Tunneling Between Optima: Partition Crossover for the Traveling Salesman Problem

Darrell Whitley, Doug Hains and Adele Howe
Computer Science
Colorado State University
Fort Collins, CO 80523
whitley, dhains, howe, @cs.colostate.edu

ABSTRACT

A new recombination operator is introduced for the Traveling Salesman Problem called *partition crossover*. Theoretical and empirical results indicate that when two local optima are recombined using partition crossover, two offspring are produced that are highly likely to also be local optima. Thus, the operator is capable of jumping or *tunneling* from two local optima to two new and distinct local optima without searching intermediate solutions. The operator is *respectful* and it *transmits alleles* which means that 1) all common edges from the two parents are inherited and 2) the offspring are constructed using only edges inherited from the two parents. Partition crossover is not always feasible: sometimes two new Hamiltonian circuits cannot be constructed by the operator using only edges inherited from the two parents. But empirical results indicate that partition crossover is feasible 95 percent of the time when recombining randomly selected local optima. Furthermore, from a sample of local optima that are within a short random walk of the global optimum, partition crossover typically relocates the global optimum in a single move when crossover is feasible.

Category and Subject Descriptors: I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, Search

General Terms: Theory, Algorithms

Keywords: fitness landscapes, Traveling Salesman Problem, recombination

1. INTRODUCTION

For more than 20 years, evolutionary algorithm researchers have attempted to construct good recombination operators for the Traveling Salesman Problem (TSP). According to Johnson and McKeogh [6], the use of genetic algorithms for the TSP influenced the development of the Iterated Lin-Kernighan algorithm. Yet, recombination operators for the TSP are not well understood and many are not particularly good at transferring edges found in parent solutions to offspring [14] [4] [8].

A new recombination operator called *partition crossover* (PX) is presented in this paper. A proof is presented showing that partition crossover displays what Radcliffe and Surry [13] call "respectful recombination that transmits alleles." The alleles in this case are edges found in the parent solutions. A *respectful* operator ensures all common edges shared by the two parents are inherited; an operator *transmits alleles* if the offspring are constructed using only edges inherited from the two parents.

Partition crossover is not guaranteed to be feasible; the method that it uses to detect new Hamiltonian circuits can fail. Detecting Hamiltonian circuits in graphs is NP-Hard. However, empirically the operator is feasible over 95 percent of the time when recombining randomly chosen local optima.

Our analysis suggests that when *partition crossover* is used to recombine two solutions that are locally optimal, the offspring are also likely to be locally optimal. This result is general and holds for different local search operators. The analysis takes the form of a constraint argument. Empirically, we find that the recombination of locally optimal solutions typically produced locally optimal offspring distinct from the two parents. Empirical data were generated using three benchmarks from TSPLIB: ATT532, d1291 and pr2392. We recombined 10 locally optimal solutions for three different TSP instances. There are 45 $(10 \cdot 9/2)$ pairings of the 10 local optima. For each test problem, the 45 possible recombinations were feasible for at least 43 of the pairings; almost all (more than 95 percent) of the resulting offspring were new local optima.

One can view *partition crossover* as a macro-operator linking local optima. The ability to jump or "tunnel" from local optimum to local optimum makes search using *partition crossover* quite different than other methods for exploring local optima for the Traveling Salesman Problem.

The current paper focuses on the landscape that is induced by partition crossover and its ability to "tunnel" to new optima. The paper applies the operator to randomly selected local optima, as well as local optima that are close to the global optimum. The local optima that are close to the global optimum are found by doing a random walk away from the global optimum until a new basin of attraction is found. Empirically, when partition crossover is used to recombine local optima that are close to the global optimum, the offspring that are produced include the global optimum with high probability.

1.1 About Notation: Red and Blue Tours

Most of this paper looks at the recombination of two TSP tours that correspond to Hamiltonian circuits. A useful device is to refer to these as the red tour and the blue tour. When viewed in color, all figures use red and blue to distinguish the tours; in black and white, the red tour is shown as a solid line and the blue tour as a dashed line.

2. BACKGROUND ON TSP OPERATORS

Early recombination operators for the TSP include 1) cycle crossover [10], 2) Grefensette's greedy crossover [5], and 3) Goldberg and Lingle's [4] partially-mapped crossover (PMX). None of these were particularly effective, partly because the transfer of edges (alleles) from parents to offspring was relatively poor.

Two of the more effective early crossover operators for the Traveling Salesman Problem are 1) edge recombination [14] and 2) Muehlenbein's Maximally Preservative Crossover (MPX) [8]. Both operators are designed to explicitly transfer edges from parents to offspring. But neither operator could construct offspring using *only* edges from the parents. Despite the name of MPX, empirical studies show that edge recombination does a better job at preserving (i.e., transmitting) edges from parents [7]. And when used alone, edge recombination is the more effective operator. However, when used in combination with 2-opt, MPX is somewhat more effective. This is consistent with the somewhat unintuitive observation made by Johnson and McKeogh [6]: applying 2-opt to candidate solutions which have been improved by other methods can result in poorer overall results than applying 2-opt to less improved candidate solutions.

Another notable recombination operator for the TSP is Edge Assembly [9]. Edge Assembly exploits what its creators call **AB-cycles**. Assume we have two Hamiltonian circuits that are candidate solutions to a TSP problem. Label one of the Hamiltonian circuits **A** and the other **B**. The 1993 Edge Assembly paper contains the following definition and Theorem for the symmetric TSP. The definition and theorem have been reworded for clarity.

Definition: AB-cycle

Starting at any initial vertex, a cycle defined as an **AB-cycle** is generated by alternately selecting edges of tour-A and tour-B until the initial vertex is reached.

Theorem:

Let G be a graph created by merging the vertices and edges from two Hamiltonian circuits labeled **A** *and* **B**. *There exists one (or more) decomposition(s) of edges into* **AB-cycles** *such that every edge in G is a member of just one* **AB-cycle**. *Furthermore, there exists no way of decomposing G into* **AB-cycles** *such that every edge does not belong to some* **AB-cycle**.

It follows that there exists a simple greedy algorithm for decomposing G into **AB-cycles**. Different decompositions can be obtained by randomly starting at a different initial point; in addition, when there are two **A** edges or two **B** edges to select from, just chose one randomly. Using this approach every graph which is the union of two Hamiltonian circuits can be fully decomposed into **AB-cycles**. Half of the edges in the **AB-cycle** decomposition come from parent **A** and half come from parent **B**.

Noted that there are degenerate **AB-cycles**. This occurs when there exists an edge $e(v_1, v_2)$ from tour **A** and an edge $e(v_2, v_1)$ from tour **B**: the cycle travels from v_1 to v_2 and back to v_1 over a single (common) edge in the two tours.

Edge Assembly uses **AB-cycles** to cut the parent tours into subtours, which are then reassembled in a greedy fashion to create offspring. Thus Edge Assembly does not transmit alleles. The greedy "assembly" process introduces new edges in the offspring that are not directly inherited from either parent.

2.1 Theoretical Properties of Recombination

Several concepts from Radcliffe's work [12] formalizing properties of recombination operators are useful when evaluating recombination operators. The following definitions are from Radcliffe and Surry ([13]; pg 55):

> A recombination operator is said to *respect* a representation if and only if every child it produces contains all the alleles common to its two parents.

While respect means that alleles common to both parents must be inherited, it does not require that all alleles in the offspring must be inherited from one or the other parent. This is captured in another concept: transmission.

> A recombination operator is said to *transmit alleles* with respect to a given (formal) allelic representation if and only if each allele in every child it produces is present in at least one of its parents.

Radcliffe uses the concept of an *allelic representation* to characterize representations for the TSP. An allelic representation does not have genes, only alleles. If we decompose the TSP into edges and designate these to be the alleles, then there are no genes, only alleles. An operator that transmits alleles is *not* necessarily respectful, since transmitting alleles from the parents does not ensure that all shared alleles are transferred to offspring.

Note that any operator for the TSP which transmits alleles and is respectful will minimize disruption during recombination. The fitness of offspring will be more correlated with the fitness of parents, since they are composed of the same edges and have the same common edges.

3. A NEW RECOMBINATION OPERATOR

It is sometimes possible for a recombination operator for the Traveling Salesman Problem to both *transmit alleles* and exhibit *respect*. In graph theoretic terms, this implies that we construct a graph G from the two parent tours and the graph G has more than two Hamiltonian circuits. At least one Hamiltonian circuit is distinct from the parents and also has the same shared common edges as the parents. The trick, of course, is to know when these Hamiltonian circuits are present and then how to locate them. Finding a Hamiltonian circuit in an arbitrary graph is NP-Hard [2].

3.1 The Reduced Graph G*

To aid in locating Hamiltonian circuits, we will use a *reduced graph*.

Let G be a graph produced by unioning two Hamiltonian circuits. We construct a reduced graph G* which is a minor

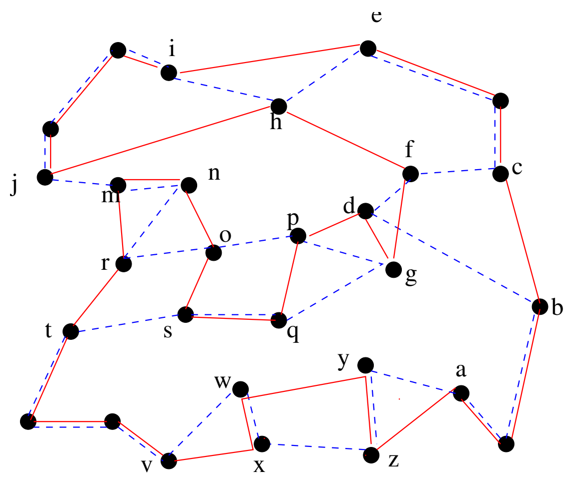

Figure 1: A random Traveling Salesman Problem and two Hamiltonian circuits. Cities that are part of common subtours are not labeled and will be removed.

graph of G. We can classify edges in G as either *common edges* which appear in both parents, or as *non-common edges* which appear in only one parent. A *common subtour* is a contiguous set of one or more common edges. If there exists a non-common edge in G, this edge and the incident vertices are copied to G*. If there is a common subtour in G composed of exactly one edge, this is also copied to G*. We will define a *surrogate edge* as a single edge in G* that replaces a common subtour (of length > 1 edge) in G. If there is a common subtour with more than one edge in G, this is contracted and replaced by a surrogate edge in G*. This is represented in G* as a single edge between the start and end vertices of the common subtour.

Another simple way to construct G* is as follows. Since G is made up of two tours, and each vertex in a single tour has only two edges touching a vertex, all of the vertices in G have degree two, three or four. Vertices with degree two are part of a common subtour of length greater than one. All vertices in G with degree two are deleted to form G*, and the edges of each corresponding common subtour are replaced by a single surrogate edge.

Not all of the Hamiltonian circuits in G* are also Hamiltonian circuits in the original graph G. Only those circuits that traverse the surrogate edges in G* are circuits in G.

A 29 city TSP is shown in Figure 1 along with two Hamiltonian circuits which are represented by the red (solid) and blue (dashed) tours. One can think of the red and blue tours are parent solutions.

Let $e(a,z)$ denote an edge between vertices a and z. In Figure 1, the edges $e(m,n), e(s,q), e(w,x)$ and $e(y,z)$ are single common edges but do not correspond to surrogate edges. Common subtours of length greater than one include the subtours that go from i to j; informally this will be denoted by $i{\equiv}j$. Other common subtours include $c{\equiv}e, a{\equiv}b$, and $t{\equiv}v$. The reduced graph G* can be created by deleting those vertices that are unlabeled in Figure 1 and by adding surrogate edges between $i{\equiv}j, c{\equiv}e, a{\equiv}b$, and $t{\equiv}v$.

Note that if we cut the graph between $t{\equiv}v$ and between $a{\equiv}b$, it cleanly breaks the graph into two independent pieces. Thus, we can follow the red (solid) edges from vertex b to t, and then follow the blue (dashed) edges from v to a to create a new Hamiltonian circuit (below labeled as **child1**). And we can follow the blue (dashed) edges from b to t and then follow the red (solid) edges from v to a to create a second circuit (labeled **child2**). This results in the following crossover between $t{\equiv}v$ (and $a{\equiv}b$).

red:	a≡b c≡e i≡j h f g d p q s o n m r t≡ v x w y z
blue:	a≡b d f c≡e h i≡j m n r o p g q s t≡v w x z y
child1:	a≡b c≡e i≡j h f g d p q s o n m r t≡v w x z y
child2:	a≡b d f c≡e h i≡j m n r o p g q s t≡v x w y z

Sections 3.2 and 3.3 introduce an algorithm for *partition crossover*. The algorithm uses the concept of a reduced graph and graph partitioning to isolate and exploit common subtours.

There can potentially be many Hamiltonian circuits in G and G* in addition to those found by partition crossover. For example, there are at least 4 ways to get from b to t that preserve common edges in addition to the path used by the original parent tours in Figure 1:

b c≡e h i≡j m n r o p d f g q s t
b d p o s q g f c≡e h i≡j m n r t
b d g f c≡e h i≡j m n r o p q s t
b d p q g f c≡e h i≡j m n r o s t

And there are two ways to get from v to a:

```
v w x z y a
v x w y z a
```

This means that there are at least 8 additional Hamiltonian Circuits in Figure 1 not found by partition crossover. We note as a topic of future research that partitioning decomposes the search for Hamiltonian circuits into smaller independent subproblems.

3.2 Partition Crossover

A **graph partition** of cost 2 exists in graph G* if the vertices in G* can be partitioned into two non-empty sets and there are only two edges connecting the two sets of vertices.

Theorem 1:
For any reduced graph G produced from a graph G which is the union of two distinct Hamiltonian circuits, if there exists a graph partition of G* with cost 2, the partition also exists in G and a recombination is possible that generates two offspring that are distinct Hamiltonian circuits: the resulting recombination is both respectful and transmits alleles.*

Proof: Assume there is a graph partition of the reduced graph G* of cost 2. The two edges that connect the two partitions must be common edges. Common edges are counted as a single edge in G*. The division must cut each tour exactly twice. If the edges connecting the partitions were not common edges, the minimum cost of the graph partition would be greater than two.

We will again use the concept of red and blue tours. The path followed by the red tour and the blue tour must be different in each partition. If the tour were the same in either partition, then this would represent a common subtour and the edges within the partition would be absorbed into a surrogate edge.

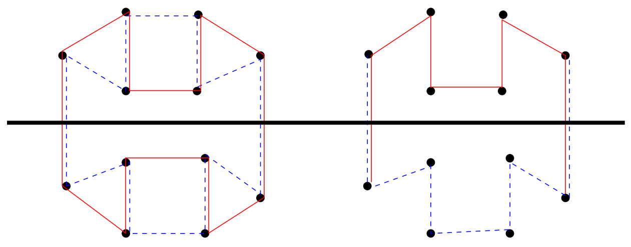

Figure 2: An illustration of Partition Crossover. The partitions are above and below the dark solid line. Only one offspring is shown on the right.

Both the red tour and the blue tour must visit all the cities in a partition before leaving that partition. Therefore, recombination can visit the cities in one partition by following the route used by the red tour, and then follow the route used by the blue tour in the other partition to produce a new Hamiltonian circuit. One can reverse this mapping to produce a second offspring.

The resulting recombination *transmits alleles* since it only uses edges in G* and G. The resulting recombination is *respectful* since it traverses the two common edges *between* partitions, and if there are common edges *inside* the partitions, they must are traversed by both the red and blue tours. Therefore all common edges are traversed. □

This theorem proves we can construct a crossover operator for the TSP by checking if there exists a partition of cost 2 in a graph G*. If the partition does not exist, the recombination operator fails. An example of partition crossover is given in Figure 2.

Corollary 1:
Let $P1$ and $P2$ be two parent tours and $C1$ and $C2$ be the two offspring. Let f be the evaluation function. Partition crossover guarantees that

$$f(P1) + f(P2) = f(C1) + f(C2)$$

This follows directly from the constructive proof of Theorem 1. Assume we union the two parents to produce a graph G; by construction if we union the two offspring we obtain the same graph G.

3.2.1 Some Observations about Partition Crossover

When partition crossover fails, this does not mean there is not a third Hamiltonian circuit, only that there is not a Hamiltonian circuit that can be found by this method.

Empirically, partition crossover is very successfully at generating offspring. But if partition crossover only finds one particular type of Hamiltonian circuit, why is it successful at finding new Hamiltonian circuits?

One answer may be that we have focused on recombining local optima, and local optima tend to display common subtours. Two solutions that share many common subtours are more likely to result in graphs that can be partitioned.

Because partition crossover is invertible, one can recombine the two parents to obtain the two children, or the roles can be reversed and one can recombine the two "children" to obtain the "parents."

Observation: *Given two randomly chosen parents, the probability of an offspring yielding an improving move under partition crossover is 50 percent.* This is proven by the following argument. Given two parents and two offspring, one of the four solutions must be best. But since partition crossover in invertible, the role of parents and offspring can be reversed. Given that the parent were chosen with a uniform random probability, the two offspring could have been chosen as the "parents" with the same probability. Thus, in expectation when recombining *randomly* chosen parents, half of all recombinations will yield an improving move.

This observation generalizes to any invertible, respectful crossover operator that transmits alleles.

We have observed both empirically and analytically that if the parents are locally optimal, then the offspring are also likely to be locally optimal. The reasons for this will be explored in the next section. From the observation in the previous paragraph, if the probability of selecting all local optima as parents is equal, one might expect that the recombination of two randomly chosen local optima (when the offspring are also local optima) will discover a new improved local optima with probability 1/2. However, in this case it is an open question as to whether the probability of selecting arbitrary local optima as parents is uniformly likely.

3.2.2 Why Partition Crossover Generates Local Optima

Empirically, when partition crossover is applied to parents that are local optima, the offspring are also usually local optima. This was predicted based on the following observations. If the two offspring are local optima, they must be two new local optima distinct from either parent. This follows from the proof of Theorem 1.

Why are the offspring likely to be local optima? It is useful to think of the partition as a barrier. (See Figure 2 again.) The parent solutions have their vertices distributed on either side of this barrier in the same fashion as do the offspring. Assume the offspring are not locally optimal and that local search is applied to the offspring (the choice of operator is not important). The sequence of vertices on each side of this barrier is already locally optimal in both the parents and the offspring; therefore no application of local search (using the same operator) involving vertices on just one side of this barrier can improve these segments. Improvement is only possible if vertices (and edges) move across this barrier.

If we do find an offspring that is not a local optimum, it is even less likely that the offspring would lie in the basin of attraction of one of the parents. For a solution to end up in the basin of attraction of one of the parents, local search must somehow move vertices (and edges) across the barrier, and then later move the same vertices (and edges) back across the barrier. Assume an offspring does lie in the same basin as a parent. If the union this offspring (before applying additional local search) and the parent, then they have exactly the same vertices on each side of the partition barrier. If an offspring is not a local optimum then the application of local search must break this barrier; if the barrier is broken, then the new improved offspring cannot be in the same basin of attraction as the parents. For the offspring to be in the same basin as a parent, local search must first break the barrier, and then somehow restore the barrier. The "movement" of vertices (and edges) *across* the barrier *and back again* must be done in such a way that local search is able to resequence the vertices and also restore the distribution of vertices on either side of the barrier in order to be in the basin of attraction of a parent. This would appear to be highly unlikely.

In practice we expect partition crossover to *usually* produce offspring that are local optima; and then we expect the offspring to almost always reside in basins that are distinct from the parents.

3.2.3 *The Problem with Transmitting Alleles*

A major problem with all operators that transmit alleles is that they never generate new alleles. This means that one cannot simply drop an operator that transmits allele into a genetic algorithm, turn it on, and expect good solutions. The solutions can be no better than the edges in the initial population.

When 2-opt is applied along with genetic recombination, it not only improves the solutions, it also introduces new edges. But if the offspring are local optima, 2-opt cannot be used to introduce new edges. Thus, while it is exciting that partition crossover can discover new local optima, in some sense it would be better if the offspring were not local optima, since this would create an opportunity to introduce new edges and gain further improvements.

Because the offspring are usually local optima, it appears likely that partition crossover must be used in some form of hybridized algorithm in order to be effective.

3.3 The Complexity of Partition Crossover

The graph partitioning problem is NP-complete. However, graph partitioning can be polynomial for specialized problems [3]. Partition crossover can be implemented in $O(N)$ time. In this section, we briefly sketch the basic details of an $O(N)$ implementation.

We construct an edge table as originally used for edge recombination [14]. If a vertex b is adjacent to vertices a and c in the red tour, and vertex b is adjacent to a and k in the blue tour, this results in an edge table entry `(b: a' c k)`. Entry `a'` denotes that the edge $e(b, a)$ is a common edge. Construction of the edge table requires $O(N)$ time. The degree of a vertex can also be obtained from the edge table.

Partitions can be found by "deleting" all common edges. Any partitions in G (or G*) would now have cost 0. Since partition crossover only cuts common edges, partitions are now disconnected and any connected components must be in the same partition. We don't literally need to construct G*. When scanning the edge table, the search follows those edges which are not flagged as common edges to find each set of connected components.

To locate all partitions in G in $O(N)$ we construct two additional arrays. LIST1 is an unsorted list of all vertices. LIST1 is divided at location i such that all cities at locations less than or equal to i have not been assigned to a partition, and all cities at locations greater than i have been assigned to a partition. LIST2 indexes into LIST1 so that the location of a particular vertex in LIST1 is found in constant time.

LIST1 can be initialized in $O(N)$ time so that all cities of degree 2 are at the front of LIST1. Initialize i to point to the last element of LIST1. Initialize LIST2 so that LIST2(a) returns the location of vertex a in LIST1.

Start at the first vertex, c, in LIST1 with degree 3 or 4, assign it to partition P_1 by labeling it in the edge table. Swap the locations of vertex c and vertex LIST1(i). Decrement i and update the corresponding pointers in LIST2. Place the vertices in the edge table that are connected to vertex

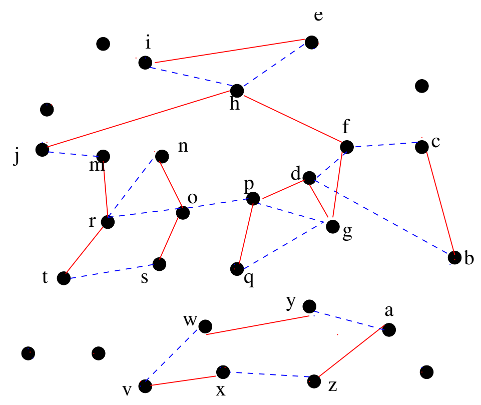

Figure 3: The graph from Figure 1 with common edges deleted. The two partitions are now disconnected.

c in a FIFO queue to be processed (ignoring common edges and those vertices which have already been assigned to a partition). (With additional bookkeeping, the FIFO can be implemented as part of LIST1.)

As each vertex c is processed, it swaps position with the vertex at LIST1(i), pointer i is decremented and LIST2, the FIFO, and the edge table are updated. This procedure is repeated for all the connected components in partition P_1.

When there are no more connected components in partition P_1, check for unprocessed cities in LIST1 with degree 3 or 4. If there are additional cities of degree 3 or 4 in LIST1 with index $\leq i$, then partition crossover is feasible.

Every subset of connected components is processed in the same fashion and assigned to a new partition.

An example of the graph from Figure 1 with the common edges removed is given in Figure 3. Note that there are isolated vertices; these are vertices of degree 2 in G. Using the concept of a surrogate edge, if both ends of a surrogate edge connects to the same partition, all of the vertices in the surrogate edge belong to that partition. If the two ends of a surrogate edge connect to two different partitions, the graph can be partitioned by cutting that surrogate edge.

Each vertex in LIST1 is processed one time using a constant amount of work. Thus the complexity of partition crossover is $O(N)$.

There can be more than 2 partitions. In general, if there are z partitions, it is possible to generate $2^z - 2$ legal offspring using partition crossover; however, the best offspring can be found by selecting the shorter segment associated with each partition. In the current set of experiments when the algorithm found 1 partition, it halted and did a single crossover. In future work we need to exploit all partitions.

3.4 Tunneling Between Optima: Empirical Results

Two empirical tests illustrate both the utility and the limitations of partition crossover. The experiments were initially done using the benchmark ATT532 which is a 532 city tour and is relatively difficult given its size [11]. The global optimum has an evaluation of 27686. We also did the same tests for two TSPLIB benchmarks: d1291 and pr2392 which are printed circuit board problems. We present the ATT532 results in some detail, and summarize the results for d1291 and pr2392.

3.4.1 Experiment One: Random Local Optima

In section 3.2 we made two observations about partition crossover. 1) The operator will fail if a partition does not exist, and 2) we expect offspring to usually be local optima. The first experiment assessed how often partition crossover is feasible on randomly selected local optima, and how often the offspring are local optima.

We generated 10 unique random local optima by applying 2-opt to 10 random tours. For ATT532 the evaluation for the sampled local optima ranged from 29493 to 30090; the evaluations can be found in Table 1 (Parent 1 or Parent 2). We attempted to recombine all of the $(10 \cdot 9)/2 = 45$ pairs of local optima. Partitions were found in 43 of the 45 cases (95%), resulting in the location of 86 new solutions (two offspring per pair).

All of the 86 new solutions were local optima under 2-opt. 47 of the 86 local optima had distinct evaluations. None of the 86 local optima has the same evaluation as the 10 parents. Some solutions were duplicate local optima, and some solutions were different local optima with the same evaluation.

Seven of the 86 local optima were better than 29493, the best local optima of the initial 10. Five of the seven solutions that were better than 29493 were offspring of parent 29493. The seven improved local optima have four distinct evaluations: 29371, 29402, 29486 (4 times) and 29491.

We could have limited the number of recombinations and only applied partition crossover using the two best of the initial local optima paired against all of the others (and each other). This would have found all 7 of the improving moves. To prevent presenting a very large table, these are the only results shown in Table 1. While Table 1 only shows the best of the two offspring, the evaluation of the second offspring can be directly computed based on the evaluation of P1, P2 and the one offspring using Corollary 1.

The results for TSP d1291 and pr2392 are similar. When partition crossover is applied to all pairwise combinations of 10 random local optima, crossover is again feasible in at least 43 out of 45 cases (43 for d1291 and 44 for pr2392). For both TSP instances, the 9 best offspring were produced by recombining the best of the local optima with the other 9 local optima; there were 5 improving moves for d1291 and 7 improving moves for pr2392 out of the 9 recombinations. In 2 cases for d1291 and 3 cases for pr2392 the offspring were not local optima. If would be interesting to find more cases where the offspring are not local optima and to study how much improvement can be gained in these cases.

What do these results tell us? First, it illustrates that the space of local optima appears to be densely linked under this operator: partition crossover was able to recombine two local optima and find two new local optima in at least 43 of 45 cases for all 3 TSP instances: this is a 95 percent success rate for recombining randomly selected local optima. Second, greedy is good. The improving moves were found by recombining the best local optima with the other local optima. This can greatly reduce search costs.

About half of the recombinations find an offspring better than the parents. On the ATT532 city problem, 21 of the 43 recombinations found an offspring better than the parents. On the d1291 city problem, 22 of the 43 recombinations found one offspring better than the parents. On the pr2392 city problem, 25 of the 44 recombinations found one offspring better than the parents.

Parent 1	Parent 2	Child
29493	29579	29371*
29493	22934	29486*
29493	29611	29486*
29493	29495	29486*
29493	30090	29486*
29493	29953	29510
29493	29950	29520
29493	29825	29606
29493	29813	29606
29495	29611	29402*
29495	29934	29491*
29495	29813	29502
29495	29825	29502
29495	29953	29502
29495	29579	29502
29495	30090	29503
29495	29950	29522

Table 1: The results of applying partition crossover pairing the two best local optima against eight other local optima. The numbers given here are the evaluations of the tours. The * symbol indicates an improved solution relative to the initial sample of local optima. Only the best offspring is shown. The evaluation of the second offspring can be calculated.

3.4.2 Experiment Two: Optima near the Global

Given the problem of only transmitting alleles, what if partition crossover is only used after the search gets close to the global optimum? To test this idea, we generated a set of solutions that are close to the global optimum.

Starting at the global optimum, we did a random walk of k moves and then reapplied steepest descent 2-opt local search to locate a local optimum that is near the global optimum. Figure 4 shows each random walk as a connected string of triangles. The long solid lines connect the last step of the random walk with the local optima found after re-optimizing with 2-opt.

We will refer to the number of moves required to converge to a new local optimum as the *escape distance*. Under steepest-descent, escape distances of 15 to 30 moves are common. (Much to our surprise, we also found a second solution with an evaluation of 27686 for the ATT532 city problem.)

The random walk procedure was repeated 10 times to locate 10 local optima that are "near" the global optimum. We then attempted to recombine all of the $(10 \cdot 9)/2 = 45$ pairs of local optima. For the ATT532 city problem, these 10 local optima range in evaluation from 27687 to 28597. Partition crossover was feasible in 34 of the 45 cases. *In all 34 cases, partition crossover rediscovered the global optimum.*

The results for ATT532 are summarized in Table 2. In this case, the local optima are labeled from A to J, and only those pairings that could not be recombined by partition crossover are itemized. For example, solution A paired with the other nine local optima rediscovered the global optimum 8 out of 9 times, but could not be recombined with solution D by partition crossover. Solution B could be recombined with all other solutions. The worst solution, J, with an evaluation of 28597 could be recombined with 8 of 9 solutions to rediscover the global optimum. While Table 2 does not

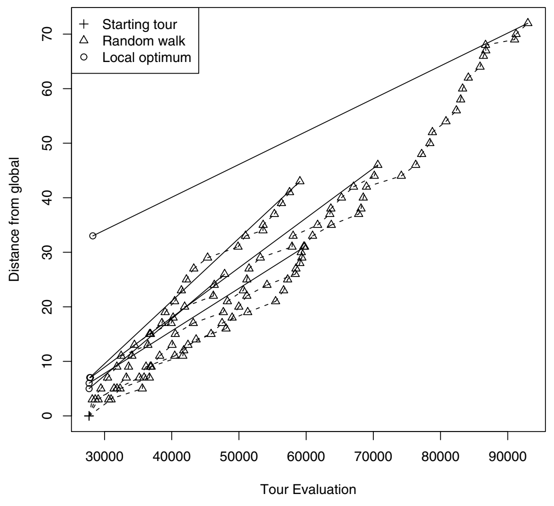

Figure 4: Four random walks from the global optimum. The walks are extended until steepest descent 2-opt is able to locate a new local optimum.

explicitly list the offspring, one of the offspring is always the global optimum with an evaluation of 27686 and thus the evaluation of all of the offspring can be calculated.

The results for d1291 and pr2291 were similar. For both problems 10 local optima were also generated near the global optimum by doing a random walk away from the global optimum until a new local optimum is located using 2-opt. There are 45 possible pairings. For d1291, partition crossover was feasible in 21 cases and in 20 of these cases the global optimum was rediscovered. For pr2291, partition crossover was feasible in 27 cases and in 20 of the 27 cases, the global optimum was rediscovered.

3.4.3 *Implications and the Big Valley Hypothesis*

We have shown empirically that partition crossover jumps between local optima, making it possible to directly move from one local optimum to the next. One interesting question is how does this relate to the "Big Valley Hypothesis" [1] for the TSP?

The Big Valley hypothesis is based on the observation that local optima that are similar in evaluation are also similar in terms of their distance from the global optimum. To illustrate this, we generated a sample of local optima. To generate this sample, and to ensure that we obtained local optima near to the global, we did random walks of length k (that is, k random 2-opts) away from the global optimum, then reapplied next-descent 2-opt to find a local optimum. By using increasing values for k, local optima could be generated at increasing distance from the global optimum. The sample of local optima is shown in Figure 5. On the x-axis is the evaluation of the local optima. On the y-axis is the distance from the global optimum as measured in terms of the number of edges that differ from those in the global optimum. The result is what is described as a "Big Valley."

To better understand how 2-opt moves are related to the Big Valley structure, we examined four of the random walks used to escape the global optimum. Referring again to Figure 4, three of the four local optima are close to the global in that they share all but 5–7 of their edges with the global optimum. One of the four local optima has more than 30 edges not found in the global optimum. The long walks and the high evaluations along the random walks indicate that barriers between local optima that are near the global optimum can be significant.

Crossover of 10 parents close to the global		
ID	Eval	Results
A	27687	Found Global on 8/9, failed on D
B	27709	Found Global on 9/9
C	27720	Found Global on 7/9, failed on D, E
D	27761	Found Global on 5/9, failed on A, C, E, G
E	27768	Found Global on 6/9, failed on C, D, G
F	27818	Found Global on 7/9, failed on I, J
G	27823	Found Global on 5/9, failed on D, E, H, I
H	27881	Found Global on 7/9, failed on G, I
I	28014	Found Global on 6/9, failed on F, G, H
J	28597	Found Global on 8/9, failed on F

Table 2: The results of applying partition crossover pairing 10 local optima that are near the global optimum. Partition crossover was feasible in 34 or 45 pairings, and all 34 feasible recombinations generated the global optimum. The 10 tours are labeled from A to J. "Failed" indicates which cities could not be recombined. For example, C could not be feasibly recombine with cities D or E. "Found" indicates how many recombinations were feasible *and* found the global optimum.

Figure 6 shows four examples of partition crossover applied to local optima that are relatively near to the global optimum. The four examples (8 parents and 8 offspring) were selected to not overlap so that they are easy to see. The figure illustrates how a sample of local optima that share 90 percent or more edges with the global optimum behaves under partition crossover. The two parents closest to the global optimum generate the global optimum as an offspring.

It should be noted that local optima do not have to be extremely close to the global optimum in terms of evaluation for partition crossover to be effective. In Experiment Two in section 3.4.2, the global optima is "linked" by partition crossover with some solutions ranging in evaluation from 28000 to almost 29000.

4. CONCLUSIONS

Partition crossover provides a new mechanism that allows search to tunnel between local optima with no additional evaluation of points in the search space. As our experiments show, when a set of local optima that are "near" the global optimum are recombined using partition crossover, one can jump directly to the global optimum. Our experiments also show that it is relatively easy to find many local optima that are "linked" to the global optimum.

In some sense, the fact that the offspring of partition crossover inherit all of their edges from their parents is a disadvantage. Given a set of local optima, partition crossover cannot generate new edges; all new local optima that are generated by partition crossover will be composed of edges

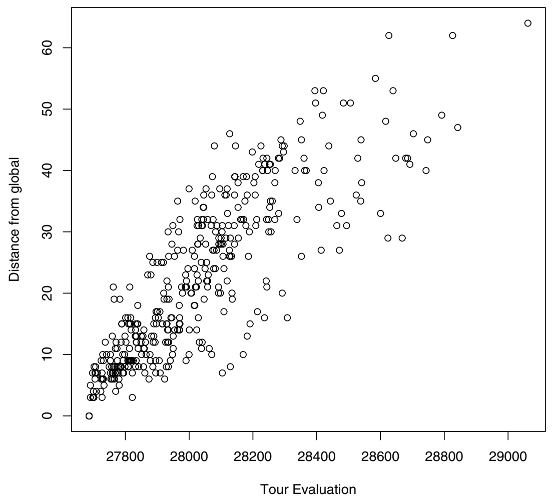

Figure 5: A sample of local optima displaying a "Big Valley" distribution. Local Optima that have more edges in common with the global optimum are closer in evaluation to the global optimum.

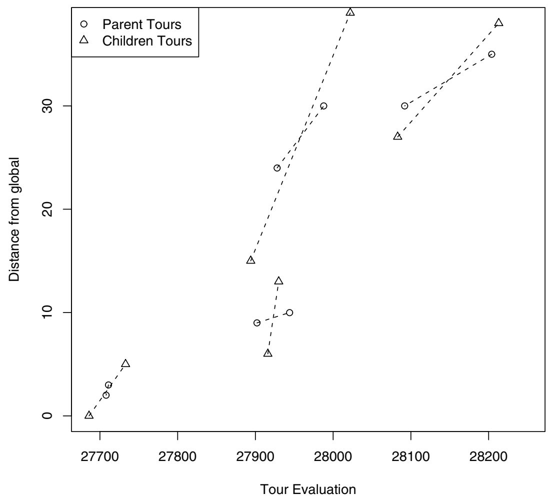

Figure 6: Four examples of partition crossover showing how the parents and offspring are distributed relative to their evaluations and the number of edges shared with the global optimum.

from the parents. Thus partition crossover must depend on some other algorithm to find good local optima, or to otherwise introduce new good edges into the search.

Nevertheless the performance of partition crossover makes it clear that good solutions can be decomposed into segments that can be rearranged to find improved solutions.

Acknowledgments

This effort was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-08-1-0422. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

5. REFERENCES

[1] K. Boese. Cost versus distance in the traveling salesman problem. Technical report, Computer Science Department, UCLA 1995.

[2] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. McGraw Hill, New York, 1990.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[4] David Goldberg and Jr. Robert Lingle. Alleles, Loci, and the Traveling Salesman Problem. In *International Conf. on GAs*, pages 154–159, 1985.

[5] J. Grefensette. Incorporating Problem Specific Knowledge in Genetic Algorithms. In L. Davis, ed, *Genetic Algorithms and Simulated Annealing*, pages 42–60. Morgan Kaufmann, 1987.

[6] D. S. Johnson and L. A. McGeoch. The Traveling Salesman Problem: A Case Study in Local Optimization. In E. H. L. Aarts and J.K. Lenstra, eds, *Local Search in Combinatorial Optimization*, pages 215–310. Wiley, 1997.

[7] Keith E. Mathias and L. Darrell Whitley. Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem. In *Parallel Problem Solving from Nature, 2*, pages 219–228. Elsevier, 1992.

[8] H. Mühlenbein. Evolution in Time and Space: The Parallel Genetic Algorithm. In G. Rawlins, ed, *FOGA -1*, pages 316–337. Morgan Kaufmann, 1991.

[9] Yuichi Nagata and Shigenobu Kobayashi. Edge Assembly Crossover: A High-Power Genetic Algorithm for the Traveling Salesman Problem. In T. Bäck, editor, *7th International Conf. on GAs*, pages 450–457. Morgan Kaufmann, 1997.

[10] I. Oliver, D. Smith, and J. Holland. A Study of Permutation Crossover Operators on the Traveling Salesman Problem. In *GAs and Their Applications: ICGA 2* Erlbaum, 1987.

[11] W. Padberg and G. Rinaldi. Optimization of a 532 City Symmetric TSP. *Optimization Research Letters*, 6(1):1–7, 1987.

[12] Nicholas J. Radcliffe. The Algebra of Genetic Algorithms. *Annals of Maths and Artificial Intelligence*, 10:339–384, 1994.

[13] N.J. Radcliffe and P.D. Surry. Fitness Variance of Formae and Performance Predictions. In D. Whitley and M. Vose, eds, *FOGA - 3*, pages 51–72. Morgan Kaufmann, 1995.

[14] L. Darrell Whitley, Timothy Starkweather, and Daniel Shaner. The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination. In L. Davis, ed, *Handbook of Genetic Algorithms*, pages 350–372. Van Nostrand Reinhold, NY, 1991.