Generalized Asymmetric Partition Crossover (GAPX) for the Asymmetric TSP

Renato Tinós
Department of Computing and Mathematics
University of São Paulo
Ribeirão Preto, SP, Brazil
rtinos@ffclrp.usp.br

Darrell Whitley
Department of Computer Science
Colorado State University
Fort Collins, CO, USA
whitley@cs.colostate.edu

Gabriela Ochoa
Department of Computing Science and Mathematics
University of Stirling
Stirling, FK9 4LA, Scotland
gabriela.ochoa@cs.stir.ac.uk

ABSTRACT

The Generalized Partition Crossover (GPX) constructs new solutions for the Traveling Salesman Problem (TSP) by finding recombining partitions with one entry and one exit in the graph composed by the union of two parent solutions. If there are k recombining partitions in the union graph, $2^k - 2$ solutions are simultaneously exploited by GPX. Generalized Asymmetric Partition Crossover (GAPX) is introduced; it finds more recombining partitions and can also find partitions for the asymmetric TSP. GAPX does this by locating partitions that cut vertices of degree 4 in the union graph and by finding partitions with multiple entry and exit points, both in $O(n)$ time. GAPX can improve the quality of solutions generated by the Lin-Kernighan-Helsgaun heuristic and improve the state of the art for the asymmetric TSP.

Categories and Subject Descriptors

1.2.8 [**Artificial Intelligence**]: [Problem Solving, Control Methods, and Search]

Keywords

Asymmetric Traveling Salesman Problem, Recombination Operator, Evolutionary Combinatorial Optimization

1. INTRODUCTION

An instance of the Traveling Salesman Problem (TSP) can be defined by a complete weighted graph $G(V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n vertices (or cities) and E contains edges between every pair of vertices in V. Each edge $e_{i,j} \in E$ between vertices $v_i, v_j \in V$ is associated with a weight $w_{i,j} \in \mathbb{R}^+$, generally indicating the distance or travel cost between the two cities. All candidate solutions for a TSP are Hamiltonian circuits in G, and thus each solution $x \in X$ is typically represented as permutations over the set of vertices in graph G. Considering the vertex v_{x_1} as the fixed start and end point, then the evaluation of a particular solution $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}} \in X$, specifying a permutation on $n-1$ vertices, is given by:

$$f(\mathbf{x}) = w_{x_n,x_1} + \sum_{i=1}^{n-1} w_{x_i,x_{i+1}} \quad (1)$$

The objective of the optimization process is to find $\mathbf{x} \in X$ with the minimum evaluation $f(\mathbf{x})$. The TSP can be symmetric or asymmetric. For example, the *distance* between two cities is typically a symmetric cost; however, the travel time from city A to city B might be different from the travel time from city B to city A depending on traffic patterns or time of day.

An amazing number of optimization algorithms have been proposed for the TSP. One of the best heuristics is the Lin-Kernighan-Helsgaun (LKH) algorithm [5]. The core of LKH is a variable depth local search (LK heuristic). In LKH, a number of improvements have been added to the original LK heuristic, like the use of sensitive analysis to estimate the probability of an edge appearing in an optimal solution. LKH is also a form of iterated local search which uses soft restarts. In addition, LKH includes a recombination operator, called Iterative Partial Transcription (IPT) [7], which is similar in effect to the Generalized Partition Crossover (GPX) operator independently developed by Whitley et al. [13] [12]. Both GPX and IPT are 1) respectful and 2) transmits genes. Respectful operators transfer all common features (edges) found in both parents to the offspring. Operators that "transmit genes" generate offspring that are composed only of features (edges) contained in the parent solutions. GPX has $O(n)$ complexity while IPT as described has $O(n^2)$ complexity.

Hains et al. [4] improved multi-trail LKH by using GPX. However, due to the similarity of Helsgaun's IPT recombination operator and GPX, the improvement was not a result of using GPX, rather the improvement was a result of using respectful recombination more frequently. Hains et al. recombined each new locally optimal solution with *all* of the previously discovered local optima. LKH with IPT only recombined each new locally optimal solution with the current best so far solution. By accumulating *all* of the local optima encountered, Hains et al. incrementally constructed a population on the fly which could be exploited by recombination.

In this paper, we add three enhancements to GPX. First, GPX is adapted for use with the asymmetric TSP (ATSP). The new operator is called *GAPX: Generalized Asymmetric Partition Crossover*. This means that GAPX considers the direction in which the tour (solution) is moving along each

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2576768.2598245.

edge during recombination. GAPX can still be applied to the symmetric TSP. Second, GAPX exploits cuts that break nodes of degree 4 of the union graph between two parent solutions as a site for crossover. IPT also exploits "cuts" at vertices of degree 4 during recombination. While IPT appears to have $O(n^2)$ complexity in the worst case, Helsgaun's implementation of IPT is often efficient because recombination can only occur at common edges, or at vertices of degree four. The third enhancement allows GAPX to identify partitions with multiple entries and exits, i.e., subgraphs with more that two points connecting the partition to the rest of the union graph. Both the original GPX operator and the IPT operator exploit only subgraphs with 1 entry and 1 exit. These three enhancements do not change the complexity of the original GPX; the cost of recombination is $O(n)$, where n is the number of cities in the TSP. GAPX can find more feasible recombinations than GPX or IPT, and thus, it can exploit a larger number of possible offspring.

In Section 3, GAPX is evaluated in two ways: first, as a recombination operator inside a Genetic Algorithm (GA) using local search based on 3-opt moves; second, as a recombination operator used to improve solutions generated by multi-trial LKH. The experimental results indicate that the GAPX can be employed to produce very competitive algorithms for the ATSP. This paper is concluded in Section 4 with a discussion of future work.

2. GAPX

Whitley et al. [13] presented the GPX operator for the symmetric TSP. Both GPX and GAPX recombine solutions by exchanging subpaths that contain edges not shared by the parents. The subpaths are identified by removing the common edges of the union graph $G_u = G_1 \cup G_2$, where $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are the parent solutions, and connected components are found using Breadth First Search in the remaining graph. The solutions generated by partition crossover (PX, GPX or GAPX) are always Hamiltonian circuits [12]. If there are k connected components representing feasible partitions, then $2^k - 2$ offspring different from the parents can be generated by crossover [13]. As the evaluation of a Hamiltonian cycle length is the sum of the subpaths inside and outside the recombining partitions in the TSP, the best of the $2^k - 2$ offspring solutions is found by selecting the best partial solution for each subgraph partition. In practice, the GPX operator has been observed to exploit as much as 27 partitions, and the set of reachable offspring contained more than $2^{26} = 64$ million local optima. GPX, IPT, and GAPX all return the best of the 2^k offspring; GAPX is guaranteed to yield a larger value of k than GPX or IPT.

Previous work has also shown that if the parent solutions are local optima, the majority of the offspring are also local optima [12]. This is due to the fact that no feasible partition can be further improved by local search; if this is true for the parents, it must also be true for the offspring. The only local search move than can improve a solution after GAPX (or GPX) is one that moves multiple cities distributed across multiple partitions, and this type of improving move is rare.

Figure 1 illustrates two ideas: it illustrates how recombination exploits partitions of the graph G_u and it also illustrates how a graph can be divided at a vertex of degree 4. Let the red (dashed) lines represent parent one and blue (solid) lines represent parent two. First consider the middle graph. A partition exists which cuts only 2 common edges. This creates a single entry and exit from each subgraph. Given one entry and exit, a tour can be broken at these "cuts" and recombination will always yield new distinct and feasible offspring [12]. If there are multiple partitions that cut 2 common edges, this process can be repeated (i.e., iterated). The leftmost graph illustrates another idea: nodes of degree 4 can be "split". If vertex 6 in Figure 1 is broken into vertices 6 and 6', with a zero cost on the new edge, the graph can again be partitioned by cutting common edges. We will denote 6' as a *ghost* vertex. Ghost vertices are removed after recombination.

One possible challenge with creating *ghost* vertices is that every vertex of degree 4 can be split in two ways, because the ghost vertex might be inserted before or after the original vertex. Thus, if there are $O(n)$ vertices of degree 4, there are potentially $O(2^n)$ different graphs that might be generated by splitting vertices of degree 4. In practice, this problem is resolved by recognizing that the desired graph is obtained by doing the insertion consistently. All vertices of degree 4 are split by consistently inserting the ghost vertex immediately after the original vertex. In the ATSP this is sufficient to correctly split all of the vertices of degree 4 because all solutions are directional. In the symmetric TSP, each solution can be thought of as being bi-directional since the tour can be navigated in a clockwise or counter-clockwise fashion. Luckily, it does not matter which direction is chosen as long as the insertion of ghost vertices is done consistently after (or before) the original vertex.

We will refer to the direction of movement from one vertex to the next as the "flow" of the Hamiltonian circuit. The direction of "flow" is particularly important for the ATSP. This process is captured in the following procedure.

Procedure 1: *Create a ghost vertex in G_1 and G_2 for each vertex with degree 4 in G_u. Determine the direction of flow. Insert the ghost vertex immediately after the original vertex of degree 4 in both G_1 and G_2. The common edge between the original vertex and the ghost vertex is assigned weight 0.*

After *ghost* vertices are inserted, we ask which common edges partition the graph. If a partition yields a single entry and exit point, then recombination can occur and will yield a new Hamiltonian circuit. The properties of GPX are explained in more detail by Whitley et al. [12, 13]. Using GAPX to split vertices of degree 4 is illustrated in the following example (see Figure 1). The * symbol illustrates the (only) location where respectful recombination is feasible.

```
  Parent 1:   11 * 1 2 3 4 5 6 * 6' 7 8 9 10
  Parent 2:   11 * 1 3 2 5 4 6 * 6' 8 7 10 9
Offspring 1:  11 * 1 2 3 4 5 6 * 6' 8 7 10 9
Offspring 2:  11 * 1 3 2 5 4 6 * 6' 7 8 9 10
```

GAPX is illustrated on a more complex example in Figure 2. In this case, there are 3 partitions and thus there are $2^k - 2 = 8 - 2 = 6$ possible offspring. It is not necessary to generate all offspring; the best possible offspring is obtained by selecting the best subpaths inside each subgraph [12].

Splitting a vertex of degree 4 does not necessarily divide the graph into connected subgraphs. This can be seen by considering the vertices 10, 10', 13, 13', 15 and 15' in Figure 2. These nodes are also split but do not yield common edges that can be used to divide graph G_u.

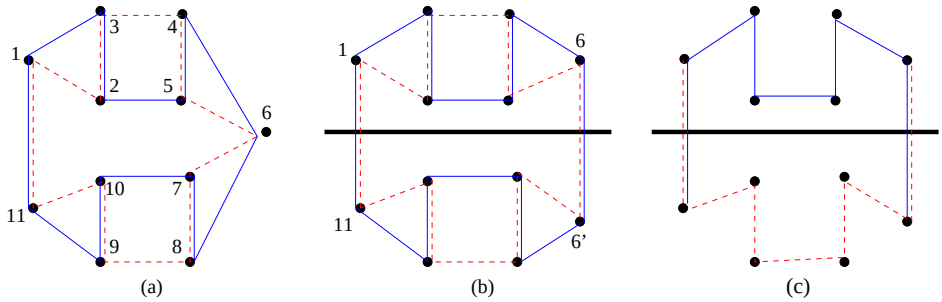

Figure 1: Recombination by breaking vertices of degree 4. a) The blue (solid) and red (dashed) solutions can be recombined by breaking the graph into two subgraphs, where each subgraph has only one entry and exit for both tours. b) Vertices of degree 4 (vertex 6 in this case) are split into two vertices (6 and 6'), with a dummy (zero cost) edge between them. The graph is then cut or "partitioned" using common edges. c) One of the offspring generated by recombination.

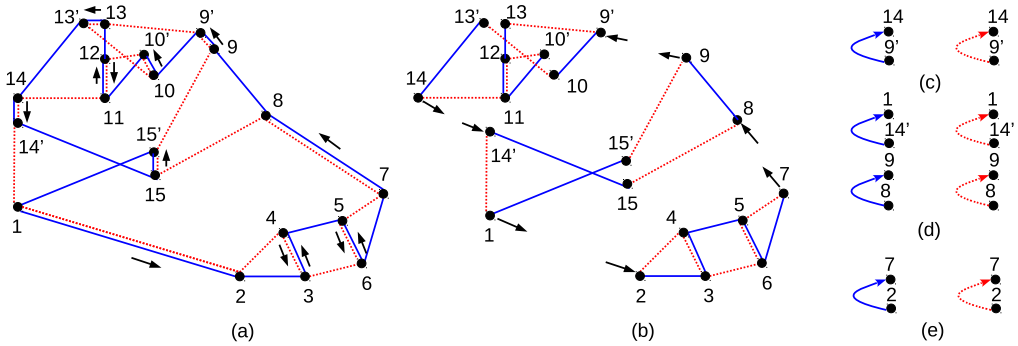

Figure 2: GAPX applied to an example with 15 vertices. a) Insertion of the ghost nodes after vertices with degree 4 with nodes 9', 10', 13', 14' and 15' being ghost nodes; b) After removing the common edges three connected components are identified; c), d), e) Because the simplified graphs are equal for both solutions, the three candidate partitions are feasible partitions for recombination.

2.1 GAPX for the ATSP

For the ATSP, GAPX considers an edge as a common edge only if the respective edges in both parents have the same flow direction, e.g., the union of edge (1,2) and (1,2) is a common edge, while the union of edge (1,2) and (2,1) is not a common edge. GAPX also recognizes more complex partitions that can have multiple entries and exits.

GAPX generates offspring in the following way. 1) Generate graph G_u by merging the parents, G_1 and G_2. 2) Split vertices of degree 4 by inserting ghost vertices after the original vertex; denote the new graph as G'_u. 3) Delete the common edges from G'_u and identify the connected subgraphs; copy the common edges to the offspring. 4) Determine if the connected subgraphs are *feasible partitions*. 5) Copy the edges from the shortest subpath for each *feasible partition*.

A *feasible partition* is one where both tours enter at the same entry points, both tours visit all of the vertices of the partition, and both tours exit at the same exit points. Both IPT and the original GPX use only partitions with a single entry and exit, with one exception: the "residual graph" can have multiple entry and exit points. For example, in Figure 2 there are two partitions with a single entry and exit. After the feasible partitions are processed, the *remainder* of the graph has multiple entries and exits. After feasible partitions are identified, the remainder of the graph is also considered a feasible partition for recombination.

GAPX can be applied to the symmetric TSP by arbitrarily selecting the direction of flow. Thus, the label of entry or exit is arbitrary (i.e. exchangable) but fixed.

2.2 Multiple Entries and Exits Partitions

The second major contribution of this paper is a procedure to identify feasible partitions with multiple entries and exists. After identifying the connected components in the graph obtained from removing the common edges of G'_u, the obtained partitions must be tested. Denote these as *candidate partitions*. The candidate partitions are connected components of G'_u after all of the common edges are deleted. All of the vertices in a candidate partition must be reachable by traversing single edges (not common edges). If a vertex has only one edge that connects to another vertex in the connected components, then it is a potential entry or exit.

If a partition subgraph is not a feasible partition for recombination, it cannot be used to generate a Hamiltonian cycle. To test if a candidate partition is a feasible partition, the following test is applied.

Procedure 2:

i) *For each candidate partition, create a simplified directed graph G_{in} for each Hamiltonian cycle (parent)*

containing only the entry and exit vertices found in each candidate partition. Replace the path inside the candidate partition between each entry and respective exit vertices by only one edge;

ii *Verify if the simplified graphs inside the candidate partition are equal for both parents. If the simplified graphs are equal, the candidate partition is a feasible partition under recombination.*

The identification of the entry and exit vertices, as well the edges connecting them, can be done by visiting each vertex of G'_u following the paths given by G_1 and G_2. Therefore, all simplified graphs G_{in} can be constructed in $O(n)$ time.

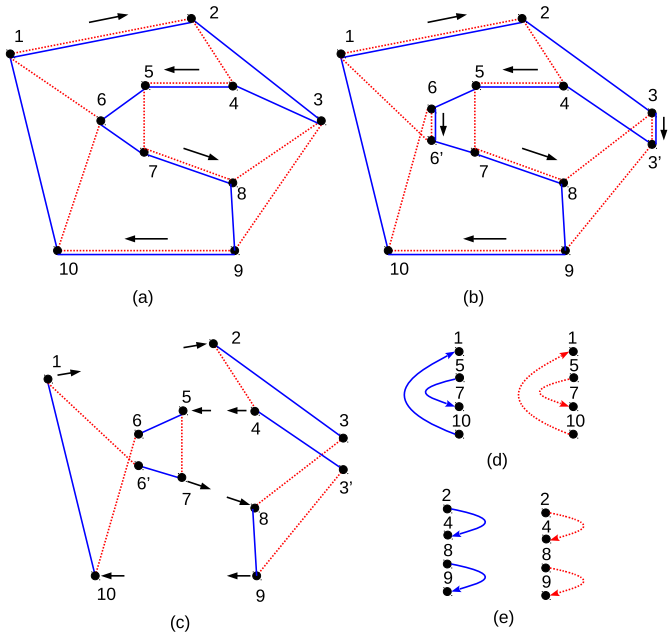

Figure 3: GAPX applied to an example with partitions with multiple entries and exits. a) Assume we know the direction of "flow"; b) Ghost vertices are inserted after vertices with degree 4 (Procedure 1); c) After removing the common edges, the two connected components are identified; d), e) Each candidate partition generates two simplified graphs, one for each solution. Because the simplified graphs are equal, the two candidate partitions are feasible partitions for recombination (Procedure 2). If the paths inside one partition are exchanged, the offspring must be Hamiltonian cycles.

Examples of the application of procedures 1 and 2 are given in figures 2 and 3. In Figure 2 GAPX identifies the 2 feasible cuts yielding 3 feasible partitions. In Figure 3, there are two entry points and two exit points and two feasible partitions. Note that vertices 1, 5, 7 and 10 are all connected vertices reachable by single edges; similarly, 2, 4, 8 and 9 are all connected vertices. Thus GAPX identifies a feasible cut that cross 4 instead of 2 common edges. The implementation of GAPX is described in Algorithm 1.

When compared to the original GPX, the following differences can be observed: i) Step 1 is not present; ii) The common edges are independent of the flow direction (Step 3); iii) The identification of recombining partitions (Step 5) is made by testing if the candidate partition has one entry and one exit.

Algorithm 1 GAPX

Step 1. Insert a ghost vertex immediately after each vertex with degree 4 in both Hamiltonian cycles. The common edge between the original vertex and the ghost vertex has weight 0 (Procedure 1). The graphs G_1 and G_2 (parents) with the added ghost vertices are respectively denoted G'_1 and G'_2.
Step 2. Create a union graph $G'_u = G'_1 \cup G'_2$.
Step 3. Remove all common edges (with same flow direction) of $G'_u = G'_1 \cup G'_2$.
Step 4. Use Breadth First Search to find the connected components of G_u to identify the candidate partitions for recombination.
Step 5. Determine if the candidate recombining partitions are feasible partitions for recombination. To test each candidate partition, create 2 simplified graphs, one for each parent. The simplified graphs should contain as vertices only the entry and exit vertices of the candidate partition. Inside the partition, replace the path between each entry and respective exit vertex by a single edge. Verify if the two simplified graphs are equal (Procedure 2). If the graphs are equal, the candidate partition is a feasible partition for recombination.
Step 6. Apply crossover by finding the shortest path inside each feasible partition.

2.3 Analysis

In this section, we present a theorem stating that recombination using GAPX results in offspring that are Hamiltonian circuits. Recall that GAPX cuts only common edges with same flow direction for both parents. For the symmetric TSP, the flow directions can be the same, or exactly the opposite. The following lemmas are about the identification of the cutting points for the candidate partitions.

LEMMA 2.1. *The direction of flow across a common (cut) edge separating feasible partitions is the same for both Hamiltonian cycles (parents) for the ATSP. In the case of the symmetric TSP, the direction of flow must be* consistent *across cuts.*

PROOF. Edges must be copied to the offspring in the same order that they appear in the parents. A cutting point that is an entry for the path given by one parent and an exit for the path given by the other parent would cause the inversion of the paths in the offspring for an ATSP. For the symmetric TSP, if all of the "cut" common edges consistently have reversed flow, the flow direction of one of the parents (which parent is arbitrary) can be reversed. □

LEMMA 2.2. *A entry or exit of a recombining partition in $G'_u = G'_1 \cup G'_2$, where G'_1 and G'_2 are created from graphs G_1 and G_2 using Procedure 1, can only be a common edge of G'_1 and G'_2.*

PROOF. Entry and exit points must connect all of the feasible partitions. The procedure that creates ghost vertices removed all vertices with degree 4 in the union graph. Thus, all cut points in the graph must cut both Hamiltonian cycles in the same place, and this only happens when common edges are cut. □

THEOREM 2.3. *The candidate partitions identified using Procedure 2 are feasible partitions for recombination. The result of recombination using these feasible partitions yield Hamiltonian circuits.*

PROOF. For the previous lemmas, one can observe that the entry and exit vertices of a candidate partition are entries or exits for both Hamiltonian cycles (parents). However, the order in which the entry/exit pairs are visited can be different for the paths given by each parent.

Let $G_{in}^{p_1} = (V_s, E_{in}^{p_1})$ and $G_{in}^{p_2} = (V_s, E_{in}^{p_2})$ denote the simplified graphs for the parents with edges representing the paths **inside** the partition, where V_s contains the entry/exit vertices of the partition and $E_{in}^{p_i}$ are edges representing the paths inside the partition for parent p_i. Let $G_{out}^{p_1} = (V_s, E_{out}^{p_1})$ and $G_{out}^{p_2} = (V_s, E_{out}^{p_2})$ denote the simplified graphs for the parents with the edges representing paths **outside** the partition, where $E_{out}^{p_i}$ are edges representing the paths outside the partition for parent p_i. Then the combined simplified graphs in the two offspring o_i are $G_s^{o_1} = (V_s, E_{in}^{p_1} \cup E_{out}^{p_2})$ and $G_s^{o_2} = (V_s, E_{in}^{p_2} \cup E_{out}^{p_1})$.

If $E_{in}^{p_1} = E_{in}^{p_2}$, then the combined graphs of the offspring are equal to the simplified combined graphs for the parents, i.e., $G_s^{p_1} = G_{in}^{p_1} \cup G_{out}^{p_1}$ and $G_s^{p_2} = G_{in}^{p_2} \cup G_{out}^{p_2}$. Since the simplified combined graphs for both parents are Hamiltonian cycles, the offspring generated in this way are also Hamiltonian cycles. One can observe that the same procedure cannot be adopted for testing if $G_{out}^{p_1}$ is equal to $G_{out}^{p_2}$ because these simplified graphs can be changed by other recombining partitions. □

THEOREM 2.4. *The time complexity of the GAPX operator is $O(n)$.*

PROOF. Procedure 1 generates vectors with size equal to the number of vertices of degree 4 (n_{v4}) plus the number of cities (n). As $n_{v4} < n$, the procedure is $O(n)$. The same complexity $O(n)$ is observed in steps 2, 3, 4, and in the procedure used to identify the entries and exits in the partitions. When Procedure 2 is applied to test the i-th candidate partition, two simplified graphs with size $n_{io}(i)$ are built, where $n_{io}(i)$ is the number of entries and exits in the i-th candidate partition. As the sum of $n_{io}(i)$ for all candidate partitions is equal or less than the number of nodes in the union graph, then the procedure is $O(n)$. □

2.3.1 Observations and Opportunities

Using GAPX, all feasible partitions for recombination in graph G_u must cut an even number of common edges. However, one can prove that not all partitions that cut an even number of common edges yields a feasible partition for recombination. In some cases there are connected subgraphs that do not yield feasible partitions and cannot be combined to yield feasible partitions. In other cases, two non-feasible partitions can be unioned to create a feasible partition; we ignore this in the current implementation because in the worse case these unions appears to require $O(n^2)$ time. This points to two lines of future research: 1) we have still not exploited all of the ways in which respectful and transmitting recombination can be used to find improved solutions and 2) we continue to explore how to identify more of these partitions in $O(n)$ time.

3. EXPERIMENTS

The GAPX can be applied as a recombination operator in different algorithms employed to generate improved solutions for the ATSP. This section demonstrates the ability of the GAPX to find superior solutions in two different approaches.

3.1 Genetic Algorithm with GAPX

We apply the GAPX as a recombining operator inside a Genetic Algorithm (GA) that uses local search based on the 3-opt moves (ls3opt).

In Algorithm 2, ls3opt is applied to random solutions in order to generate the initial population. Elitism is used to preserve the current best solution, while tournament selection (where two individuals are randomly chosen and the best is selected with probability 0.8) is used to select parents for crossover. If crossover did not improve the best solution, mutation is applied. In the mutation operator, the solution is transformed, with same probability, by: a 2-opt move, a double bridge move and ls3opt. While 3-opt and double bridge moves keep the order between cities, 2-opt moves allow the inversion of subpaths between two cities in the solution. In order to reduce the computational time of the local search procedure, the ls3opt employed here uses a limited number of neighbor vertices in the 3-opt moves, and "don't look bits" and "first found solution" strategies (we use the same procedure employed in [8]). In the GA, if the best solution is not improved in the last 20 generations, the neighborhood size employed by ls3opt is increased (starting with 10 cities until a limit of 150 cities or n if it is smaller than 150). Then, all solutions, with exception of the best solution, are replaced by random solutions optimized by ls3opt (immigration). The parameters employed here are: population size equal to 300 individuals, number of runs equal to 25, and execution until the optimal solution is found or until a maximum number of 1500 generations.

The GA with GAPX is here compared to 2 other approaches. All algorithms are exactly the same, except that they use different crossover operators. This means that differences in the results are due to the effectiveness of recombination. The only difference between GPX and GAPX is that GAPX finds more feasible partitions for recombination. The three crossover operators were the following:

i) Same-site-copy-first crossover (SSCFX) [11]. SSCFX was used in the evolutionary algorithm applied to the ATSP by [14]. This operator has similarities to several operators from the 1990s that emphasized the inheritance of location and relative position.

ii) GPX adapted to the ATSP. Only partitions with 2 cutting points are used, and vertices with degree 4 are not considered as possible cutting points. To be applied to the ATSP, the direction of flow of the common edges is considered when determining if a connected subgraph is a feasible partition for recombination.

iii) GAPX. The operator splits nodes of degree 4 and finds partitions with multiple entries and exits.

Table 1 presents the obtained results for the asymmetric instances of the TSPLIB [9] (the instances generated from ftv170 were not employed here). Note that all instances of the ATSP in the TSPLIB contain less than 500 cities. In order to allow the comparison to the results obtained by other algorithms (e.g., the results compiled in [8]), Table 1 presents the follow measures: *success*, that indicates the percentage of runs where the global optimum was found; *a-err*, that is the average percentage excess with respect to the global optimum evaluation; *a-T*, that is the average computation time for each run in seconds. The experiments were performed on a Core 2 Quad 2.83 GHz processor.

Algorithm 2 GA with GAPX

```
P=popInit();
while termination condition is not satisfied do
  Q(1)=bestSolution(P);
  for i=2 to maxpop do
    (p1,p2)=selection(P);
    Q(i)=crossover(p1,p2);
    if crossover did not improve the solutions then
      Q(i)=mutation(Q(i));
    end if
  end for
  if best solution did not improve in last 20 gen. then
    increase the neighborhood size used in ls3opt;
    Q=immigration();
  end if
  P=Q;
end while
```

Of the three operators, the GA with GAPX shows the best performance; it is the only operator that resulted in an optimal solution in every case. SSCFX is a disruptive operator that will often introduce new edges during recombination and there is little to ensure these edges are likely to be found in competitive solutions. While it was the poorest of the 3 operators, it did surprisingly well, solving 10 of the 24 test problems with 100 percent success. When we analyze the final evaluation produced by the GA using SSCFX, we found that the GA with SSCFX was sometimes trapped in local optima from which it was not able to escape.

Table 1 shows that GAPX produced better results than GPX adapted to the ATSP. We know the operators are doing basically the same thing; the only difference is that the new operator is able to find more feasible partitions for recombination. This means that GAPX is breaking the parent solutions into more and smaller subgraphs; both operators make greedy choices when selecting the best solutions from each subgraph. In the worst case, i.e., when GAPX cannot find any additional feasible partition for crossover, both operators generate the same offspring. In the best case, the new operator finds additional offspring that are *guaranteed* to be as good or better than the offspring produced by GPX; GAPX can generate all of the offspring found by GPX, plus more offspring not found by GPX.

When compared to the results of state-of-art evolutionary computation approaches compiled in [8], the GA with GAPX presents better performance for *a-err* (and *success*) than the GA in [2], the Memetic Algorithm (MA) in [1], and the MA in [14]. The GA with GAPX displays similar performance to the GA using the edge assembly crossover (EAX) of Nagata [8]. The GA with EAX had 100 percent success on 23 of 24 instances, while the GA with GAPX displayed 100 percent success on all instances. However, the runtime costs (labeled *a-T* in our Table 1) reported by Nagata et al. [8] were significantly lower.

The GA with EAX uses the concept of AB-cycles to copy the edges of the parents to the offspring and, in order to produce Hamiltonian cycles, introduces new edges not present in the parents. The results presented in [8] are generated by the consecutive application of the EAX crossover operator from a initial population of 300 individuals generated by ls3opt. It would appear that the lower runtime cost is due to the fact that ls3opt (which is costly) is not used by the GA with EAX after the population is initialized.

EAX and GAPX should be viewed as being complementary rather than competitive. EAX is more explorative, and generally exploration is costly. An application of the EAX operator is more expensive that applying GAPX. However EAX is less expensive that the ls3opt local search operations; the results of Nagata suggest that EAX could be used instead of ls3opt to provide the exploration that is needed. GAPX is more exploitative than EAX, because GAPX aggressively finds the best local optima reachable from the available set of solutions that are undergoing recombination. But GAPX can only utilize the edges that are available in the set of solutions that are undergoing recombination, it does not create or destroy new edges.

In the next section, we illustrate other ways in which GAPX can be used as a low cost exploitive operator.

3.2 Lin-Kernighan-Helsgaun (LKH) and LKH using GAPX

While not initially developed for the ATSP, the LKH algorithm is capable of yielding very impressive results for the ATSP. This is accomplished by transforming the ATSP into a symmetric TSP by doubling the size of the original problem. Both LKH and the *Concorde* Branch and Bound use this transformation. If city A appears in the symmetric TSP, then two cities A_1 and A_2 created from A will appear in the ATSP. A_1 can only be a destination and A_2 can only be a point of origination, and the cost of going from A_1 to A_2 is zero, while the cost of going from A_2 to A_1 is infinite (or prohibitively large).

LHK [5] is able to find the optimal solution for all asymmetric instances of the TSPLIB. As with the GA with EAX, LHK results in an optimal solution with lower frequency than the GA with GAPX (i.e., *a-err* and *success* are better for GAPX compared to LKH), but LKH displays lower runtime costs (as measured by *a-T*). The LKH results also appear to be superior to EAX: EAX solved 121 out of the 126 instances [8] presented in [10] in at least one of 100 runs, while LKH solved all 126 instances [6].

This section uses GAPX to improve solutions generated by the multi-trial LKH using the default parameters [6]. Hains et al. [4] describe two ways to do this for the symmetric TSP: i) generate solutions from LKH across runs representing multiple runs of LKH, or ii) generating solutions from LKH across restarts of the same run.

Applying GAPX "across runs" means that multiple runs of LKH are being used, and as local optima are found by the different runs, GAPX recombines the solutions to look for better solutions. When using GAPX across runs, the new solutions found by GAPX are not re-inserted into the search process. This means that GAPX runs as an isolated side process recombining the solutions found by LKH, but it does not influence LKH in any way.

LKH is a form of iterated local search that uses a "kick" operator to escape local optima. LKH already uses IPT before applying the kick operator. In the version of LKH we have investigated, IPT recombines the current local optimum with the previous best local optimum. Applying GAPX "across restarts" means that when LKH would normally apply IPT, we apply GAPX instead; also, instead of just recombining the current local optimum with the previous best local optimum, we save all of the previously discovered local optima and recombine the best solution so far with *all* of the other local optima.

Table 1: Results for the GA with: same-site-copy-first crossover (SSCFX) [11]; adapted GPX; GAPX.

Problem	SSCFX [11]			adapted GPX			GAPX		
	success	*a-err*	*a-T*	*success*	*a-err*	*a-T*	*success*	*a-err*	*a-T*
br17	100	0.000	0.02	100	0.000	0.03	100	0.000	0.03
ftv33	100	0.000	0.36	100	0.000	0.34	100	0.000	0.21
ftv35	88	0.016	24.55	92	0.011	23.00	100	0.000	0.75
ftv38	52	0.063	57.87	92	0.010	45.05	100	0.000	1.83
p43	100	0.000	16.31	100	0.000	16.05	100	0.000	7.02
ftv44	100	0.000	4.34	100	0.000	1.42	100	0.000	1.08
ftv47	76	0.081	76.32	100	0.000	28.67	100	0.000	2.45
ry48p	52	0.265	114.08	100	0.000	29.79	100	0.000	2.82
ft53	100	0.000	33.15	100	0.000	27.83	100	0.000	3.57
ftv55	100	0.000	21.06	100	0.000	11.70	100	0.000	1.10
ftv64	92	0.065	70.72	100	0.000	57.66	100	0.000	3.19
ft70	64	0.032	242.90	92	0.004	169.61	100	0.000	16.26
ftv70	72	0.068	249.84	64	0.074	313.30	100	0.000	10.98
kro124p	56	0.124	954.02	92	0.002	606.14	100	0.000	34.84
ftv170	44	0.501	4600.13	40	0.672	5270.10	100	0.000	1243.55
rbg323	100	0.000	32.05	100	0.000	39.13	100	0.000	31.68
rbg358	100	0.000	390.60	100	0.000	432.25	100	0.000	301.12
rbg403	100	0.000	74.23	100	0.000	80.05	100	0.000	78.85
rbg443	100	0.000	31.83	100	0.000	26.97	100	0.000	29.97

success: % of successful runs; *a-err*: average % excess; *a-T*: average computation time (in sec.).

In the experiments presented here, we used LKH applying GAPX "across runs." In this case, GAPX does not interfere in the LKH heuristic. Also, GAPX can be applied to the original ATSP without needing to transform it into the (twice as large) symmetric TSP.

We again use the controlled experiments similar to those used by Hains et al. [4]. The experiments were executed 25 times, each time with a different random seed for LKH. For each execution, LKH was run with 10 hard restarts, and each run was allowed 50 soft restarts; thus a total of 500 locally optimal solutions are generated for each execution. Assume the 10 hard restarts are being executed as 10 parallel runs. After the i^{th} soft restart, we obtain 10 solutions from each parallel run. After the first restart, we add the best solution found so far by GAPX to this set. In this way, we do not lose the best solution found by GAPX until the last soft restart. This means there are 11 solutions at each soft restart. The best of the 11 solutions is selected, and it is recombined with the other 10 solutions. For each execution of LKH, this process is repeated 50 times, once for each soft restart. This means that the GAPX operator is applied 500 times for each execution of LKH. Thus, LKH will sample 500 locally optimal solutions, and GAPX will generate an additional 500 locally optimal solutions. By adding GAPX in this way, we can never do worse than multi-trial LKH, but GAPX can potentially find better solutions.

Table 2 shows results over 25 executions for the best solutions in 4 real ATSP instances with more than 800 cities [3] and for instances generated by inserting random Gaussian deviations (with standard deviation equal to $0.2w_{ij}$) to each weight w_{ji} in the symmetric TSPs generated from cities randomly distributed according to uniform distribution (E1k and E3k) or clusters of cities (C1k and C3k) [3]. Also, the statistical comparison using the non-parametric Wilcoxon signed-rank test with significance level 0.02 is presented.

GAPX presented better average results for 15 out of 16 instances, and improved the best solution found by LKH 12 times. In the multi-trial LKH, IPT is employed to recombine the solution found in the current trial to the best solution found in the run and to recombine the final best solutions between runs. Thus even after the application of IPT, GAPX was still able to recombine the solutions and generate improved solutions. It is important to observe that GAPX is dependent on the quality and diversity of the solutions produced in the different runs of LKH. When the diversity of the solutions produced in different runs of LKH is low, there is less diversity to exploit and GAPX generally produces fewer improvement. In the experiments shown in Table 2, the lowest diversity between the evaluations across the runs (in each execution) was produced for the instance td1000. For problems were LKH finds good solutions very fast, GAPX is not capable to improve the results because the diversity across the runs is low after a few trials of LKH.

4. CONCLUSIONS

In this paper, a new version of partition crossover is presented. The major contributions are mechanisms that allow GAPX to: i) consider vertices of degree 4 as potential cutting points for the feasible partitions for recombination; ii) exploiting feasible partitions with any even number of cutting points; iii) performing these operators in $O(n)$ time. The original GPX operator only recombined partitions with one entry and exit point, and only cut common edges. The IPT operator does recombination by exploiting vertices of degree 4, but the description of the operator results in $O(n^2)$ complexity. These properties result in a high probability of generating locally optimal solutions if the parents are local optima, which is very helpful in escaping from local optima. Also, our search strategies apply recombination much more frequently than does the LKH algorithm.

GAPX exploits and explores more solutions than GPX or IPT. Given k feasible partitions, $2^k - 2$ solutions different from the parent solutions are generated by any partition crossover operator. This is true for GPX, IPT and GAPX. Because GAPX finds more partitions, the number of solutions explored by GAPX increases exponentially.

Very competitive results were generated using the GAPX in the experiments performed here. The GAPX can be used in combination with other search methods. This was demonstrated by applying it in two different approaches: i) as a

Table 2: Results for LKH and LKH+GAPX. The last column shows the results of the statistical comparison (+ indicates better performance for LKH+GAPX and = indicates similar performance).

Problem	n	LKH		LKH+GAPX		Statistical Test
		mean+std	best	mean+std	best	
dc849	849	37570.8 ±10.5	37556	37569.7 ±10.7	37552	+
dc895	895	108247.0 ±53.2	108130	108217.2 ±59.1	108050	+
dc932	932	481413.6 ±115.7	481178	481340.9 ±119.0	481123	+
td1000	1001	1242210.6 ±108.2	1242183	1242210.6 ±108.2	1242183	=
C1k0a	1000	10114604.9 ±20532.7	10082973	10113136.2 ±20950.2	10080354	+
C1k1a	1000	10032272.4 ±12865.5	10018188	10031731.7 ±12922.4	10018188	+
C1k2a	1000	9475235.9 ±40639.6	9432171	9472372.2 ±41076.8	9427838	+
C3k0a	3162	17387429.6 ±27973.5	17337642	17386118.9 ±28121.6	17335639	+
C3k1a	3162	16977847.2 ±51494.6	16900660	16976991.4 ±51012.3	16900660	+
C3k2a	3162	17584664.4 ±60515.7	17459990	17584018.6 ±60703.9	17459508	+
E1k0a	1000	22107830.2 ±35056.4	22041858	22102500.2 ±34363.0	22041858	+
E1k1a	1000	21807800.6 ±34677.8	21760078	21803461.9 ±30111.9	21754715	+
E1k2a	1000	22002493.0 ±38596.9	21932612	21990489.1 ±40564.8	21924819	+
E3k0a	3162	39062713.8 ±66376.4	38945473	39030295.6 ±68831.6	38916452	+
E3k1a	3162	38859134.2 ±97705.1	38659539	38832147.1 ±98882.3	38647128	+
E3k2a	3162	38841077.8 ±65879.6	38705784	38803683.3 ±68168.7	38660737	+

recombination operator in a GA, it was able to find the global optimum in all runs for instances in TSPLIB; ii) as an external-improvement operator used to improve locally optimal solutions produced by multi-trial LKH, GAPX was able to improve solutions in 12 out of 16 instances with more than 800 cities.

GAPX can be used to improve the results of any iterative method that generates multiple local optima; it collects these local optima into a population and then recombines the best solution with the other solutions in the population. This can also be done online or offline.

We know that GAPX still does not exploit all possible partitions. We can prove that there exits non-feasible partitions that can be merged or "fused" into a feasible partition for recombination. In this worse case, this "fusion" could have $O(n^2)$ cost, but it is possible this complexity can be reduced by developing more efficient methods to recognize which non-feasible partitions should be fused and which non-feasible partition should not be fused. Finally, GAPX can be used for other types of TSPs, including dynamic TSPs and symmetric TSPs. Applying GAPX to the symmetric TSP is a natural future step of this work.

5. ACKNOWLEDGMENTS

Renato Tinós was a visiting researcher in the Department of Computer Science at CSU. The visit of the researcher was made possible by the support provided by FAPESP (under grant 2012/22200-9), USP, and CSU. This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF (under grant FA9550-11-1-0088). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

6. REFERENCES

[1] L. Buriol, P. M. França, and P. Moscato. A new memetic algorithm for the asymmetric TSP. *Journal of Heuristics*, 10(5):483–506, 2004.

[2] I.-C. Choi, S.-I. Kim, and H.-S. Kim. A genetic algorithm with a mixed region search for the asymmetric TSP. *Comp. & Op. Res.*, 30(5):773–786, 2003.

[3] J. Cirasella, D. S. Johnson, L. A. McGeoch, and W. Zhang. The asymmetric TSP: algorithms, instance generators, and tests. In *Alg. Eng. and Experimentation*, pages 32–59. Springer, 2001.

[4] D. Hains, D. Whitley, and A. Howe. Improving Lin-Kernighan-Helsgaun with crossover on clustered instances of the TSP. In *Proc. of PPSN XII*, pages 388–397. Springer, 2012.

[5] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Oper. Res.*, 126(1):106–130, 2000.

[6] K. Helsgaun. LKH results for Soler's ATSP instances. http://www.akira.ruc.dk/ keld/research/LKH/, April 2012.

[7] A. Möbius, B. Freisleben, P. Merz, and M. Schreiber. Combinatorial optimization by iterative partial transcription. *Phys. Rev. E*, 59(4):4667, 1999.

[8] Y. Nagata and D. Soler. A new genetic algorithm for the asymmetric TSP. *Expert Syst. with Applications*, 39(10):8947–8953, 2012.

[9] G. Reinelt. TSPLIB 95. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/, 1995.

[10] D. Soler, E. Martínez, and J. Micó. A transformation for the mixed general routing problem with turn penalties. *Journal of the Oper. Res. Soc.*, 59(4):540–547, 2007.

[11] H.-F. Wang and K.-Y. Wu. Hybrid genetic algorithm for optimization problems with permutation property. *Comp. & Op. Res.*, 31(14):2453–2471, 2004.

[12] D. Whitley, D. Hains, and A. Howe. Tunneling between optima: partition crossover for the TSP. In *Proc. of GECCO'09*, pages 915–922, 2009.

[13] D. Whitley, D. Hains, and A. Howe. A hybrid genetic algorithm for the TSP using generalized partition crossover. In *Proc. of PPSN XI*. Springer, 2010.

[14] L.-N. Xing, Y.-W. Chen, K.-W. Yang, F. Hou, X.-S. Shen, and H.-P. Cai. A hybrid approach combining an improved genetic algorithm and optimization strategies for the asymmetric TSP. *Eng. Applications of Art. Int.*, 21(8):1370–1380, 2008.