mutation was 0.03, the probability of crossover was 0.6, the population size was 30, and maximum allowed generations was 51.

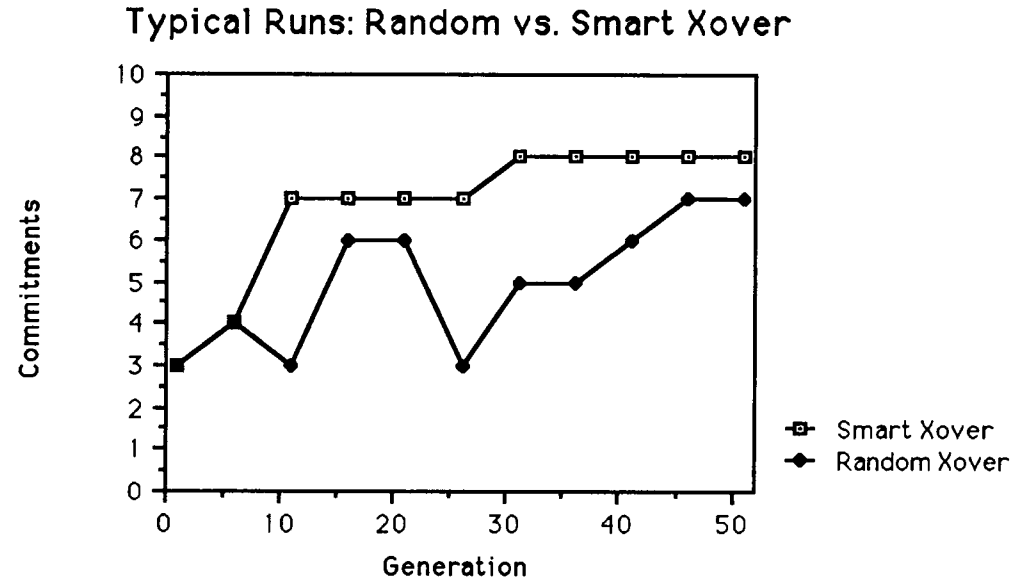

Figure 3. Smart vs Random Crossover. The data presented here were selected from "average" runs for both algorithms. The solution goal was 10 commitments.

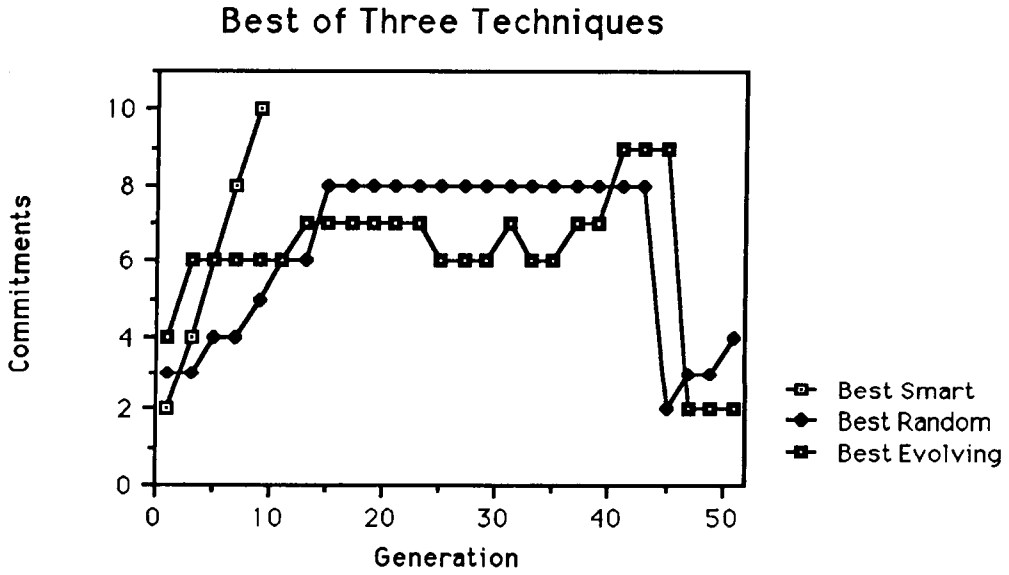

Figure 4. Random vs Evolving vs Smart Crossover. For each algorithm, the best (highest fitness) run has been plotted. The number of trials was 10 and the solution goal was 10 commitments.

5.1. Smart Crossover

Table 1 compares the performance of Smart versus Random Crossover. In all trials, the solution goal was 30 commitments, the population size was thirty, and the number of generations was thirty-one. The data indicate that the Smart Crossover produces statistically better results. The null hypothesis (see Section 4.1) is rejected and the true hypothesis is supported.

Xover	No. of Trials	Mean	Stand Dev	Z stat
Random	10	19.8	1.6	
Smart	10	25.1	1.79	6.98

Table 1. Random versus Smart Crossover. The z statistic has been calculated to determine whether the trial samples have been drawn from the same population and seem to indicate otherwise.

5.2. Crossover Point Evolution

The comparison of Random and Evolving Crossover Point algorithm is summarized in Table 2. The two populations are not significantly different and so the null hypothesis (see Section 4.2) regarding this technique is supported.

Xover	No. of Trials	Mean	Stand Dev	Z stat
Random	10	19.8	1.6	
Evolving	10	19.5	0.7	0.5

Table 2. Random versus Evolving Crossover. The z statistic has been calculated to determine whether the trial samples have been drawn from the same population and this is indicated.

5.3. GA vs NN

Tables 3 and 4 contain the results of comparisons of the two algorithms. The neural network algorithm, when applied to the same proposal as the Genetic Algorithms, arrived at solutions in much less time (generally a few seconds); the GA took many minutes to compute solutions (or near solutions). The solutions that were acheived by the NN tended to be better than the GA solutions.

Algorithm	N	Mean Secs to Soln	Stand Dev	Z stat
NN	5	4.8	0.8	
GA	5	1435.0	133.0	24

Table 3. Comparison of two search optimization algorithms with respect to speed. In all trials, >90% commitment of activities was achieved. The Z statistic indicates that the samples are not from the same population.

Algorithm	N	Mean Suitability	Stand Dev	Z stat
NN	5	18.0	0.1	
GA	5	9.9	1.3	13.9

Table 4. Comparison of GA and NN optimization techniques with respect to schedule suitability. The Z statistic indicates that the samples are not from the same population.

6. Discussion

The experimental results concerning the Smart Crossover algorithm indicate that this method tends to more quickly approach a solution; exploitation seems to be favored at the expense of exploration however and this may reduce the chance of finding a global optimum. The rapid movement by the GA to a solution may be a good thing in this scheduling domain; the computational expense of the GA (and the associated fitness testing) requires that population size and number of generations be kept to reasonable numbers.

The Random Cossover algorithm seems to more casually explore the search space. Looking at Figure 3, one sees empirical evidence for this in the oscillations and occasional radical backtracking to low fitness populations. The latter occurs after long periods of little change (and so may represent the probability that a higher than average number of mutations occurred in a single generation).

It is unclear why the Evolving Crossover Point performed no better than Random Crossover. Tools for tracing the evolution of single chromosomes have been proposed and could perhaps aid in "debugging" GA runs.

Continued work on this application of GA technology may yield better performing systems. There is a wealth of domain knowledge (constraint specifications) that can be tapped and so the design of a competition-based rule induction system should be a part of future work.

Acknowledgements

The author thanks the following persons for their ideas and comments concerning this work: Lashon Booker, Ken De Jong, John Greffenstette, Mark Johnston, Glenn Miller, and Shon Vick.

References

Baker, J. (1988). Reducing bias and inefficiency in the selection algorithm. *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 14-21). Hillsdale, NJ: Lawrence Erlbaum Associates, Pub.

Baffes, P. and Wang, Lui (1988). Mobile transporter path planning using a genetic algorithm approach. *Proceedings of the SPIE Cambridge Symposium on Advances in Intelligent Robotics Systems*, vol. 1006 (pp. 226-234).

De Jong, K. (1980). Adaptive system design: a genetic approach. *IEEE Transactions on Systems, Man, and Cybernetics* , SMC-10(9) (pp. 566-574).

Grefenstette, J. (1988). Credit assignment in genetic learning systems. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 596-600). San Mateo, CA: Morgan Kaufman Publishers, Inc.

Goldberg, D. (1988). Genetic algorithms in search, optimization, and machine learning. Reading, MA: Addison-Wesley Pub. Co. Inc.

Hilliard, M., G. Liepins, and M. Palmer (1988). A classifier-based system for discovering scheduling heuristics. *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 231-325). Hillsdale, NJ: Lawrence Erlbaum Associates, Pub.

Holland, J. (1975). *Adaptation In Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Johnston, M. and H. Adorf (1989). Learning in stochastic neural networks for constraint satisfaction problems. *Proceedings of the NASA Conf. on Space Telerobotics* .

Miller, G., M. Johnston, S. Vick, J. Sponsler, and K. Lindenmayer (1988). Knowledge based tools for Hubble Space Telescope planning and scheduling: constraints and strategies. *Proceedings of the 1988 Goddard Conference On Space Applications of Artificial Intelligence*. Reprinted in *Telematics and Informatics* 5 (1988) (pp. 197-212).

Schaffer, J. and Morishima, A. (1988). Adaptive knowledge representation: a content sensitive recombination mechanism for genetic algorithms. *International Journal of Intelligent Systems*, Vol 3 (pp. 229-246).