Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator

Darrell Whitley, Tim Starkweather and D'Ann Fuquay
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523 USA
(303) 491-1463
whitley@cs.colostate.edu

Abstract

This paper outlines a new approach to generating solutions to Traveling Salesman Problems. A new operator is introduced which recombines the edges (or links) between cities from two parents to create a single new offspring. This operator is different from past operators that emphasized edges in that 95compose the offspring are inherited from one of the two parents. The recombination operator does not use information about the distance between cities or any heuristic operators. This operator has found known solutions and matched "best known" solutions for every problem on which it has been tested. The functionality of this new operator can be explained using a variation on Holland's original schema theorem. Because this operator uses no information about the actual cost associated with edges and requires the use of no additional heuristics, it can also be used on sequencing problems where there are no actual distances to measure, but rather only some overall evaluation of the total sequence. An example is given of how this operator has been used to generate job shop schedules.

1 INTRODUCTION

The theory behind genetic algorithms is well developed for problems that can be encoded as a binary string with no ordering dependencies. However, many potential applications of genetic algorithms involve complex ordering dependencies similar to those found in the Traveling Salesman Problem. The Traveling Salesman Problem involves finding the shortest Hamiltonian path or cycle in a complete graph of N nodes. This problem is a classic example of an NP-hard problem; all known methods of finding an exact solution involve searching a solution space that grows exponentially with N, the number of nodes in the graph. Because the solution space is too large to actually search when N is not trivially small, methods of heuristically searching this space have been developed.

All of the results reported in this paper were obtained using the G*ENITOR* package developed as part of the G*ENITOR* project at Colorado State University. This genetic algorithm uses one-at-a-time recombination, where the newly created offspring replaces the lowest ranking individual in the population rather than a parent; this algorithm also allocates reproductive trials according to the rank of the individual in the population rather than using fitness proportionate reproduction. We have found this approach yields substantially better results than standard genetic algorithms, especially on problems with larger encodings. We have not done any comparison of the G*ENITOR* approach versus the standard genetic algorithm on the problems reported here, but have published comparisons on (1) DeJong's standard test suite (Whitley 1988) and (2) on neural networks optimization problems (Whitley 1989a; 1989b).

There have been many attempts to tackle the Traveling Salesman Problem using genetic algorithms, all with modest success. In reviewing these attempts it was found that many operators concentrated on manipulating the "cities" in the tour, rather than the "links" between cities. However, there are very good reasons to believe this approach is not the best way to optimize the problem. If the search space for a Traveling Salesman Problem is viewed as a network, where the cities are nodes and the distances between cities are the weights on the arcs (i.e. the links or graph edges), then it is clear that optimizing a Traveling Salesman Problem involves finding the best combined set of edges that constitute a legal tour. What is important is not that a particular city occurs in a particular position, but rather that the genetic operations preserve and exploit critical links that contribute to the minimization of the overall tour. The methods that concentrate on exploiting information about city positions often do a very poor job of preserving information about these edges. The new recombination operator emphasizes edge recombination.

John Grefenstette has developed an operator that emphasizes edges, but he reports that transferring edges from parents to offspring is only 60are randomly picked (Grefenstette 1987). We have developed a new operator which is 95-99to a single new offspring. We refer to this operator as "edge mapped recombination" or simply "edge recombination."

The theoretical foundations of genetic algorithms assume there exists some (ideally binary) representation of a problem that can be manipulated by genetic operators. Each problem encoding is referred to as a "genotype." When the problem can be represented as an n-bit string, it can be shown that genetic algorithms sample hyperplanes in an n-dimensional hypercube. Each potential problem solution corresponds to a corner in the hypercube.

Unfortunately, there is no practical way to encode a Traveling Salesman Problem as a binary string that does not have ordering dependencies or to which operators can be applied in a meaningful fashion. Simply crossing strings of cities produces duplicates and omissions. Thus, to solve this problem some variation on standard genetic crossover must be used. The ideal recombination operator should recombine critical information from the parent structures in a non-destructive, meaningful manner.

The operator we have devised not only produces legal tours with very little disruption of edges, but we have also been able to show that a binary representation exists which can be used to track how "edge recombination" moves the search to different corners of a hypercube that corresponds to the problem solution space. The binary representation also shows that appropriate schemata are being sampled. This binary representation makes it possible to express what happens when searching the Traveling Salesman Problem solution space in terms that closely correspond to Holland's original schema theorem. However, these binary representations cannot be recombined without additional information about what is and is not a legal tour; the edge recombination operator ensures that a legal tour is produced.

2 Existing Operators for Traveling Salesmen

The first efforts to find near optimal solutions to Traveling Salesman Problems using genetic algorithms are those of (Goldberg 1985) and (Grefenstette 1985), although (Davis 1985) also deserves equal credit for working on scheduling problems which have the same kind of ordering-dependent constraints. Most of these approaches generally work in the following way. (There is some variation, depending on which approach one considers. Grefenstette's approach is particularly different.) The problem is represented as a list of cities, where the order indicates the sequence in which the cities are visited. Each list is treated as a "genotype" to which the genetic operators are to be applied. Notice that the actual cities are manipulated rather than some underlying binary representation. These studies, which achieved modest but encouraging results, tried to more or less ignore the difference between the Traveling Salesman Problem and the kind of ordering *independent* problems to which genetic algorithms are more naturally suited. In other words, tours are simply crossed to produce offspring tours, even though this typically results in illegal tours–that is, tours which visit some cities twice and some cities not as all. To deal with this problem, the offspring tours are corrected so that the duplicate cities are replaced by the omitted cites or otherwise eliminated.

In 1987 at the Second International Conference on Genetic Algorithms there were several papers that suggested some new approach to the Traveling Salesman Problem. Most of these involve the use of some additional heuristics (Liepins 1987; Suh 1987). One paper, however, pursued a different strategy. This work, (Oliver 1987), is of particular interest, not because the results were better than others (in fact they were not as good) but because a new crossover operator was specially constructed so as to always produce legal tours and always use information extracted from one of the two parent structures. The operator was referred to as a "cycle crossover operator." It works in the following manner. Consider the tour [A B C D E] and [E D A B C]. These (very short) tours contain two cycles: "A-C-E" and "E-A-C" is one while "-B-D-" and "-D-B-" is the other. Thus, a cycle refers to a common subset of cities in the two parent tours that occupy a common subset of positions. The cycles A-C-E and E-A-C both contain the cities A,C, and E in the positions 1-3-5; note that order is not important. What is important is that cycles can be exchanged between parents without introducing any city duplicates or omissions. The offspring in this example are [A D C B E] and [E B A D C]. The resulting offspring is guaranteed to inherit all of its city *positions* from one of the two parents. (Finding a cycle is relatively simple. See [Oliver 1987] for an explanation.)

Unfortunately preserving city positions often tends to destroy links that existed in the parent structures. Cycles tend not to correspond to tour subchunks. The idea of exclusively extracting information from parents to generate offspring is a good one; unfortunately, the wrong information is extracted. The fact that this operator performs rather poorly also reinforces the notion that it is edges which are critical, not city positions. On the other hand, this operator might be very useful for other kinds of sequencing problems where *position* in the sequence is critical.

3 The Case for Edges

Another approach to the Traveling Salesman Problem is to use inversion rather than crossover. Inversion involves selecting two cities in the tour and reversing the sublist in between. Thus inversion breaks only two links. The motivation to use inversion is that it is a known genetic operator which theoretically should be useful for finding good string orderings (Holland 1975) and is always guaranteed to produce a legal tour. The drawback to using inversion alone is that it takes information from only one parent, and thus does not achieve information recombination. This means that it does not carry with it the search power that results from implicit parallelism; recombination is necessary to exploit the power of genetic algorithms.

Nevertheless, inversion proved to be a reasonably good operator. Using inversion on a 50 city Traveling Salesman Problem we found that it out performed a "cross and correct" operator.

Another possibility is to select three or more break points and invert each sublist, or actually swap sublists. Again, these operations are performed on one parent, so a legal tour always results. We found that *increasing the number of break points degrades performance.* The fewer breaks that occur, the more the algorithm preserves the "edges" which exist in the parent tour. Each broken edge means that a new, random edge is created; this suggests that operators which break fewer edges do a better job of finding good solutions. This also suggests that a crossover operator should try to preserve edges. Ideally, every edge in the offspring will be taken from one of the parent structures. Every edge that is not derived from the parents strings represents a point of mutation in the offspring, and thus represents a random, unguided move in the search space.

The cycle crossover operator guarantees that each city occurs in a position identical to that in one of the two parents. However, by empirically studying what this does in a number of test problems, we found that it actually destroys more edges than the "cross and correct" operators. Furthermore, we found that Davis's "order crossover" seems to break fewer links than Goldberg's PMX crossover operator. Oliver et al. (1987) found that of these three operators, the "order crossover" gave the best performance, followed by the PMX operator, with "cycle crossover" doing the poorest of the three at optimizing Traveling Salesman Problems. While our empirical study of these operators was on a limited number of cases and should be taken as only being suggestive, it was found that the differences in performance appeared to correspond to the ability of the operators to preserve edges. This informal observation lead us to form the following general hypothesis: to improve the performance of the genetic algorithm at solving Traveling Salesman Problem one must develop a recombination operator that transfers edges from parents to offspring as effectively as possible. To test this hypothesis we developed the "edge recombination" operator.

4 Implementation and Results

Consider the following tours: [A B C D E F] and [B D C A E F]. If we represent the "link" information in the tour we have "ab bc cd de ef af" and "bd dc ca ae ef bf" as parent tours. We assume that the tours are circular–the journey finishes again at the initial city thus forming a Hamiltonian cycle. (A Hamiltonian path visits every city, but does not "close the loop" and return to the origin.) Also note that edge "ab" has exactly the same value as "ba." All that is important is the value of the link, not its direction. Thus, an ideal operator should construct an offspring tour by exclusively using links present in the two parent structures. On average, these edges should reflect the goodness of the parent structure. There is no random information that might drive the search toward arbitrary links in the search space. Thus, an operator that preserves edges will exploit a maximal amount of information from the parent structures. Operators that break links introduce unwanted mutation. This mutation can be thought of as a kind of "leak" in the search process. Applying search pressure is not maximally effective if past optimization is lost due to random loss of edges.

The edge recombination operator which has been developed uses an "edge map" to construct an offspring that inherits as much information as possible from the parent structures. This edge map stores all the connections from the two parents that lead into and out of a city. Since the distance is the same between two cities whether one is coming or going, each city will have at least two and at most four edge associations–two from each parent. Consider these tours: [A B C D E F] and [B D C A E F]. The edge map for these tours is as follows:

```
A has edges to : B F C E
B has edges to : A C D F
C has edges to : B D A
D has edges to : C E B
E has edges to : D F A
F has edges to : A E B
```

The problem that has normally occurred with edge recombination is that cities are often left without a continuing edge. Thus they become isolated so a new edge has to be reintroduced. Using an edge map it is possible to choose "next cities" in such a way that those cities which currently have the fewest unused edges (and thus appear to have the highest probability of being isolated) have priority for being chosen next. Of course, there must be a connecting edge to a city before it is actually a candidate, but at any point in time there are up to three candidates (and up to four in the case of the initial city), and the candidate with the fewest edge connections is chosen; in the case of a tie the algorithm chooses randomly from among the candidate edges.

To begin with, the new "child" tour is initialized with one of the two initial cities from its parents. Theoretically, it does not matter which city is chosen to initialize the tour, since the tours are circular. However, we have only tested the algorithm using one of the two intial cities from the parents to initialize the offspring. Also, we could perhaps simplify the problem by "fixing" the initial city so that it is always the same. Again we have not tried this, but have made the following observations. Fixing the first city so that it is alway the same could possibly make the problem easier to solve, since fixing the first city theoretical reduces the search space. (By fixing the first city, we leave only (N-1)! number of combinations to consider.) But fixing the first city could also cause problems: the one edge that the edge recombination operator fails to enforce is the edge that goes from the final city to the initial city. On Hamiltonian paths, this is not an issue, but on Hamiltonian cycles it means that a limited amount of mutation is going to occur (on at most 1 link in the offspring, i.e. at a rate of 1/N). Fixing the first city would always force the mutation to occur at the same place. By allowing different cities to occupy the initial position in different genotypes, this mutation is distributed over the various links. From an analytical perspective, this is perhaps the greatest weakness we have found in the operator. We have not attempted, however, to empirically study the impact of this weakness.

Tracing an example recombination helps to explain how recombination proceeds. First an initial city is chosen. In the above case A and B both have four edges, so the algorithm will make a random choice. Assume B is chosen. The candidates for the next city are A C D and F. C, D and F all have two edges (the initial three minus B–A now has three edges and thus is not considered) so randomly choose between C, D and F. Assume C is chosen. C now has edges to A and D. D is chosen next, since it has fewer edges. D only has an edge to E, so E is chosen next. E has edges to A and F, both of which have one edge left. Randomly choose A. A must now go to F. The resulting tour is [B C D E A F] and is composed entirely of edges taken from the two parents. As already noted, a thousand trial recombinations on a 30 city problem resulted in only 278 new edges being introduced out of the 30,000 edges manipulated. This is an effective mutation rate of .009, or less than 1problem. This time 753 new edges were introduced, for a mutation rate of 1.5up slightly when longer tours are recombined. Note that these disruption rates apply to tours representing Hamiltonian paths; the rate for tours representing Hamiltonian cycles will be slightly higher, but this increase will be less for tours with more cities. The additional mutations will be $<= \ 1/N$, where N represents the number of cities in the tour; thus even for Hamiltonian cycles the edge transfer will be 96

To date, the operator has been applied to three Traveling Salesman Problems: a 30 city problem, a 50 city problem, and a 75 city problem. These problems are part of a package put together by G. Liepins of the Oak Ridge National Laboratory for testing new approaches to the Traveling Salesman Problem. The package gives "best known"/optimal solutions taken from the literature. *Edge recombination found solutions slightly better than "best known" on all three problems.* Table 1 presents information about the results of the G*ENITOR* runs, giving both best and average performance. As indicated in the Table, the average performance compares well with the best performance. The average performance indicates that even when the algorithm failed to match best known, the performance is competitive. Notice that G*ENITOR* found new "best known" on the 30 city, 50 city and 75 city problems. We have verified that these results are accurate given the evaluation function we are using. The problems originally appear in Eilon (1969) and Lin/Kernighan (1973). Perhaps round-off error could account for the difference, since distances are calculated from the (x,y) coordinates of a city.[1] Our evaluation function does, however, return the correct evaluation for the best known tour on the 30 city problem; the actual sequence has been published for this tour. In other words, our evaluation function correctly returns 424 for the previously best known sequence, which suggests that it is accurate. We do not have the actual sequence for the best known tours for the 50 and 75 city problems. As far as we have been able to determine, these new best known solutions are valid previously unknown solutions and not merely an artifact of the evaluation function. To aid comparisons, however, the sequences for these new best known are given in Appendix One along with our evaluation function.

We devised the 36 city problem in an attempt to further test the algorithm on a problem with a known solution.

[1] This footnote was added after publication. There was indeed a round-off error in one version of TSPlib in 1989. The results do not improve on previously best known.

Source	N	BestKnown	GENITOR	Average	Pool.size	Crosses	Bias
Oliver	30	424	421	437	250	3,200	2.3
—-	36	36	36	36	300	5,600	1.7
Eilon	50	430	428	439	600	25,000	2.0
Eilon	75	553	545	559	1,000	80,000	2.0

Table 1: Traveling Salesman Results: "Source" identifies the reference in which the "bestknown" value appears. "N" indicates the number of cities. The "G$ENITOR$" column presents the best results achieved with the edge recombination operator. Average gives the mean performance over 10 runs for the edge recombination operator. "Crosses" indicates the number of recombinations per run; "Pool.size" represents the population size. "Bias" refers to the selective pressure used (See Whitley 1989b).

This problem involves finding a tour in a 6 by 6 grid of evenly spaced points. Any four evenly neighboring points that form a square have a edge distance of 1 between them. The diagonal of the square rounds off to an integer value of 1. To differentiate between edges and diagonals the evaluation was altered to use real values. Note that the optimal solution for this problem is 36, since there is no edge in the problem less that 1, and 36 edges are required to connect the 36 nodes. The configuration does not appear to contain anything that would make the genetic algorithm's job easier; nevertheless, the genetic algorithm found optimal solutions with no difficulty on every run. This may suggest that the problem is somewhat easier than the other problems, perhaps because several solutions exist.

Although we have yet to conduct a thorough comparative analyses, we believe this new operator works better than other genetic approaches to the Traveling Salesman Problem and may be competitive with other heuristic methods. However, its greatest potential is probably not for solving Traveling Salesman problems, but rather on other ordering problems where local sequencing evaluation is not possible (i.e. there is no local edge information), and thus where more traditional heuristic are not applicable. Genetic edge recombination can be used in such domains because it requires no local information, only feedback about the total evaluation of some string.

5 Edge Recombination and Schema Theory

In many ways the edge recombination operator conforms much more closely to the standard theory of genetic algorithms than other "order-preserving" operators. In particular, it removes the large amounts of mutation that occur with the other methods. Oliver et al. [1987] tried to remove this bias, but failed to do so in an appropriate way for the Traveling Salesman Problem. We have also been able to show that the operator works on an underlying representation that is binary.

Because city positions are not critical to this operator, the normal analysis of positionally defined schemata does not directly apply. But we also believe that "ordering schemata" which have previously been used to analyze operators for the Traveling Salesman Problem are also inappropriate. We propose the following. Rather than optimizing for positions or order, the algorithm should provably allocate more reproductive trials to high performance edges and find critical edge combinations. An underlying binary representation exists for this edge information, which means this new recombination operator can be related to the same notions on which Holland's original schema theorem is built.

The following characterization of the Traveling Salesman Problem presents how the solution space of all possible edges can be represented in binary form. The tour [A B C D E F] has the following edges: ab, bc, cd, de, and ef. Recall that the edge "ef" is exactly the same as the edge "fe." Therefore all possible edges in this six city problem are:

```
ab ac ad ae af bc bd be bf cd ce cf de df ef
```

The Hamiltonian cycle [A B C D E F] can be represented as a binary mask that indicates which links are present. This is illustrated in the following table.

```
ab ac ad ae af bc bd be bf cd ce cf de df ef
 1  0  0  0  1  1  0  0  0  1  0  0  1  0  1
```

This binary string represents a corner in a hypercube. However, as one would expect, not all corners in this hypercube are legal solutions. Strings which constitute legal tours have exactly N "1" bits, since a Hamiltonian cycle with N cities (or vertices) must have N edges. Furthermore, of these binary strings, only those that visit every city (and thus have no duplicates) are legal. This means the bits must be distributed so that there is an edge into and out of each city. This constrains the job of the recombination operator, but in no way compromises the fundamental theorem of genetic algorithms–we still want to recombine these binary strings in such a way as to allocate more trials to those hyperplanes in the search space that display above average performance. Given this characterization of the problem, edge recombination does exactly that. Consider again the tours [A B C D E F] and [B D C A E F]. We now add this second tour as the second binary mask in the our table.

ab	ac	ad	ae	af	bc	bd	be	bf	cd	ce	cf	de	df	ef
1	0	0	0	1	1	0	0	0	1	0	0	1	0	1
0	0	0	1	0	1	0	1	1	1	0	0	0	1	0

Now consider the offspring which was created using the edge recombination in the example presented earlier: [B C D E A F]. We add the mask for this tour as the third binary string in our table.

ab	ac	ad	ae	af	bc	bd	be	bf	cd	ce	cf	de	df	ef
1	0	0	0	1	1	0	0	0	1	0	0	1	0	1
0	0	0	1	0	1	0	1	1	1	0	0	0	1	0
0	0	0	1	1	1	0	0	1	1	0	0	1	0	0

The recombination that took place to produce this offspring can be thought of as a "bit-by-bit" recombination, where each bit is picked from one of the two parents. Also, note that where the parents have the same bit, the offspring will have the same bit. To simplify the binary mask, we remove all those bit positions where neither parent has an edge, and indicate how recombination occurred. (Note that if we also removed those links that appear in both parents, then we would have Booker's [1987] "reduced surrogates.")

ab	ac	ad	ae	af	bc	bd	be	bf	cd	ce	cf	de	df	ef
1	-	-	0	(1)	(1)	-	(0)	0	1	-	-	1	(0)	1
(0)	-	-	(1)	0	1	-	1	(1)	(1)	-	-	0	1	(0)
0	-	-	1	1	1	-	0	1	1	-	-	1	0	0

Clearly edge recombination is manipulating schemata in this representation space. If genotypes actually used this representation the binary strings would be much larger than the list of cities: for N cities the binary representation has length $(Nsup2 - N)/2$. Furthermore, there is no clue as to which recombinations result in legal tours. This characterization of the solution space, however, makes it clear that edge recombination preserves schemata. There is no need for any new notion of "schema" with its own special theorem. All that is needed is a version of the schema theorem that allows one-at-a-time reproduction (which the G$ENITOR$ schema theorem does [Whitley 1988]) and a variation on the theorem that allows bit-by-bit recombination (which we are currently developing).

6 A Job Shop Scheduling Application

As previously noted, this approach to generating potential solutions to the Traveling Salesman Problem is particularly interesting in that *it does not use any information about the distance between cities, only the "cost," or "performance" of the overall tour.* This is important, because it means the method may be used on sequencing problems where there are no actual distances to measure, but rather only some overall evaluation of the total sequence.

We have used this recombination operator to drive a job shop scheduler. (The job shop is modeled after a board assembly line at Hewlett Packard in Fort Collins, CO. Information about the line was supplied by Daniel Shaner.) Davis [1985] has also worked on scheduling problems using genetic algorithms.

The problem we are working on is such that the solution space is highly constrained once an initial machine is scheduled. If a good sequence of jobs could be found then the remaining machines can be scheduled using a few simple rules that act as a greedy scheduler with some look ahead capabilities. The approach used was as follows: the schedule for the first machine was treated as a sequencing problem similar to the Traveling Salesman Problem. In other words: How should jobs be sequenced in order to best schedule the job shop in the context of the greedy scheduler? The greedy scheduler uses information about the initial job sequence to produce a complete schedule which is then evaluated. This evaluation is then assigned as the "performance" value associated with the initial sequence. Thus, the goal of recombination was to find a sequence which optimized the performance of the final schedule in the environment of the secondary greedy scheduler. Again, note that edge information does not explicitly exist. This means that methods for the Traveling Salesman Problem that use information about the relative cost of edges would not be appropriate, since each edge change would require that the entire schedule be reconstructed and reevaluated.

The strategy behind the scheduler implementation is as follows. (1) Using the new recombination operator, the genetic algorithm produces a "primary schedule" indicating the sequence in which jobs are loaded onto the first machine. (2) The "secondary" rule based scheduler then generates a full schedule. The rules for the secondary scheduler are relatively simple since the problem is highly constrained by the availability of jobs as they come off the first machine. (3) The schedule evaluator then assigns a performance value to the full schedule. This will be a *de facto* evaluation of the primary schedule. This value is passed to the genetic algorithm as the "fitness" of the "primary schedule." This serves as the "fitness" used to rank the genotypes and thus to allocate reproductive trials in a population of "job sequence" genotypes. The idea is that this feedback provides information about how the primary schedule affects all subsequent machines schedules.

The rules used to implement the secondary scheduler do not actually look ahead, but rather use a "wait and see" approach similar to that used by "wait and see" parsers. If a machine is already setup to run "Job X" and a job X is available, then it is scheduled. If a machine is setup to run "Job X" and X is unavailable then an "idle time" is introduced. The scheduler will wait and see if this idle time can be filled later by another job. If X becomes available before another job can be setup, then the scheduler waits for X. However, if at some later point in time X is still not available and another Job Y could have been setup and started, then the idle time is "back-filled" by scheduling a setup for the new Job Y. If there are multiple jobs that could be setup in place of X (the unavailable job the machine is currently setup to run), then further comparisons are made. First, the machine decides which jobs have the shortest setup time. If two or more jobs have identical setup times, the one that has the shorter execution time is chosen.

The most complicated and difficult problem which has been attacked to date involves 20 different types of jobs, scheduling 10 jobs of each type. Numerous small simulations have been run; the results obtained on this example is typical of the other simulations. Thus the primary schedule consists of a sequence of 200 jobs. The complete expanded schedule involves 6 different types of machines with 2 identical machines belonging to each type. This means that 12 machines in all are being scheduled and that the scheduling dependencies are 6 machines deep. In order words, the scheduling of machine F is dependent on the scheduling of machine E, which is dependent on D, which is dependent on C, which is dependent of B, which is dependent on the primary scheduling of machine A. Note that the genetic algorithm only schedules A, that all the other machines are scheduled using the simple rules outlined above, but that the total schedule evaluation is used by the genetic algorithm. We stress the depth of these dependencies because of concern that the genetic algorithm might not generate primary schedules that are sensitive to such distant and indirect dependencies. The idea, however, is that by using the total schedule evaluations to assign a "fitness value" to the primary schedule that selective pressure will favor primary schedules that produce a balanced optimization across all of the machines.

We should also explain how the job sequencing problem was encoded as a sequencing problem. An actual sequence might be something like the following: [A A A B B A A C C]. Note that job types reoccur. Our strategy for encoding the problem was to simply ignore the fact some of the jobs are the same and thus can change positions without altering the schedule. Thus, [A A A B B A A C C] is encoded as [A1 A2 A3 B1 B2 A4 A5 C1 C2]. This means we are not using all the information available to us; nevertheless, the results are positive. It may be possible to devise a version of the edge recombination operator that exploits this additional information, but we have not vigorously explored this possibility. The results indicates that the operator works quite well even without this additional information.

Different setup times and run times for each type of job on each type of machine were specified. Bottlenecks were created for select jobs on certain machines in the processing sequence. In the simulation, each machine has an associated cost and time variable for job setup, for running a job, or for being idle. These costs and time measures are used to evaluate the complete schedule.

For the problem presented here a theoretical limit can be calculated which, although it can not be achieved, gives a lower bound on the cost function. This theoretical limit is based on the assumptions of zero idle time on each machine and only one setup per machine per job type. We know for certain that this limit is impossible to reach. In the schedules evaluated here there is a necessary initial idle time on every machine except the first machine; machines that occur later in the production sequence must remain idle while the first job trickles through the line. We can start the simulation with jobs already distributed down the line (i.e. a schedule that "picks up where it left off yesterday") but have not done this for our test runs.

A comparative search was also carried out without attempting to optimize the initial schedule. The initial schedule order was generated at random, and then the complete schedule was expanded using the secondary scheduler. This results in a "greedy" search of the problem space and gives a crude upper bound to the cost function. We expected the genetic algorithm to do substantially better. Also, this method was applied iteratively to gauge the difficulty of finding an appropriate initial sequence and to try the "greedy" scheduler from numerous different initial positions.

After 50,000 recombinations, the genetic algorithm generated a schedule within 16.8 percent of the theoretical limit (a cost of 44,111 for the genetic algorithm versus 37,763 for the limit), whereas after 100,000 iterations of the "greedy" scheduler from different initial positions, the best was only within 34.8 percent of the theoretical limit (a cost of 50,897 for the "greedy" scheduler). Figure 1 shows an example of the limit, the greedy scheduler and the genetic algorithm schedule. As can be seen, the genetic algorithm algorithm makes steady progress at finding scheduling improvements. The best schedules for the job shop scheduling simulations have been studied and traced by hand; they do very well at avoiding idle time and setups.

FIGURE 1. The optimization of a job shop scheduler using the genetic edge recombination operator. This graph shows the theoretical limit, the result of running the greedy secondary scheduler from multiple initial positions, and the results produced by using the genetic algorithm G*ENITOR*. One trial implies 1 recombination (the generation of one new offspring) in the genetic algorithm. One trial implies 1 run of the greedy scheduler from a different initial position. The theoretical limit is constant.

7 Conclusions

The new edge recombination operator for the Traveling Salesman Problem has produced impressive results on a number of different problems. Furthermore, the theoretical analysis shows that the search space for this problem can be modeled as a binary hyperspace. This means the functionality of the recombination operator can be explained using a variation on traditional schema theory. However, good heuristic methods already exist for the Traveling Salesman Problem. We believe that this new recombination operator may prove most useful on sequencing problems that are similar to the Traveling Salesman Problems, but where it is not possible to obtain "local" information such as the cost associated with individual "edges." This new operator is able to generate very good sequences given no other information except the cost of the entire sequence. The applications of edge recombination to a job shop sequencing problem demonstrates the potential of this approach in novel domains. The job shop scheduler also demonstrates that it is possible to achieve balanced optimization in a situation where the genetic algorithm is only generating partial solutions, but being evaluated according to the performance of the fully expanded solutions. This also freed us from having to develop an encoding for entire schedules–a task that often forces ad hoc representations that are not theoretically well founded.

We have tried to point out some of the open research questions while discussing the results presented in this paper. As noted earlier in this paper, we have not done any comparisons on this problem between G*ENITOR* and a standard genetic algorithm. However, our experience is that G*ENITOR* consistently out-performs a standard genetic algorithm on problems with larger encodings (more than 50 bits). It is possible that the results presented in these paper are in part attributable to the G*ENITOR* algorithm itself. We will of course test this hypothesis in future work. For a more thorough discussion of the G*ENITOR* algorithm and its theoretical foundations, see (Whitley 1988, Whitley 1989a, Whitley 1989b).

ACKNOWLEDGEMENTS

This research was supported in part by a grant from the Colorado Institute of Artificial Intelligence (CIAI). CIAI is sponsored in part by the Colorado Advanced Technology Institute (CATI), an agency of the State of Colorado. CATI promotes advanced technology education and research at universities in Colorado for the purpose of economic development.

APPENDIX ONE

The sequence of coordinates for the various problems are as follows.

The new best solution (length 421) for Oliver's 30 city problem, giving the coordinates in order of the cities visited: (54,67) (54,62) (37,84) (41,94) (2,99) (7,64) (25,62) (22,60) (18,54) (4,50) (13,40) (18,40) (24,42) (25,38) (44,35) (41,26) (45,21) (58,35) (62,32) (82,7) (91,38) (83,46) (71,44) (64,60) (68,58) (83,69) (87,76) (74,78) (71,71) (58,69)

The new best solution (length 428) for Eilon's 50 city problem, giving the coordinates in order of the cities visited: (37,69) (27,68) (31,62) (42,57) (37,52) (38,46) (42,41) (45,35) (40,30) (32,22) (27,23) (20,26) (17,33) (25,32) (31,32) (32,39) (30,48) (21,47) (25,55) (16,57) (17,63) (5,64) (8,52) (12,42) (7,38) (5,25) (10,17) (5,6) (13,13) (21,10) (30,15) (36,16) (39,10) (46,10) (59,15) (51,21) (58,27) (48,28) (52,33) (52,41) (56,37) (61,33) (62,42) (58,48) (49,49) (57,58) (62,63) (63,69) (52,64) (43,67)

The new best solution (length 545) for Eilon's 75 city problem, giving the coordinates in order of the cities visited: (48,21) (52,26) (50,30) (55,34) (54,38) (50,40) (51,42) (55,45) (55,50) (50,50) (41,46) (45,42) (45,35) (40,37) (38,33) (33,34) (29,39) (33,44) (35,51) (30,50) (22,53) (21,48) (21,45) (21,36) (20,30) (26,29) (22,22) (27,24) (30,20) (35,16) (36,6) (26,13) (15,5) (15,14) (16,19) (12,17) (6,25) (11,28) (12,38) (7,43) (9,56) (15,56) (10,70) (17,64) (26,59) (30,60) (31,76) (40,66) (35,60) (40,60) (47,66) (50,70) (55,65) (57,72) (70,64) (62,57) (55,57) (62,48) (67,41) (62,35) (65,27) (62,24) (55,20) (60,15) (66,14) (66,8) (64,4) (59,5) (50,4) (54,10) (50,15) (44,13) (40,20) (36,26) (43,26)

REFERENCES

[Booker 1987] Booker, L. Improving search in genetic algorithms, in: Lawrence Davis (Ed.), *Genetic Algorithms and Simulated Annealing.* (Morgan Kaufmann, 1987) 61-73.

[Davis 1985] Davis, L. Job shop scheduling with genetic algorithms, in: John Grefenstette (Ed.), *Proc. of an International Conf. on Genetic Algorithms and Their Applications.* (L. Erlbaum, 1988, original proceedings 1985) 136-140.

[Eilon 1969] Eilon, Watson-Gandy, and Christofides. Distribution management: mathematical modeling and practical analysis, *Operational Research Quarterly* 20: 309.

[Goldberg 1985] David Goldberg and R. Lingle. Alleles, loci, and the traveling salesman problem, in: John Grefenstette (Ed.), *Proc. of an International Conf. on Genetic Algorithms and Their Applications.* (L. Erlbaum, 1988, original proceedings 1985) 154-159.

[Grefenstette 1985] J. Grefenstette, R Gopal, B. Rosmaita, D. van Gucht. Genetic algorithms for the traveling salesman problem. Proc. Intern. Conf. of Genetic Algorithms and their Applications. John Grefenstette, ed. Pp: 160-165.

[Grefenstette 1987] John Grefenstette. Incorporating Problem Specific Knowledge in Genetic Algorithms. IN Genetic Algorithms and Simulated Annealing. Lawrence Davis, ed. Pp: 42-60.

[Holland 1975] Holland, J. *Adaptation in Natural and Artificial Systems.* (Univ. of Michigan Press, Ann Arbor, 1975).

[Liepins 1987] Liepins G., Hilliard M., Palmer M., and Morrow M. Greedy genetics, in: John Grefenstette (Ed.), *Genetic Algorithms and their Applications: Proc. of the Second International Conf.* (L. Erlbaum, 1987) 90-99.

[Lin 1973] Lin S.and Kernighan B.W. An Effective Heuristic Algorithm for the TSP, *Operations Research,* 21 (2):498-516.

[Oliver 1987] I. Oliver, D. Smith, and J. R. Holland. A study of permutation crossover operators on the traveling salesman problem. John Grefenstette (Ed.), *Genetic Algorithms and their Applications: Proc. of the Second International Conf.* (L. Erlbaum, 1987) 224-230.

[Suh 1987] Jung Suh and Dirk Van Gucht. Incorporating heuristic information into genetic search, in: John Grefenstette (Ed.), *Genetic Algorithms and their Applications: Proc. of the Second International Conf.* (L. Erlbaum, 1987) 100-108.

[Whitley 1988] Whitley D., and Kauth J. GENITOR: a different genetic algorithm, in *Proceeding of the Rocky Mountain Conference on Artificial Intelligence,* Denver, CO (1988) 118-130.

[Whitley 1989a] Whitley D., and Hanson, T. Optimizing Neural Networks Using Faster, More Accurate Genetic Search. *Proceeding of the Third International Conference on Genetic Algorithms, 1989.* Morgan Kaufmann, Publishers.

[Whitley 1989b] Whitley D. The G*ENITOR* algorithm and selective pressure: why rank-based allocation of reproductive trials is best. *Proceeding of the Third International Conference on Genetic Algorithms, 1989.* Morgan Kaufmann, Publishers.

```
builddistance (x, y, d)
 int x[LENGTH], y[LENGTH];
 int d[LENGTH][LENGTH];
 {
 int first, second;
 for(first=0; first<LENGTH; first++)
    {d[first][first] = 0.0;
    for (second=0; second<first; second++)
       {d[first][second] =
         (int)((sqrt((double)(DIFFSQ(x[first],x[second])
           +(DIFFSQ(y[first],y[second]))))) + 0.5);
        d[second][first] = d[first][second];
       }}
    }
```

Table 2: The builddistance function builds a table of distances from the (x,y) coordinates. The code uses integer coordinates to compute the edges as integer values.