This is is the author's final version of an article accepted for publication in *Evolutionary Computation* published by MIT Press. https://doi.org/10.1162/evco_a_00254

A New Generalized Partition Crossover for the Traveling Salesman Problem: Tunneling Between Local Optima

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

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

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

Abstract

Generalized Partition Crossover (GPX) is a deterministic recombination operator developed for the Traveling Salesman Problem. Partition crossover operators return the best of 2^k reachable offspring, where k is the number of recombining components. This paper introduces a new GPX2 operator, which finds more recombining components than GPX or Iterative Partial Transcription (IPT). We also show that GPX2 has O(n) runtime complexity, while also introducing new enhancements to reduce the execution time of GPX2. Finally, we experimentally demonstrate the efficiency of GPX2 when it is used to improve solutions found by multi-trial Lin-Kernighan-Helsgaum (LKH) algorithm. Significant improvements in performance are documented on large ($n > 5000$) and very large ($n = 100,000$) instances of the Traveling Salesman Problem.

Keywords

Traveling salesman problem, recombination operator, evolutionary combinatorial optimization

1 Introduction

Deterministic recombination operators have been developed for the Traveling Salesman Problem that have the ability to tunnel directly between local optima. These operators include forms of partition crossover (Whitley et al., 2009, 2010; Tinós et al., 2014), as well as *Iterative Partial Transcription* (IPT) (Möbius et al., 1999). Assuming there are two parents and these parents are locally optimal with respect to some local search operator, the offspring are guaranteed to be *piecewise locally optimal* under the same local search operator used to improve the parent solutions. In many cases, the offspring are also true local optima in the full search space (Veerapen et al., 2016). This means that recombination is often able to move directly from parents that are locally optimal to an offspring that is also locally optimal. This paper introduces a new form of partition crossover called GPX2, which is able to find more recombination opportunities compared to IPT and other previously defined partition crossover operators. This implementation of GPX2 and the associated proofs and data structures also improve on

all previously published work on GPX, including an early version of GPX2 (Sanches et al., 2017).

The *Traveling Salesman Problem* (TSP) can be defined by a complete weighted graph $G(V, E)$, where every vertex in $V = \{v_1, v_2, \ldots, v_n\}$ is linked to all other vertices. Each edge $e_{i,j} \in E$ between vertices $v_i, v_j \in V$ is associated with a weight $w_{i,j} \in \mathbb{R}^+$. The feasible solutions of the TSP are Hamiltonian cycles in G. The evaluation of a particular solution $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}} \in X$, specifying a Hamiltonian cycle in G, is given by:

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

where node v_{x_1} is considered the start point and the final point. The objective is to find $\mathbf{x} \in X$ with the minimum evaluation $f(\mathbf{x})$. The weight matrix $W = [w_{ij}]$ can be symmetric or asymmetric. In the former case, the problem is denoted as a *symmetric TSP*, while it is denoted as a *asymmetric TSP* in the latter case.

There are excellent exact methods capable of solving instances of the TSP with hundreds of cities in seconds (Cook, 2011). Some TSP instances are easier to solve than might be expected (Hoos and Stützle, 2014); however, in general the average computation time of exact methods for the TSP increases exponentially with n. Heuristics that produce very impressive results for TSP instances with many thousands of cities have been proposed. One of the most successful search heuristics is the *Lin-Kernighan-Helsgaum* (LKH) algorithm (Helsgaun, 2000), that holds the record for finding the best solution for several TSP instances with unknown optima (Helsgaun, 2009). The core of LKH is the variable depth local search heuristic developed by Lin and Kernighan (Lin and Kernighan, 1973), but a number of improvements have been incorporated to LKH along the years. The latest versions of LKH explore general p-opt submoves and the partition of large instances into smaller subproblems. Also, in multi-trial LKH, the Iterated Partial Transcription (IPT) crossover operator is used to recombine solutions generated by soft restarts of the LK heuristic (Helsgaun, 2018). IPT has not been well documented in the literature. Thus, most researchers do not realize that recombination is an important part of the LKH algorithm. One disadvantage of IPT is that it has $O(n^2)$ complexity in the worst case. All of the partition crossover operators (PX, GPX, GAPX, and GPX2) have complexity $O(n)$.

The use of recombination operators such as IPT and GPX2 also has some similarity in motivation to the use of "Tour Merging" as proposed by Cook and Seymour (Cook and Seymour, 2003). However, this method relies on a branch decomposition of a graph combining two or more TSP tours. Because of the high cost of the branch decomposition, we have found that this is not a cost effective way of doing recombination. Cook and Seymour used tour merging to combine solutions found by different algorithms (or algorithm configurations) at the end of the runs. Thus they only used one application of tour merging at the end of the search. Finding the best possible offspring as a result of recombination, given two parent solutions, is known as the Optimal Recombination Problem (Eremeev and Kovalenko, 2017). The Optimal Recombination Problem is an NP-hard problem (Eremeev and Kovalenko, 2014), but understanding this problem may motivate the design of new efficient recombination operators such as partition crossover operators.

The next section discusses the key ideas that motivate the use of partition crossover operators. The GPX2 recombination operator is presented in Section 3. GPX and GAPX were already used in Evolutionary Algorithms and Variable Local Search Algorithms (Whitley et al., 2010; Hains et al., 2012; Tinós et al., 2014). Section 4 demonstrates the

ability of GPX2 to improve solutions generated by multi-trial LKH. Experimental results show that GPX2 generates more successful recombinations than GPX and IPT. This paper is concluded in Section 5.

2 Partition Crossover Operators

All partition crossover operators are deterministic in the sense that crossover points are not chosen randomly; we include IPT in the set of partition crossover operators (Tinós et al., 2018a). Partition crossover operators exploit natural decompositions that are inherent in the parents. Most recombination operators used by evolutionary algorithms are stochastic in terms of how crossover points are selected and in terms of the number of potential offspring they generate. The number of crossover points under partition crossover is explicitly determined by the decomposition of the parents.

Unlike other recombination operators for the TSP, all of the partition crossover operators (including IPT) are "respectful" and "transmit alleles" (Radcliffe, 1994; Radcliffe and Surry, 1995). In "respectful" recombination, all common features (edges in the TSP case) found in both parents are always inherited by the offspring. All offspring generated by operators that "transmit alleles" are composed only of features (edges) contained in the parents. An operator that "transmits alleles" cannot introduce a new edge not found in either parent. Radcliffe (1994) defined the concepts of "respectful" and "transmitting" recombination because offspring that are generated by these types of recombination operators will have an evaluation that is more correlated with the evaluation of parents, since offspring are only composed of edges found in the parents.

The first step of partition crossover for the TSP is to create the union graph $G_u = P_1 \cup P_2$ where $P_1 = (V, E_1)$ and $P_2 = (V, E_2)$ are the parent solutions. Examples of the graph G_u are presented in Figures 1 and 2. We also make the following observation about the graph G_u: every vertex has degree 2, 3 or 4. If a vertex has degree 2, then the edges that touch that vertex are the same in both parents, and thus are automatically inherited by the offspring. This means that vertices of degree 2 can be removed from graph G_u as a first step, and all remaining vertices are of degree 3 or 4. Common edges, i.e., edges shared by both parents, are then removed from G_u to create a graph G'_u. This breaks the graph into multiple connected-subgraphs.

Definition 1. *A* **candidate component** *is made up of one or more connected subgraphs of* G'_u.

Definition 2. *A* **portal** *is a vertex in a candidate component that connects to another vertex in a different candidate component. Thus, portals exist as pairs,* v_i *and* v_j*, such that* v_i *and* v_j *are in different* **candidate components** *of* G_u*, but* v_i *and* v_j *are either 1) directly connected by common edges, or 2)* v_i *and* v_j *are connected by common edges after other candidate components have been removed from graph* G'_u*. Both* v_i *and* v_j *are* **portals.**

Definition 3. *A* **recombining component** *of graph* G_u *is a candidate component of* G'_u *such that: 1) it contains* z *vertices, where* $2x$ *vertices are portals that connect to other recombining components by common edges, and the remaining* $z - 2x$ *vertices only connect to vertices inside the recombining component; 2) it has the property that there exists a traversal of the two parent permutations over the vertices in graph* G_u *such that both parents* **enter** *the recombining component at exactly the same portals and then* **exit** *the recombining component at exactly the same portals.*

Informally, a traversal of the parent permutations divides the $2x$ vertices that act as portals into x "entry" portals and x "exit" portals. Note that the role of entry and exit can be reversed (e.g., by reversing permutation P_1 and/or $P2$).

Common edges in G_u act as "bridges" between recombining components. In the simplest case there is one entry and one exit, as illustrated in Figure 1. In Figure 1 vertex 1 is a portal in the same recombining component as vertex 8, which is also a portal. One parent traverses the recombining component in the order $< 1, 12, 11, 10, 9, 8 >$ while the other parent traverses the recombining component in the order $< 1, 11, 12, 9, 10, 8 >$. These partial solutions allow "transmitting" recombination to occur because the partial solutions that are exchanged have the same portals (1 and 8) and traverse the same subset of vertices. We can think of 1 as the entry and 8 as the exit; or we can reverse both permutations and think of 8 as the entry and 1 as the exit.

But, as shown in Figure 5, there can also be multiple entry and exit points. In Figure 5 in one recombining component the portals are the vertices 1, 5, 7, 10 and in the other recombining component the portals are 2, 4, 8 and 9. When candidate components are not recombining components, two or more candidate components can be merged or "fused" to discover a new recombining component (Section 3.3). Fusion can also result in indirect "bridges" between recombining components.

The original GPX and the IPT operator only detect recombining components with a single entry and single exit. In the original description of IPT (Möbius et al., 1999), after vertices of degree 2 are removed, the IPT operator examines all subchains of vertices of length from 4 to $n/2$ (see Appendix A). Assuming that one parent traverses the recombining component with a lower cost, the offspring inherits the subchain of vertices with the lower cost: this "fills" the corresponding positions in the offspring. This process continues iteratively until the offspring inherits a permutation of all of the vertices, ideally composed of a mix of subcircuits inherited from both parents.

The description of IPT (Möbius et al., 1999) found in the literature is incomplete in two ways. First, IPT must examine all subchains of vertices of length from 4 to $n/2$ in both the forward and backward direction. This is perhaps implied, but never explicitly stated. For example, assume vertex v_α is found at the beginning of a subchain found by IPT that admits recombination, and vertex v_ω is at the end of the subchain. If vertex v_α is 100 cities to the left of v_ω in parent $P1$, v_α could be 100 cities to the left of v_ω in parent $P2$, or v_α could be 100 cities to the right of v_ω in parent $P2$. Since subchains are examined only up to length $n/2$, for sufficient large values of n it is necessary to explicitly enumerate the subchains both left and right in one (but not both) of the parents. Second, IPT must deal with subchains that are nested. Much like matching nested parenthesis in a sentence, the shortest subchains are found first and removed. Furthermore, after a subchain is found and removed, it is necessary to again check smaller subchains (e.g. as small as 4 vertices) around the location where the subchain was removed: if there is a recombining component spanning 20 vertices nested inside of a recombining component spanning 24 vertices, when the smaller recombining component is found and removed, the surrounding recombining component now is composed of 4 (remaining) vertices. Examples of this kind of nesting are found in Figures 2 and 4.

Theorem 1. *The Iterated Partial Transcription procedure that scans all subchains of length 4 to $n/2$ has $O(n^2)$ complexity.*

Proof. Consider the case where there are no opportunities for recombination and the number of vertices of degree 2 is some fraction of n. After vertices of degree 2 are removed, the procedure must start at every possible vertex and scan every subchain of length from 4 to $n/2$. Explicitly examining these subchains requires $O(n^2)$ time. □

In practice, if there are many vertices of degree 2, this can potentially help to reduce the runtime cost of IPT. If there are many short recombining components, these can also

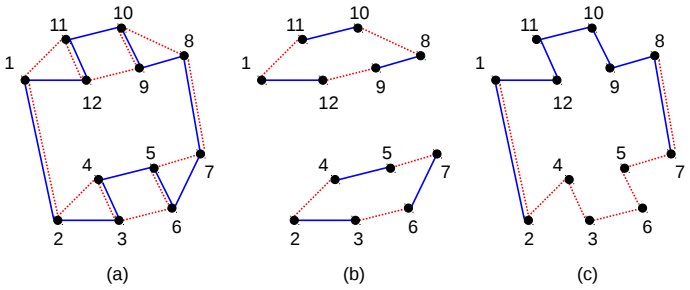


Figure 1: Example of recombination by GPX. a) Union graph composed of blue (solid) and red (dashed) parent solutions. b) The common edges of the union graph are removed and the connected subgraphs are identified. The **portals** are the pair of vertices 1 and 2, as well as 7 and 8. Both connected subgraphs are recombining components because each one has only one entry and one exit. c) Offspring generated by recombining the best partial solutions inside each partition.

be found first, and then removed, which again will reduce the runtime cost.

All of the partition crossover operators are guaranteed to execute in $O(n)$ time (Theorem 6 in Section 3.4). GPX2 will also find recombinations that are not found by IPT. A simple example of recombining two solutions using partition crossover is illustrated in Figure 1. The union graph G_u decomposes into two subgraphs when common edges are deleted to create graph G'_u. Recombination disconnects the graph by cutting common edges shared between the two parents. All the candidate components with one entry and one exit with at least 4 vertices are identified as *recombining components*.

All the paths inside a recombining component are inherited from one or another parent. Choosing the paths from one or another parent is done independently for the recombining components. GPX and GPX2 differ in the way they identify recombining components. In all partition crossover operators, the vertices of the TSP that were not assigned to recombining components, i.e., the remaining union graph, also compose a recombining component. This means that the paths in the remaining union graph also comes from one or another parent and the choice is independent from the other recombining components. Partition crossover operators consider all possible combinations of the recombining components. If k recombining components are identified, 2^k offspring are possible, including 2 equal to the parents. Solutions generated by partition crossover are always Hamiltonian circuits (Whitley et al., 2009), and thus feasible solutions. In addition, the partial solutions defined by each parent inside a recombining component can be independently evaluated.

2.1 Decomposition of the evaluation function

Assume the parents are P_1 and P_2 and two potential offspring different from the parents, C_1 and C_2, can be generated by partition crossover using only 2 recombining components. Under the objective function $f(\mathbf{x})$ (Whitley et al., 2009):

$$f(P_1) + f(P_2) = f(C_1) + f(C_2)$$

This can be seen by looking again at Figure 1. Graph 1a represents the union of the two parents, thus the sum of all the edges in Graph 1a is given by $f(P_1) + f(P_2)$. Let Graph 1c represent the child C_1. The child C_2 would be composed of all the common

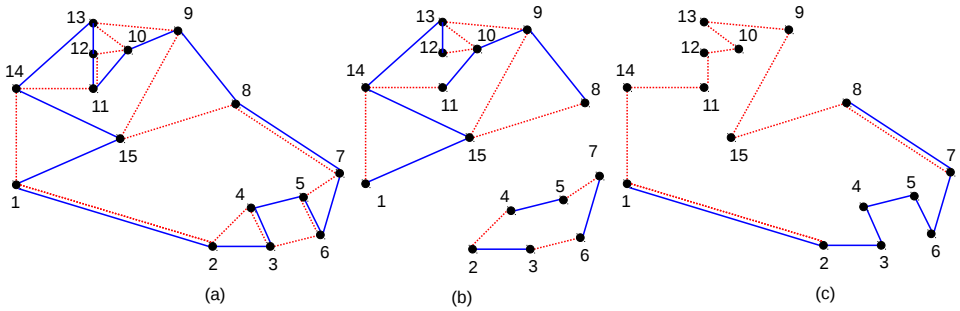


Figure 2: Recombination by GPX in an example with 15 vertices. a) Union graph. b) Recombining components. c) Offspring.

edges from Graph 1a, plus all of the edges that are in Graph 1a that are not in Graph 1c. Thus, by definition: $f(P_1) + f(P_2) = f(C_1) + f(C_2)$.

Let k denote the number of recombining components. The number of potential offspring produced by recombining parents P_1 and P_2 is 2^k. Using simple counting arguments that average over all possible offspring yields the following result (Chen et al., 2018):

$$\frac{f(P_1)}{2} + \frac{f(P_2)}{2} = \frac{1}{2^k} \sum_{i=1}^{2^k} f(C_i)$$

Partition crossover operators are only used to generate one offspring: recombination greedily produces the *best possible offspring*. By the "best possible offspring", we mean that partition crossover finds the offspring generated by the best possible combination of recombining components and not the best solution in the whole dynastic potential (Eremeev and Kovalenko, 2014). As already noted, if there are k recombining components, crossover will return the best of 2^k possible offspring. This makes partition crossover operators highly exploitative in nature: crossover retains the best combination of edges in the parents, but partition crossover cannot generate new edges that are not found in the parents. This makes partition crossover operators very different than operators such as Edge Assembly Crossover (Nagata and Kobayashi, 1997; Honda et al., 2013; Nagata and Kobayashi, 2013), which is more explorative in nature because it can introduce new edges into offspring that are not found in either of the parents.

2.2 Tunneling between local optima using GPX

Tinós et al. (2015) have shown that it is possible to apply partition crossover to bit representations for k bounded pseudo-Boolean functions[1]. One can then prove that the offspring are locally optimal in the largest hyperplane subspace that contains both parents.

In the case of the TSP, we can observe that every offspring is also *piecewise locally optimal*. Each recombining component that is inherited from one or the other parents is such that it cannot be improved by the same local search operator used to ensure that the parents are locally optimal. If the resulting offspring is not a true local optimum,

[1] In (Tinós et al., 2018b), the partition crossover presented in (Tinós et al., 2015) was extended to work with integer representation in clustering problems.

then the local search operator must find an improving move that involves simultaneously changing at least two of the recombining components at the same time.

Partition crossover operators are not effective at finding decompositions of the parents when the parents are just randomly generated solutions in the search space. This is because parents must share a subset of edges in order for decomposition to be possible. Randomly generated solutions (i.e., randomly generated permutations) are not similar enough to make the required decomposition possible in most cases, and thus, partition crossover will not be efficient.

Whitley et al. (Whitley et al., 2010) noted that GPX found on the order of 25 recombining components on two TSP instances (rand1500 and u1817[2]) using 3-opt at the local search operator. This means that GPX returned the best of 2^{25} possible offspring, and more than half of these were also local optima in the original search space.

3 Generalized Partition Crossover: GPX2

This section presents five major enhancements that increase the number of recombining components found by GPX2 in the symmetric TSP: 1) splitting vertices of degree four to create more connected-subgraphs; 2) identifying recombining components with more than one entry and one exit; 3) merging or "fusing" neighbor candidate components that are not recombining components; 4) scanning one of the two parents in both the forward and reverse directions to identify more recombining components; and 5) merging or "fusing" more complex, potentially nested candidate components with multiple entries and exits. An early implementation of GPX2 was used in combination with the Edge Assembly operator to improve the EAX algorithm (Sanches et al., 2017); this implementation, however, was incomplete and used only the first three enhancements.

We next discuss the five major enhancements used by GPX2. As a preprocessing step, we first remove all vertices of degree 2.

Procedure 0: *Remove all vertices of degree 2. All common edges are inherited by the offspring.*

3.1 Finding potential components by splitting vertices of degree 4

Splitting vertices of degree 4 can help to further decompose the recombination graph. Three recombining components can now be identified in Figure 2.a. The connected component in Figure 2.b can be split into two recombining components if we separate the graph at vertices 9 and 14, both of degree 4. Figures 3.a and 3.b illustrate how the graph presented in Figure 2.a can be divided at vertices of degree 4. Each vertex v of degree 4 is broken into vertices v and v', with zero cost on the common edge between them. We denote v' as a *ghost* vertex. Ghost vertices are removed after recombination.

In GPX2, splitting vertices of degree 4 creates a new common edge between the original vertex and the ghost vertex. When this common edge is removed, it can expose a new opportunity for recombination by further decomposing the graph G_u.

Unfortunately, every vertex of degree 4 can be split in two ways. So which of the two splits is correct? The solution lies in looking at a directional ordering of the vertices, which we denote as the "flow" of the Hamiltonian circuit. This corresponds to writing down a permutation corresponding to the Hamiltonian circuit for each parent, and then scanning the permutations in the same direction, or in the opposite direction. By imposing a directional flow, some portals function as *entries* and other portals function

[2]The numbers in the names indicate the number of cities in the instance.

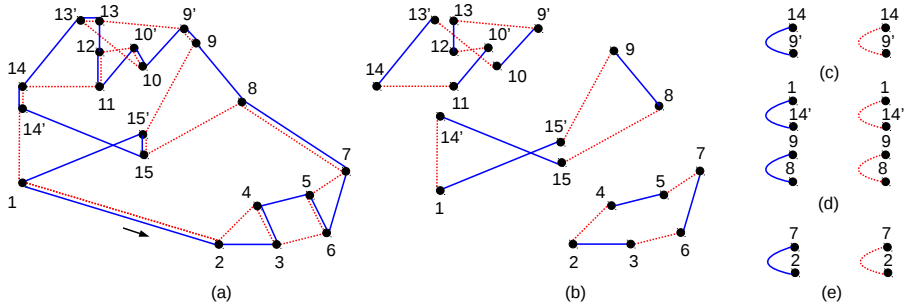


Figure 3: Example of recombination by GPX2. a) Insertion of ghost nodes after vertices of degree 4. b) Three AB cycles are identified. The three AB cycles are recombining components because their respective simplified graphs (c, d, e) are equal.

as *exits*. The "flow" of the solutions is represented by an arrow in Figure 3.a. The insertion of ghost nodes is captured in the following procedure.

Procedure 1 (Creating ghost nodes): *Determine a direction of flow in each parent P_1 and P_2. For each vertex v with degree 4 in G_u, insert a ghost vertex v' immediately after v in both P_1 and P_2. The common edge between v and v' is assigned weight 0. Remove all common edges to create the graph G'_u.*

Not every vertex of degree 4 that is split will further divide the graph G'_u. This can be seen by considering vertices 10, 10', 13, 13', 15 and 15' in Figure 3.b.

3.1.1 AB cycles

After all vertices of degree 2 are removed, and all vertices of degree 4 have been split, all of the vertices in the graph G'_u have degree 3. After common edges are removed, this completely decomposes the graph G'_u into *AB cycles*. An AB cycle is a subcircuit of G'_u where every edge from Parent 1 is immediately followed by Parent 2. This leads to a series of edges, $ABABAB...AB$ where A represents any edge from Parent 1 and B represents any edge from Parent 2. The AB cycles are obvious in Figures 1b and 3b.

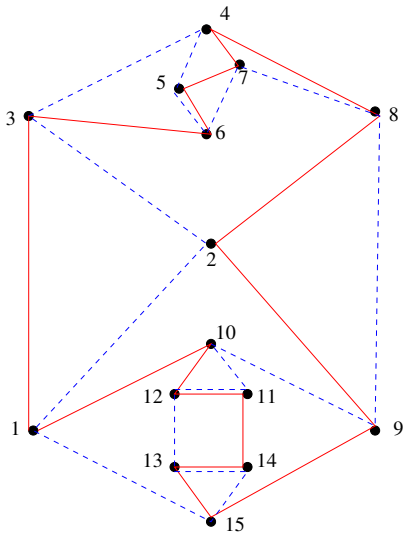
Lemma 2. *Removing common edges from graph G'_u decomposes the graph into AB cycles.*

Proof. Every vertex of graph G'_u has degree 3. Thus, each vertex is touched by one common edge and two non-common edges. After the removal of the common edge incident to a vertex, only the two non-common edges, one from each parent, remain. Any traversal of the non-common edges must use one non-common edge from one parent to "reach" the vertex and the other non-common edge from the other parent to "leave" the vertex. Because there is only 1 way to reach a vertex and 1 way to leave a vertex, an AB cycle must form. Finally, every vertex must be part of some AB cycle. □

It follows from Lemma 2 that *every* connected subgraph of the graph G'_u must be an AB cycle of G'_u. We illustrate this in Figure 4. The forward traversal of P_2 exposes AB cycles 1, 2 and 3; the reverse traversal of P_2 exposes AB cycles 4, 5, 6, and 7. The nesting of AB cycle 7 inside of AB cycle 1 is not detected by the forward scan.

3.1.2 Finding AB cycles using an extended edge table

Table 1 presents an example of an *Extended Edge Table*. The Extended Edge Table automatically captures the graphs G_u and G'_u. For each vertex, the table stores the non-



P1 = 1 1' 2 2' 3 3' 4 4' 5 6 7 7' 8 8' 9 9' 10 10' 11 12 13 14 15 15'
P2 = 1 1' 3 3' 6 5 7 7' 4 4' 8 8' 2 2' 9 9' 15 15' 13 14 11 12 10 10'

Cycle 1: 1 10' 11 14 15 9' 10 12 13 15' 1 Two portals: 1 and 9'
Cycle 2: 1' 2 8' 9 2' 3 1'
Cycle 3: 3' 4 7' 8 4' 5 7 6 3' Two portals: 3' and 8

P1 = 1 1' 2 2' 3 3' 4 4' 5 6 7 7' 8 8' 9 9' 10 10' 11 12 13 14 15 15'
reverse P2 = 9 9' 2 2' 8 8' 4 4' 7 7' 5 6 3 3' 1 1' 10 10' 12 11 14 13 15 15'

Cycle 4: 1 3' 4 8' 9 15' 1
Cycle 5: 1' 2 9' 10 1'
Cycle 6: 2' 3 6 7 4' 5 7' 8 2'
Cycle 7: 10' 11 14 15 13 12 10' Two portals: 10' and 15

Figure 4: This figure shows the AB cycles obtained following two scans. In one scan both parents are scanned in the same direction, in the other scan the direction of flow for the second parent, P2, is reversed. In practice, the forward flow and reverse flow can be captured in a single scan.

common edges that connect to adjacent vertices for each parent, as well as the common edge. One can trace the edges in an AB cycle by alternating between parent 1 and parent 2 at each step (there is no need to "search" for connected subgraphs). As the AB cycle is traced, the label for that cycle can be placed in the "Cycles" column.

When constructing the Extended Edge Table it is also not necessary to explicitly "reverse" the traversal of the parents. A single forward traversal is sufficient. Before the vertices are split, if the forward traversal is v_1, v_2, v_3, the reverse traversal is obviously v_3, v_2, v_1. We also need to determine if v_2 should be split or not. Given this information, a small table look-up (with 8 alternatives) is sufficient to determine all of the edge table entries in both the forward and reverse direction. For example, if v_2 is split, the entries for v_2' are always:

$$\text{Vertex} = v_2' \quad \text{Edge P2-Forward} = v_3, \quad \text{Edge P2-Reverse} = v_1, \quad \text{Common} = v_2$$

We note that the Extended Edge Table also can be used to implement IPT as well as GPX2. Indeed we can construct a new implementation of IPT that executes in $O(n)$ time. But we have found that the best way to implement GPX2 is not to build on top of IPT. This means GPX2 will sometime find different crossover opportunities than IPT.

Next, each AB cycle is assigned either to the set of recombining components, or to the set of *candidate components*. If an AB cycle has exactly two portals (one entry and one exit), then it is automatically a recombining component. If an AB cycle has exactly four portals, but it also contains a reversed embedded AB cycle, then it is also automatically a recombining component.

All other AB cycles with more than two portals are candidate components for recombination. Candidate components are identified as recombination components using procedures explained in next sections. We want to find as many candidate components as possible in order to explore a large number of recombination components. We

Table 1: An example of an **Extended Edge Table** that identifies AB cycles and entry/exit points into the AB cycles shown in Figure 4. Every vertex has degree 3. "Edge P1" is from Parent 1, "Edge P2" is from Parent 2, and "Common" is the edge found in both parents. An "Entry" can be either an entry or exit. The AB cycles 1, 2 and 3 are in the forward direction. The AB cycles 4, 5, 6 and 7 are in the reverse direction. A vertex v_j is a potential entry if the common edge associated with v_j connects to a different AB cycle than v_j. An AB cycle is automatically identified as a recombining component if there are only two entry vertices in that AB cycle.

Vertex	Edge P1	Edge P2-Forward	Edge P2-Reverse	Common	Cycles Forward	Portal Forward	Cycles Reverse	Portal Reverse
1	15′	10′	3′	1′	1	Yes	4	Yes
2	1′	8′	9′	2′	2		5	Yes
3	2′	1′	6	3′	2	Yes	6	Yes
4	3′	7′	8′	4′	3		4	Yes
5	4′	7	7′	6	3		6	
6	7	3′	3	5	3		6	
7	6	5	4′	7′	3		6	
8	7′	4′	2′	8′	3	Yes	6	Yes
9	8′	2′	15′	9′	2	Yes	4	Yes
10	9′	12	1′	10′	1		5	Yes
11	10′	14	14	12	1		7	
12	13	10	10′	11	1		7	
13	12	15′	15	14	1		7	
14	15	11	11	13	1		7	
15	14	9′	13	15′	1		7	Yes
1′	2	3	10	1	2	Yes	5	Yes
2′	3	9	8	2	2		6	Yes
3′	4	6	1	3	3	Yes	4	Yes
4′	5	8	7	4	3		6	Yes
7′	8	4	5	7	3		6	
8′	9	2	4	8	2	Yes	4	Yes
9′	10	15	2	9	1	Yes	5	Yes
10′	11	1	12	10	1		7	Yes
15′	1	13	9	15	1		4	Yes

use a procedure where we switch the direction of flow, each time adding the shortest AB cycles to the set of candidate components. The list of AB cycles for each flow direction is obtained from the Extended Edge Table. Only the lines of the table that were not assigned to candidate components in previous scans need to be updated. If the maximum number of scans is limited by a positive constant n_r, the candidate components are identified in $O(n)$ time (see Theorem 6 in Section 3.4). Finding the shorter AB cycles is important in order to find as many candidate components as possible.

Let s_f denote the size of the smallest unassigned AB cycle in the forward scan and let s_r denote the smallest unassigned AB cycle in the reverse scan. Both s_f and s_r assume maximum size (n plus number of ghost nodes) if there are no unassigned AB cycles in their respective flow direction. Procedure 2 is employed by GPX2 in order to assign AB cycles to the set of candidate components or to the set of recombining components. We want to find the smaller AB cycles first in order to identify embedded recombining components.

Procedure 2 (Assign AB cycles):

i) *Create the Extended Edge Table;*

ii) $counter = 0$;

iii) All AB cycles are initially unassigned.

iv) BEGIN WHILE ($counter < n_r$ and $there\ exists\ unassigned\ cycles$)
IF ($s_f < s_r$)
Assign AB cycles smaller than s_r and with 2 portals to the set of recombining components;
Assign AB cycles smaller than s_r and with > 2 portals to the set of candidate components;
ELSE IF ($s_r < s_f$)
Assign AB cycles smaller than s_f and with 2 portals to the set of recombining components;
Assign AB cycles smaller than s_f and with > 2 portals to the set of candidate components;
ELSE
Assign AB cycles with size equal to s_r and with 2 portals to the set of recombining components;
Assign AB cycles with size equal to s_r with > 2 portals to the set of candidate components;
Remove **portals** associated with assigned recombining components;
Update s_f and s_r;
$counter = counter + 1$;
END WHILE

v) If there are unassigned vertices, add the respective AB cycles to the set of candidate components;

3.2 Recombining components with multiple entries and exits

Figure 5 shows an example of candidate components with multiple entries and exits. In this case, there are two recombining components, where each recombination component has 2 entries and 2 exits, making of a total of 4 portals in each recombining component. Two mechanisms to identify recombining components with more than two portals are presented in this section, and are analyzed in Section 3.4

Consider the paths inside a candidate component for each parent. If both paths enter at the same entry points and exit at the same exit points in both parents for a candidate component, then the paths can be exchanged. The resulting solutions are Hamiltonian cycles (Section 3.4). In this way, a recombining component can be identified by analyzing the sequence in which each tour transverse the entries and exits of a candidate component. The decomposition into AB cycles means that the internal vertices have already been "grouped" together; thus, the internal vertices can be ignored and only the *portals* need to be considered for all candidate components. A practical way to implement this idea is to use simplified graphs for the tours inside the candidate components (see Figures 3 and 5), as outlined in Procedure 3.

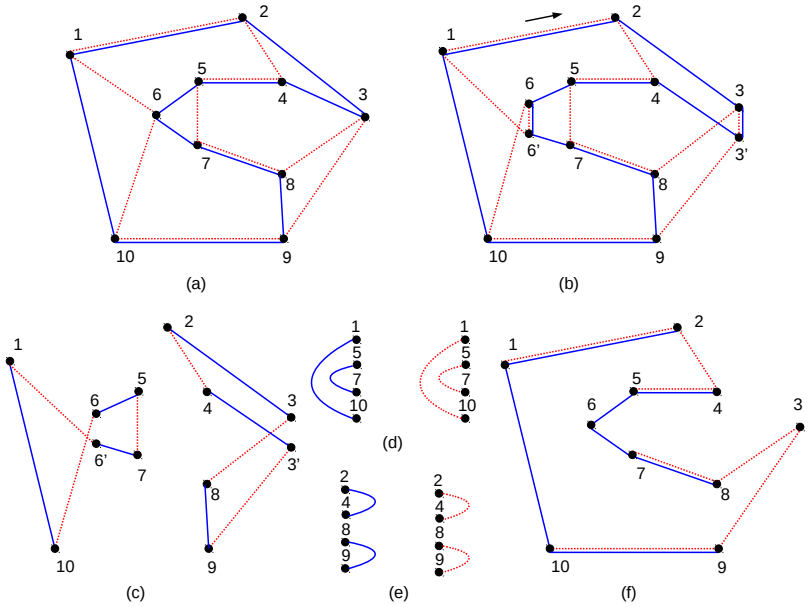


Figure 5: GPX2 applied to an example with candidate components with multiple entries and exits. a) Parent solutions. b) Ghost vertices are inserted after vertices of degree 4 (Procedure 1). c) Two candidate components are identified by Procedure 2. d), e) The candidate components are recombining components because their respective simplified graphs are equal for the parent solutions (Procedure 3). If the paths inside one recombining components are exchanged, the offspring must be Hamiltonian cycles. f) Offspring.

Procedure 3 (Simplified graphs inside of candidate components):

i) *For each candidate component, create a simplified undirected graph G_{in} for each parent solution (tour). In order to build a simplified graph for each candidate component and parent, replace every internal path that connects a portal to a portal by a single edge.*

ii) *If the simplified graphs inside the candidate component are equal for both parents (using a forward or reverse traversal), then the candidate component is a recombining component.*

Figures 3.c-e show the application of Procedure 3 to classify the candidate components presented in Figure 3.b. Figure 5 shows an application of Procedure 3 in a case with two recombining components, each one with 2 entries and 2 exits.

Recombining components with exactly 2 portals can be identified either in Procedure 2 or in Procedure 3. Some candidate components that are not initially identified as recombining components may be identified as recombining components by Procedure 4. Figure 6 shows an example of Procedure 4.

Procedure 4 (Simplified graphs with nested candidate components):

i) *Remove portals associated with known recombining components.*

ii) *If the number of updated portals is 2, then the candidate component is a recombining component.*

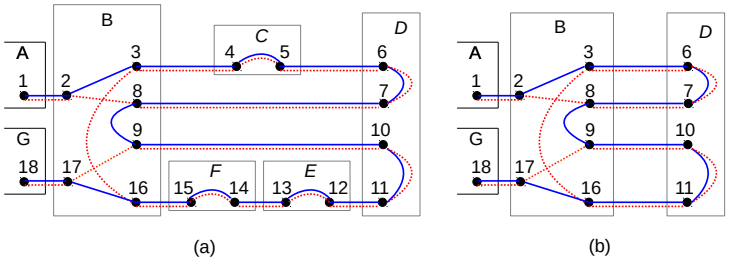


Figure 6: Identifying recombining components using Procedure 4. In this example, only a subsequence of the parent solutions is presented; only those vertices are shown that are used to construct the simplified graphs. The candidate components C, E, and F have two portals and can be identified as recombining components by either Procedure 2 or Procedure 3. The candidate component D has four portals and is identified as a recombining component by Procedure 3. After vertices 3, 8, 9 and 16 are identified as known portals, candidate component B is also recognized as a recombining component.

3.3 Fusion of infeasible components

Some candidate components do not correspond to recombining components. If one candidate component is not a recombining component, we will call it an *infeasible component*. It is sometimes possible to merge or "fuse" two infeasible components to generate a new recombining component. In this section, we present two procedures that fuse infeasible components in order to search for new recombining components.

3.3.1 Fusion Type 1: Fusion between neighboring candidate components

Two infeasible components are neighbors if they are connected by at least one common edge. Note that candidate components that are not adjacent neighbors can *become* neighbors after the removal of a recombining component. For example, in Figure 6, vertex 3 is connected to vertex 6 after the recombining component C is removed.

Figure 7.b shows 6 candidate components identified by GPX2 for the parents shown in Figure 7.a. Two of the candidate components (A and B) in Figure 7.b are recombining components. The remaining four candidate components are infeasible components, each of them with two neighbors. The infeasible component C is neighbor of subgraphs D and E. While C and D have two common edges between them, candidate components C and E have only one.

Fusing two candidate components creates one recombining component, which means that the offspring inherits the partial solution in the new recombining component from only one parent. As can be seen in Figure 7.b, an infeasible component can be a neighbor of more than one infeasible component (e.g., C is neighbors with both D and E). Testing the best of all combinations of neighbors for fusion can result in a procedure with non-linear cost. Therefore, the following heuristic is applied.

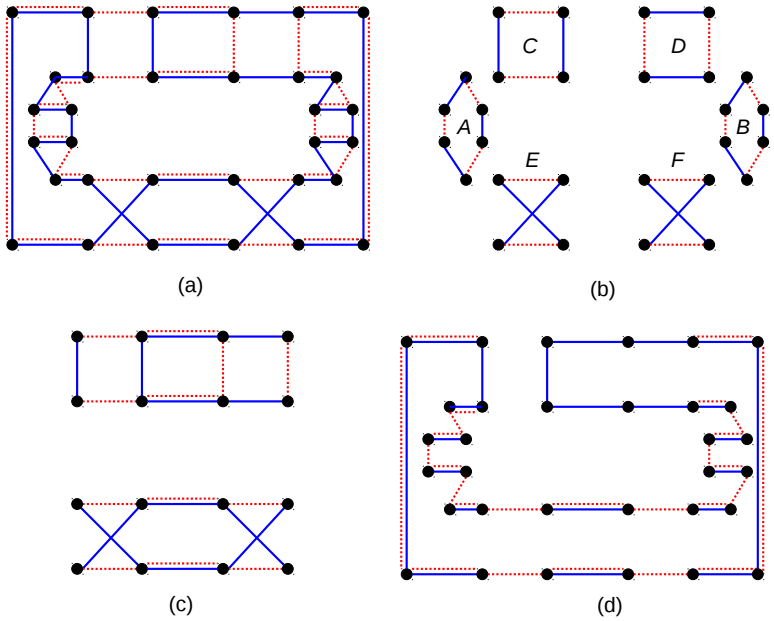


Figure 7: Fusion type 1. a) Parent solutions. b) Candidate components are identified: A and B are recognized as recombining components. Candidate components C, D, E and F are infeasible components, with 4 portals each. Infeasible component C has two neighbors: component D, with 2 common edges to C, and component E, with 1 common edge to C. c) Neighboring infeasible components with more common edges are fused (Procedure 5). Fusing C with D and fusing E with F yield recombining components because the respective simplified graphs are equal for the parent solutions (Procedure 3). d) Offspring.

Procedure 5 (Fusion type 1):

i) *Identify infeasible components with at most two neighbors that are infeasible components*;

ii) *Perform fusions between all neighboring pairs of infeasible components; the infeasible components identified in step i) with more common edges between them are fused first;*

iii) *Apply Procedure 3 to determine if each fusion yields a recombining component.*

Limiting the number of neighbors of each candidate component to 2 results in a procedure with complexity $O(n)$. Performing the fusions between neighbors with more common edges increases the probability of generating recombining components. Also note that if the number of edges between two candidate components is odd, these components **must** be fused in order to create or discover a new recombining component. Figure 7.c shows the fusion of the infeasible components presented in Figure 7.b.

Procedure 5 can be repeated n_f times in order to obtain additional recombining components. Each iteration of fusion results in larger infeasible components. If n_f is a constant, the complexity of the procedure is still $O(n)$.

3.3.2 Fusion Type 2: Fusion of nested and embedded candidate components

Another special case is when two or more candidate components are embedded inside of a larger recombining component. An example is presented in Figure 8. Parent 1 enters candidate component E (at vertex 8) then alternates successively between E and F until it leaves F (at vertex 15). After E and F are fused, there are only 2 (active) portals (vertex 8 and 15), thus the result must be a recombining component. Candidate components E and F are also nested within components C and D. After C and D are fused, there are only 2 (active) portals (vertex 4 and 19), thus the result must be a recombining component. To find these types of fusions, we introduce **high level cuts**.

Definition 4. *A* **high level cut** *is a vertex that is the first entry or the last exit for both parent tours in a candidate component. Alternatively, the vertex can be the first entry for one tour and the last exit for the other tour.*

High level cuts are identified at cost $O(n)$ by analyzing the simplified graphs and recording the index of each vertex in the parents. Note that a **high level cut** identifies a recombining component with two portals (the entry and exit vertex) but that recombining component may contain other embedded recombining components. Note that these **high level cuts** find opportunities for recombination that are also found by IPT, except that IPT finds these cuts by enumeration of subchains of vertices. Here, we look for combinations of candidate components that have two portals after one or more fusions occur.

Procedure 6 (Fusion type 2):

i) *Identify the high level cuts, i.e., the first entry and last exit common to both parents*;

ii) *Create sequences $s1$ and $s2$ containing the corresponding candidate components for the parents*;

iii) *Create an undirected graph where each node v_i corresponds to a candidate component c_i. For each transition in s1 or s2 between vertices of the TSP that are not high level cuts, add an edge between the corresponding components in the graph*;

iv) *The connected components in the graph are new candidate components. Each candidate component is eligible for fusion type 2*;

v) *Apply again Procedure 3 to verify if the new candidate components are recombining components;*

vi) *Apply Procedure 4 successively while it finds at least one recombining component or while the number of applications is smaller than or equal to n_r.*

Procedure 4 (in step vi) is repeated for a maximum number of times equal to n_r. GPX2 also successively applys Procedure 4 in order to deal with different levels of nesting. Figure 8 illustrates fusion type 2 (Procedure 6).

The implementation[3] of GPX2 is described in Algorithm 1. An offspring is generated by selecting the path inside each recombining component from one or another parent. Thus, there are 2^k possible ways of generating an offspring when k recombining components are found. GPX2 finds the best of the 2^k reachable offspring by selecting the shortest path inside each recombining component from one of the two parents (Step 9 in Algorithm 1).

[3]The source code of GPX2 is available at `https://github.com/rtinos/gpx2`.

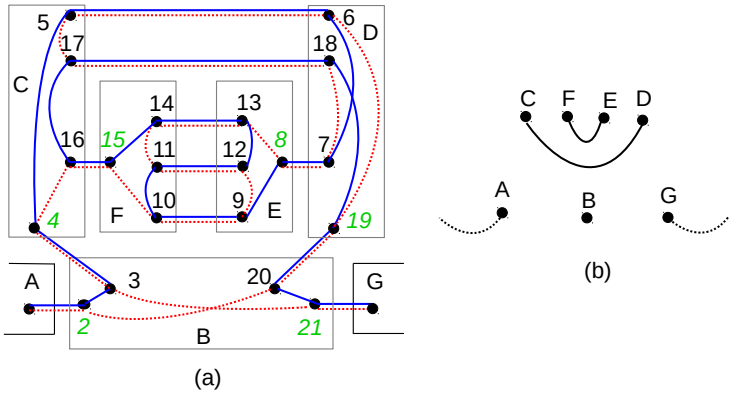


Figure 8: Example of fusion type 2 (Procedure 6) in a problem with nested and embedded components. Only a subsequence of the parent solutions is presented. a) Vertices 2, 4, 8, 15, 19, and 21 are high level cuts. b) First a new recombining component formed by the fusion of E and F is identified using Procedure 3. Then, Procedure 4 identifies the recombining component formed by the fusion of C and D. Finally, B is also identified as a recombining component using Procedure 4.

3.4 Analysis

In the following, we present a theorem stating that recombination using GPX2 results in offspring that are Hamiltonian circuits. First, we present a lemma about the identification of the cutting points for the candidate components.

Lemma 3. *A portal of a recombining component in $G'_u = P'_1 \cup P'_2$, where P'_1 and P'_2 are created from graphs P_1 and P_2 using Procedure 1, can only be a vertex of degree 3.*

Proof. The procedure that creates ghost vertices removed all vertices with degree 4 from the union graph G_u. Vertices of degree 2 are also removed from the union graph, G_u and replaced by a common edge. Thus, all vertices in the graph G'_u have degree 3. □

Theorem 4. *The candidate components identified using procedures 3 and 4 are recombining components. The result of recombination using these recombining components yields a Hamiltonian circuit.*

Proof. Every AB cycle of the graph G'_u with more than one vertex is a candidate component. By Lemma 3, the portal vertices of a candidate component are also portal vertices for both Hamiltonian cycles (parents). However, the order in which the portals are visited can be different for the paths given by each parent.

In GPX2, the offspring is generated by selecting the shortest path inside each recombining component. The selection of one of the paths inside a recombining component does not depend on the selection of paths outside this recombining component. Thus, when analyzing one recombining component, the paths outside this recombining component can be considered fixed. We have, in this way, four possible offspring, but only two different from the parents. 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 candidate component. V_s contains the entry/exit vertices of the candidate component and $E_{in}^{p_i}$ is the subset of edges representing the paths inside the candidate component

Algorithm 1 GPX2

Step 1. Remove and pass to the offspring all vertices of degree 2, leaving one common edge (Procedure 0).
Step 2. Create a ghost vertex for 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).
Step 3. Create the Extended Edge Table, which captures forward and reverse adjacent edges in Parent 2 (P_2). This also identifies all of the AB cycles (candidate components).
Step 4. Use Procedure 3 to determine if the candidate components are feasible for recombination, i.e., recombining components. To test each candidate component, create 2 simplified undirected graphs, one for each parent. The simplified graphs should contain as vertices only the entry and exit vertices of the candidate component. Inside the candidate component, replace the path between each entry and respective exit vertex by a single edge. Verify if the two simplified graphs are equal. If the graphs are equal, the candidate component is a recombining component.
Step 5. Use Procedure 4 to determine if the remaining candidate components are recombining components. Given a parent solution, create a sequence s2 containing the corresponding candidate components. Before, remove the 2-degree vertices. Create sequence s3 by removing from s2 the previously assigned recombining components. Compute the number of transitions (i.e., entries and exits) in s3 for each candidate component. If the number of transitions for a candidate component is 2, then it is a recombining component.
Step 6. Perform fusion type 1 (Procedure 5). Identify the infeasible components that are neighbors of at most two other infeasible components. Perform the fusion between the pairs of components with more common edges among them. Apply again Procedure 3 to verify if the resulting candidate components are recombining components (Step 4). Repeat this step n_f times.
Step 7. Perform fusion type 2 (Procedure 6). Find the high level cuts. Create sequences s1 and s2 from the parents containing the corresponding candidate components. Create an undirected graph where each node is a candidate component. For each transition in s1 or s2 between vertices of the TSP that are not high level cuts, add an edge between the corresponding components in the graph Each connected component in the graph is a new candidate component. Apply Procedure 3 to identify the recombining components. Then, successively, apply Procedure 4 to identify the nested recombining components while it finds at least one recombining component or while the number of applications is smaller than or equal to n_r.
Step 8. The vertices of the TSP that were not assigned to recombining components, i.e., the remaining union graph, compose a recombining component.
Step 9. Generate the offspring by selecting the shortest path inside each recombining component.

for parent p_i. Let $G^{p_1}_{out} = (V_s, E^{p_1}_{out})$ and $G^{p_2}_{out} = (V_s, E^{p_2}_{out})$ denote the simplified graphs for the parents with the edges representing paths **outside** the candidate component. $E^{p_i}_{out}$ is the subset of edges representing the paths outside the candidate component for parent p_i. Then the combined simplified graphs in the two offspring o_i different from the parents are $G^{o_1}_s = (V_s, E^{p_1}_{in} \cup E^{p_2}_{out})$ and $G^{o_2}_s = (V_s, E^{p_2}_{in} \cup E^{p_1}_{out})$.

In Procedure 3, a candidate component is identified as a recombining component only when the simplified graphs inside the candidate component are equal for both parents, i.e, $E^{p_1}_{in} = E^{p_2}_{in}$. Then the combined graphs of the offspring are equal to the simplified combined graphs for the parents, i.e., $G^{p_1}_s = G^{p_1}_{in} \cup G^{p_1}_{out}$ and $G^{p_2}_s = G^{p_2}_{in} \cup G^{p_2}_{out}$. Since the simplified combined graphs for both parents are Hamiltonian cycles, the offspring generated by Procedure 3 are also Hamiltonian cycles.

In Procedure 4, a candidate component is identified as a recombining component if the number of portals is 2. Both $E^{p_1}_{out}$ and $E^{p_2}_{out}$ are fixed if we choose one or another parent, i.e., they do not change because they contain already assigned recombining components. Thus, $E^{p_1}_{out} = E^{p_2}_{out}$. Then the combined graphs of the offspring are equal to the simplified combined graphs for the parents, i.e., $G^{o_1}_s = G^{p_1}_{in} \cup G^{p_1}_{out}$ and $G^{o_2}_s =$

$G_{in}^{p_2} \cup G_{out}^{p_2}$. Since the simplified combined graphs for both parents are Hamiltonian cycles, the offspring generated by Procedure 4 are also Hamiltonian cycles. □

Despite the simplified graphs being equal in the parents and offspring, the resulting tours are not equal as the paths inside the candidate components are different for both parents.

Lemma 5. *The* **Extended Edge Table** *is constructed in $O(n)$ time.*

Proof. Splitting vertices at most doubles the number of vertices in the parents. Each vertex v_i occupies a row in the table. Both Parent 1 and Parent 2 can be scanned in $O(n)$ time to identify each vertex before and after vertex v_i in each parent. In constant time, the common edge is identified, as well as the non-common edges from each parent. Considering only the forward direction, each vertex can appear in only one AB cycle and all of the AB cycles can be labeled in $O(n)$ time by starting at any unlabeled vertex, and then alternating indices between parents to trace the AB cycle. Similarly the AB cycles in the reverse direction are also labeled in $O(n)$ time. After the AB cycles are identified, all of the AB cycles can be traced in $O(n)$ time since each vertex appears in only one AB cycle. For each vertex there is one common edge, and in constant time we determine if the vertex and the common edge are in the same AB cycle. Vertices connected by a common edge but located in different AB cycles are marked as portal vertices. □

Theorem 6. *The time complexity of the GPX2 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 Extended Edge Table and the identification of the AB cycles in one scan is $O(n)$ according to Lemma 5. The candidate components are identified in Procedure 2 by successively finding the smaller AB cycles and eventually changing the order of the ghost nodes. The maximum number of inversions of the direction of Parent 2 (the "red tour") in the table is n_r. Because n_r is constant, Procedure 2 is also $O(n)$. The same complexity $O(n)$ is observed in the procedure used to identify the entries, exits, and high level cuts in the candidate components. When Procedure 3 is applied to test the i^{th} candidate component, 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 component. As the sum of $n_{io}(i)$ for all candidate components is equal or less than the number of nodes in the union graph, then the procedure is $O(n)$. In Procedure 4, one parent should be scanned in order to find the sequences of candidate components, what results in $O(n)$ cost. Fusion type 1 (Procedure 5) is limited to infeasible components with at most two neighbors. Therefore, it is necessary only to store the number of common edges between each candidate component and at most two neighbors. This can be done visiting each vertex of the graph, resulting in an $O(n)$ procedure. Repeating fusion type 1 n_f times, where n_f is a constant, does not alter the complexity of the algorithm. Finally, for fusion type 2 (Procedure 6), sequences $s1$ and $s2$ are obtained in $O(n)$. The graph has a number of nodes that is equal to the number of infeasible components. The number of infeasible components is smaller than n. Thus, building the graph and finding the candidate components has $O(n)$ cost. Procedure 3 is then applied one time and Procedure 4 is applied for a maximum number of times equal to n_r which is a constant. Thus, Procedure 6 is also $O(n)$. □

4 Experiments

Previous versions of GPX2 (e.g, GPX and GAPX) were already successfully employed in Evolutionary Algorithms and Variable Local Search Algorithms (Whitley et al., 2010; Hains et al., 2012; Tinós et al., 2014). Section 4.1 presents results of experiments designed to test the ability of GPX and GPX2 to improve solutions generated by multi-trial LKH with IPT (Section 4.1). We investigate the frequency of tunneling between local optima when GPX and GPX2 are employed in Section 4.2. In Section 4.3, results of experiments where GPX2 replaces IPT inside LKH are presented. The experiments were executed in a server with 2 processors Intel Xeon E5-2620 v2 (15 MB Cache, 2.10 GHz) and 32 GB of RAM.

4.1 Using GPX2 to improve solutions found by LKH with IPT

The multi-trial LKH has improved solutions of several large TSPs with unknown optima, including symmetric TSPs with almost 2 million cities. Despite being an approximate algorithm, LKH was able to find the optimal solutions for all instances with known optima tested by K. Helsgaun (Helsgaun, 2018).

In the following experiments, GPX2 is used to improve solutions generated by multi-trial LKH. Multi-trial LKH is a form of iterated local search that uses a "kick" operator to escape local optima. Multi-trial LKH uses the recombination operator IPT in two different ways. First, IPT is used to recombine the local optimum found after the kick (trial) with the current best solution found in the run. Then, it is used to recombine solutions produced in different runs.

In the experiments presented here, we save the best solution found in each run of multi-trial LKH after it was recombined by IPT with the best solution so far if this improves the current best solution. Otherwise, we save the best solution of the run before the recombination. We then apply GPX2 to recombine the solutions produced in different runs by multi-trial LKH. First, GPX2 is applied between the solutions found in runs 1 and 2 of LKH. The offspring is then recombined with the best local optimum found in run 3 of LKH, and so on (Figure 9.a). Results of experiments of multi-trial LKH with 20 runs and different numbers of trials are presented. Therefore, GPX2 is applied 19 times in each experiment. The same experiment is repeated using GPX instead of GPX2.

We also tested an approach where GPX2 is applied to recombine exhaustively all solutions generated by multi-trial LKH. First, the solution found in the first run of multi-trial LKH is recombined to the other 19 solutions, like in the previous approach. Then, the second solution is recombined to the other 19 solutions. The best solutions found are then recombined. This process is repeated for each remaining solution $\mathbf{x}_i$, recombining the best solution so far to the best solution found by recombining solution $\mathbf{x}_i$ to the other 19 solutions. Thus, GPX2 is applied 20^2 times (more precisely: (20*19)/2). We denote this as *all-to-all approach*, while *incremental approach* denotes the case where crossover is applied only 19 times (Figure 9).

The experiments were repeated 10 times, each time with a different random seed for LKH. We show results where LKH is executed each time for 1, 20, or 1000 trials (with exception of instances monalisa100K and usa115475, where LKH is executed for 1 trial). Some of the best results of multi-trial LKH reported in the literature were obtained for running LKH for 1000 trials (Helsgaun, 2009). The parameters of LKH used in the experiments are presented in Appendix B. In GPX2, five cycles ($n_f = 5$) for fusion type 1 are considered and $n_r = 1000$ for fusion type 2 (see Step vi in Procedure 6). Previous experiments indicated that choosing values of n_r in the interval between 500 and 2000

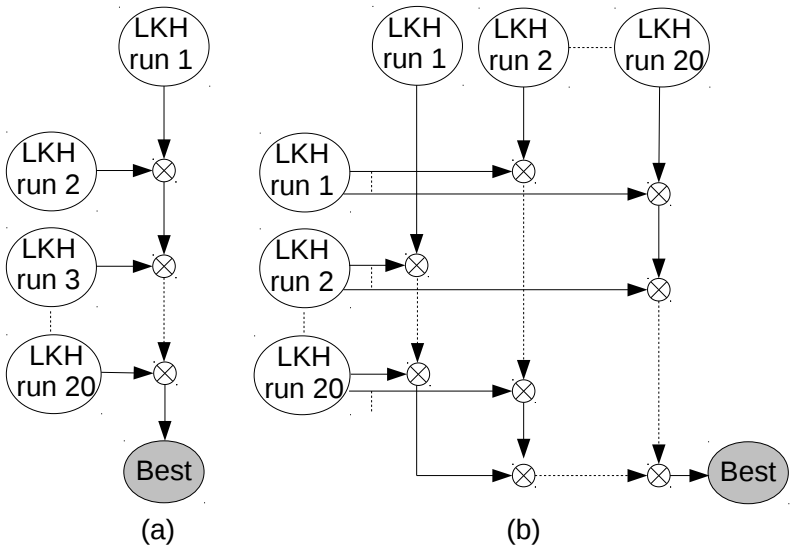


Figure 9: Incremental (a) and all-to-all (b) approaches. The symbol $\otimes$ indicates a recombination performed by GPX2.

does not significantly impact the performance of GPX2. Regarding the choice of n_f, experiments were done in order to test its impact (Figure 10). Higher values of n_r and, specially, n_f implies in higher running times for GPX2.

We tested GPX2 in four classes of symmetric TSPs with unknown optima. The 6 problems of Class 1 are artificial instances used in the 8th DIMACS Implementation Challenge (Johnson et al., 2013). In the E-instances, the location of the cites are uniformly generated in a square of 1,000,000 by 1,000,000 units (under the Euclidean metric). In the C-instances, locations of the cities consist of clustered points in the square. LKH currently holds the records for all 6 instances of Class 1 (Helsgaun, 2014). The records of the remaining problems are reported in (Cook, 2009). The 3 problems in Class 2 (pia3056, dke3097 and xqe3891) are from the VLSI TSP Collection. The 4 problems in Class 3 (tz6117, ym7663, ar9152, usa115475) are from the National TSP Collection. Finally, monalisa100K is an instance of the Art TSP Collection.

Table 2 shows the results of GPX and GPX2 applied to recombine the LKH solutions. When GPX and GPX2 are used to improve LKH solutions, we respectively call the algorithms LKH+GPX (for the incremental approach) and LKH+GPX2 (for the incremental and all-to-all approaches) The results of the percentage excess over the Held-Karp lower bounds (HK bounds) are shown. The best and the average results over 10 executions are presented. The best results of the literature (Helsgaun, 2014; Cook, 2009) are also reported. We applied the Jarque–Bera test to determine if the results are well-modeled by a normal distribution. The normality test indicated that the null hypothesis (data are normally distributed) cannot be rejected at the 0.05 significance level for most of the results. However, the null hypothesis can be rejected for some few results. Thus, we used the non-parametric Wilcoxon signed-rank test at the 0.05 significance level for the statistical comparisons of LKH+GPX against LKH+GPX2 (in both approaches). It is important to observe that GPX in the incremental approach was not able to improve solutions generated by LKH with IPT. Thus, the column 'GPX incremental approach'

Table 2: Percentage excess over the HK bounds for LKH+GPX and LKH+GPX2. Results for running LKH with 3 different numbers of multi-trials (denote by trials = 1, 20, or 1000) are shown. The symbols '=' and '+' respectively indicates that the mean for LKH+GPX2 is equal or better than the mean for LKH+GPX. The letter s indicates that the results are statistically significant.

Problem	Trials	GPX incremental approach		GPX2 incremental approach		GPX2 all-to-all approach		literature
		mean±std	best	mean±std	best	mean±std	best	
E10k0	1	0.7929 ±0.0176	0.7562	0.7770 ±0.0154 (s+)	0.7497	0.7682 ±0.0148 (s+)	0.7473	
n=10000	20	0.7446 ±0.0072	0.7357	0.7370 ±0.0080 (s+)	0.7244	0.7306 ±0.0067 (s+)	0.7216	
	1000	0.7083 ±0.0012	0.7056	0.7083 ±0.0012 (=)	0.7056	0.7082 ±0.0011 (+)	0.7056	0.7056
E10k1	1	0.7339 ±0.0114	0.7166	0.7218 ±0.0091 (s+)	0.7040	0.7199 ±0.0106 (s+)	0.7076	
n=10000	20	0.6919 ±0.0073	0.6797	0.6809 ±0.0059 (s+)	0.6721	0.6793 ±0.0070 (s+)	0.6708	
	1000	0.6539 ±0.0007	0.6526	0.6526 ±0.0008 (s+)	0.6514	0.6525 ±0.0008 (s+)	0.6514	0.6514
E31k0	1	0.7572 ±0.0051	0.7517	0.7478 ±0.0058 (s+)	0.7400	0.7446 ±0.0081 (s+)	0.7378	
n=31623	20	0.6979 ±0.0062	0.6860	0.6899 ±0.0059 (s+)	0.6808	0.6888 ±0.0056 (s+)	0.6797	
	1000	0.6487 ±0.0013	0.6467	0.6477 ±0.0016 (s+)	0.6454	0.6470 ±0.0013 (s+)	0.6453	0.6383
E31k1	1	0.7562 ±0.0068	0.7453	0.7459 ±0.0061 (s+)	0.7363	0.7442 ±0.0059 (s+)	0.7350	
n=31623	20	0.6980 ±0.0031	0.6938	0.6890 ±0.0044 (s+)	0.6830	0.6888 ±0.0063 (s+)	0.6808	
	1000	0.6444 ±0.0019	0.6419	0.6437 ±0.0019 (s+)	0.6413	0.6431 ±0.0018 (s+)	0.6410	0.6357
C10k0	1	1.2096 ±0.1514	1.0620	1.1888 ±0.1404 (s+)	1.0607	1.1740 ±0.1413 (s+)	1.0445	
n=10000	20	1.0963 ±0.1323	0.9441	1.0853 ±0.1372 (s+)	0.9270	1.0574 ±0.1596 (s+)	0.8793	
	1000	0.7319 ±0.0969	0.6677	0.7319 ±0.0969 (+)	0.6677	0.7286 ±0.0983 (+)	0.6677	0.6677
C10k1	1	1.1800 ±0.3015	0.9054	1.1650 ±0.3074 (+)	0.9005	1.1459 ±0.2891 (s+)	0.8993	
n=10000	20	1.0040 ±0.0767	0.8887	0.9968 ±0.0738 (+)	0.8887	0.9856 ±0.0699 (s+)	0.8876	
	1000	0.7262 ±0.0297	0.6988	0.7256 ±0.0293 (+)	0.6988	0.7207 ±0.0204 (+)	0.6988	0.6897
pia3056	1	1.1699 ±0.0318	1.1345	1.1699 ±0.0318 (=)	1.1345	1.1565 ±0.0190 (+)	1.1345	
n=3056	20	1.1259 ±0.0083	1.1100	1.1259 ±0.0083 (=)	1.1100	1.1234 ±0.0090 (+)	1.1100	
	1000	1.0623 ±0.0039	1.0610	1.0623 ±0.0039 (=)	1.0610	1.0623 ±0.0039 (=)	1.0610	1.0610
dke3097	1	1.3546 ±0.0221	1.3335	1.3527 ±0.0198 (+)	1.3335	1.3489 ±0.0223 (+)	1.3239	
n=3097	20	1.3268 ±0.0065	1.3239	1.3268 ±0.0065 (=)	1.3239	1.3268 ±0.0065 (=)	1.3239	
	1000	1.3239 ±0.0000	1.3239	1.3239 ±0.0000 (=)	1.3239	1.3239 ±0.0000 (=)	1.3239	1.3239
xqe3891	1	1.3101 ±0.0306	1.2620	1.2932 ±0.0349 (s+)	1.2535	1.2780 ±0.0354 (s+)	1.2367	
n=3891	20	1.2434 ±0.0148	1.2282	1.2400 ±0.0183 (+)	1.2114	1.2325 ±0.0114 (s+)	1.2198	
	1000	1.1911 ±0.0081	1.1861	1.1894 ±0.0071 (+)	1.1861	1.1861 ±0.0000 (+)	1.1861	1.1861
tz6117	1	0.0864 ±0.0259	0.0454	0.0799 ±0.0221 (+)	0.0454	0.0782 ±0.0215 (s+)	0.0454	
n=6117	20	0.0435 ±0.0066	0.0337	0.0403 ±0.0062 (s+)	0.0337	0.0391 ±0.0049 (s+)	0.0322	
	1000	0.0292 ±0.0009	0.0276	0.0292 ±0.0009 (=)	0.0276	0.0292 ±0.0009 (=)	0.0276	0.0276
ym7663	1	0.0549 ±0.0062	0.0441	0.0494 ±0.0080 (s+)	0.0386	0.0483 ±0.0079 (s+)	0.0378	
n=7663	20	0.0362 ±0.0023	0.0319	0.0346 ±0.0018 (s+)	0.0319	0.0344 ±0.0015 (s+)	0.0319	
	1000	0.0302 ±0.0000	0.0302	0.0302 ±0.0000 (=)	0.0302	0.0302 ±0.0000 (=)	0.0302	0.0302
ar9152	1	0.1560 ±0.0394	0.0952	0.1550 ±0.0393 (s+)	0.0952	0.1484 ±0.0358 (s+)	0.0952	
n=9152	20	0.0487 ±0.0087	0.0338	0.0478 ±0.0095 (+)	0.0338	0.0474 ±0.0090 (s+)	0.0336	
	1000	0.0205 ±0.0028	0.0166	0.0205 ±0.0028 (=)	0.0166	0.0198 ±0.0031 (+)	0.0134	0.0122
usa115475	1	0.8662 ±0.0032	0.8613	0.8610 ±0.0047 (s+)	0.8536	0.8593 ±0.0041 (s+)	0.8520	0.7359
n=115475								
monalisa100K	1	0.0279 ±0.0007	0.0270	0.0270 ±0.0010 (s+)	0.0256	0.0264 ±0.0009 (s+)	0.0254	0.0019
n=100000								

in Table 2 indicates results for both LKH with IPT and LKH+GPX. This is explained because both GPX and IPT find recombining components with 2 portals. However, IPT finds more recombining components because it iteratively removes one by one the recombining components, eventually transforming candidate components with more than 2 portals in recombining components with 2 portals. Thus, IPT finds more recombining components than GPX, which explains why LKH+GPX did not improve the solutions generated by LKH with IPT.

Table 3 summarizes the results. For the E-instances and C-instances, GPX2 in the incremental approach improved the best solution found by LKH 14 out of 16 times on those runs where LKH did not reach the best known solution reported in the literature. For those instances, GPX2 in the incremental approach was able to improve the best solution found by LKH with 1000 trials in 3 out of 4 runs where LKH did not reach the best known solution reported in the literature (Table 2). In fact, for E10k1, the in-

Table 3: Summary of the results for the experiments with LKH+GPX2 and LKH+GPX. The number of times that LKH+GPX2 improved the mean and best results of LKH (and also of LKH+GPX) is presented. For the best results, only the times that LKH did not reach the literature best are counted.

Problem Type	incremental approach		all-to-all approach	
	improved mean	improved best	improved mean	improved best
E and C Instances	17 out of 18	14 out of 16	18 out of 18	15 out of 16
VLSI TSPs	4 out of 9	2 out of 5	6 out of 9	3 out of 5
National TSPs	7 out of 10	2 out of 8	8 out of 10	5 out of 8
Art TSP	1 out of 1	1 out of 1	1 out of 1	1 out of 1

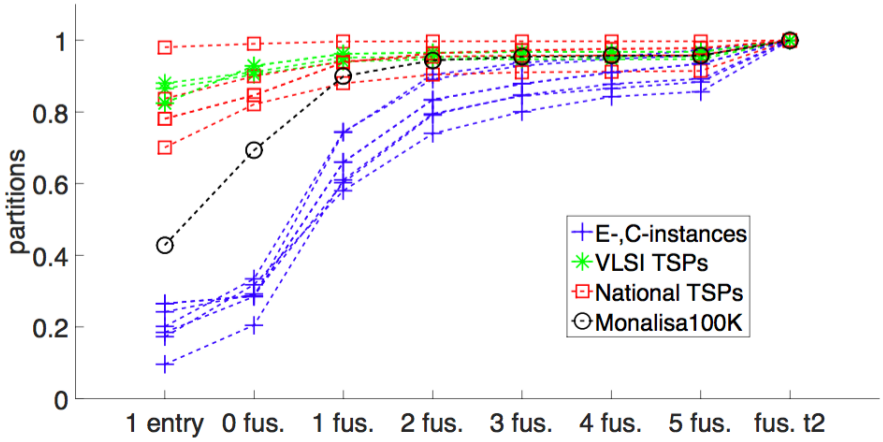


Figure 10: Scaled mean number of recombining components found in each application of GPX2 (incremental approach) for the experiments with 1 trial. The labels "0 fus." to "5 fus." refer to a type 1 fusion. The label "fus. t2" refers to a type 2 fusion.

cremental approach reached the best known result while LKH with 1000 trials reached a poorer result. For the VLSI TSP instances, the incremental approach improved the best results found by LKH in 2 out of 5 runs where LKH did not reach the best known solution reported in the literature.

The performance of GPX2 relative to IPT depends on finding more recombining components. Recall if there are k recombining components, recombination will return the best of 2^k possible offspring. Figure 10 shows the scaled mean number of recombining components found in each application of GPX2 (using the incremental approach) for the experiments with 1 trial. The scaled mean number of recombining components found with 1 to 5 cycles of fusion type 1 (n_f), with fusion type 2 and before the application of fusion is shown. The results for recombining components with 1 entry (and 1 exit) are also shown. The mean number and the maximum number of recombining components found by GPX2 are presented in Table 4. Comparing the number of recombining components with 1 entry with the number found before the fusions, we are comparing the capability of procedures 3 and 4 in detecting more recombining components. Comparing the number of recombining components before and after the fusions, we are comparing the capability of procedures 5 and 6 in producing more recombining components. The results for GPX are also presented in Table 4.

From Figure 10, we observe that the number of recombining components found by GPX2 does not increase with the same magnitude in the different classes of prob-

lems. The number of recombining components increases more for the E-Instances and C-Instances and the Art TSP; fusions increase the mean number of recombining components by approximately 5 fold. These are the instances where GPX2 had the best performance. For example, in the experiment with 1000 trials of LKH for instance E10k1 (Table 4), the mean number of recombining components found by GPX2 (1.9) is more than 10 times higher than the mean number of recombining components found by GPX (0.1). For monalisa100K, the maximum number of recombining components found in one application of GPX2 was 436, which allowed GPX2 to return the best of 2^{436} reachable offspring. When GPX was applied, the maximum number of recombining components found in one application of recombination was only 37.

For the VLSI and National TSPs, the increase in the number of recombining components produced by GPX2 is not significant (Figure 10). This occurs because the distribution of vertices is much more regular in these two classes of problems. As a consequence, most of the recombining components in the union graph G_u have only 1 entry (and 1 exit). In this case, the benefits of fusion are small because there are few infeasible components. Therefore, GPX2 does not significantly improve over the performance of IPT. However, we should also note that these problems are "easy" for IPT in the sense that there are many recombining components with 1 entry (and 1 exit). For example, for problem C10k1 where fusion is helpful, the number of recombining components found by GPX2 before fusion is 2.1 on average for 1 trial LKH. For GPX, the average number of recombining components is 2.6, indicating that GPX2 found smaller infeasible components with more than 1 entry. However, such components were merged after fusion, increasing the average number of recombining components to 7.4. Thus, fusion is most helpful when there are few recombining components with 1 entry and exit.

The performance of the all-to-all approach is better than the performance of the incremental approach. This is because recombination is being applied more frequently. The implementation of LKH with IPT employed here is an incremental approach. The all-to-all approach improved the mean and the best results of LKH more often than the incremental approach (Table 3).

4.2 Tunneling between local optima using GPX2

We investigated the frequency of tunneling between local optima when GPX2 is used to improve solutions generated by 2-opt local search. We also investigated the number of improvements and successful applications of the recombination operator. Ten local optima were generated by applying 2-opt local search in randomly generated solutions. We then recombined every local optimum with every other local optimum using GPX2, thus producing 45 offspring. We repeated the same experiment using GPX as well. For GPX2, three cycles of fusion type 1 are applied ($n_f = 3$).

Table 5 shows the experimental results for 9 instances: rd100, rd400 (randomly generated instances), u574 (printed circuit board instance), xqf131, pbm436, xql662 (VLSI instances), ga194, att532, and uy735 (national city instances).

GPX2 generates many more recombination opportunities than GPX (the number of successful applications is given in Table 5). As a consequence, the number of improvements of the offspring is higher. Except for the two smallest problems, the average percentage of improvement for GPX2 was equal or higher than 90%. For instances rd100 and xqf131 the average percentage of improvement for GPX2 was respectively 61.8% and 65.6%. When GPX was applied, the lower and higher average percentages of improvement were respectively 20.4% and 61.3%.

The average percentage of local optima is lower for GPX2, when compared to GPX.

Table 4: Average number of recombining components found in each application of GPX and GPX2 (incremental approach). The maximum number of recombining components found in one application of GPX or GPX2 is also presented. For GPX2, the number of recombining components found before the fusions, after each cycle of Fusion Type 1 (1, 3, 5 cycles), and after Fusion Type 2 (Fus. Type 2) is presented.

Problem	Trials	GPX		GPX2					
		mean±std	max.	0 Fus.	1 Fus. type 1	3 Fus. type 1	5 Fus. type 1	Fus. type 2	max.
E10k0	1	0.4 ±0.2	3	1.0 ±0.3	1.9 ±0.3	2.6 ±0.5	2.8 ±0.5	3.3 ±0.5	9
	20	0.6 ±0.2	4	1.0 ±0.3	2.0 ±0.4	2.7 ±0.5	2.8 ±0.5	3.4 ±0.5	9
	1000	0.0 ±0.0	1	0.0 ±0.0	0.1 ±0.1	0.6 ±0.3	0.7 ±0.3	1.2 ±0.4	5
E10k1	1	0.2 ±0.1	2	0.7 ±0.3	2.1 ±0.5	2.8 ±0.7	3.0 ±0.7	3.4 ±0.6	9
	20	0.5 ±0.2	3	0.8 ±0.3	2.3 ±0.7	3.2 ±0.9	3.5 ±0.8	3.9 ±0.8	12
	1000	0.1 ±0.1	2	0.1 ±0.1	0.3 ±0.1	1.1 ±0.3	1.3 ±0.3	1.9 ±0.4	5
E31k0	1	1.2 ±0.4	5	2.3 ±0.6	5.0 ±0.6	6.3 ±0.6	6.5 ±0.6	6.8 ±0.6	19
	20	1.8 ±0.5	5	2.5 ±0.5	5.9 ±1.2	7.7 ±1.2	7.9 ±1.2	8.3 ±1.2	17
	1000	1.2 ±0.4	4	1.2 ±0.4	1.5 ±0.4	2.1 ±0.4	2.3 ±0.5	2.9 ±0.4	8
E31k1	1	1.3 ±0.5	5	2.0 ±0.8	5.2 ±0.7	6.5 ±1.1	6.8 ±1.1	7.0 ±1.2	16
	20	1.6 ±0.6	6	2.5 ±0.6	5.8 ±1.1	7.4 ±1.6	7.6 ±1.5	8.1 ±1.7	19
	1000	1.3 ±0.4	4	1.3 ±0.4	2.1 ±0.4	2.5 ±0.4	2.7 ±0.5	3.4 ±0.5	7
C10k0	1	2.5 ±0.6	11	2.1 ±0.6	4.3 ±0.9	6.0 ±1.1	6.4 ±1.1	7.1 ±1.2	22
	20	2.9 ±1.0	11	1.9 ±0.8	4.0 ±1.3	6.2 ±2.0	6.5 ±1.9	7.3 ±2.0	22
	1000	1.8 ±0.5	7	1.3 ±0.4	3.3 ±0.4	3.6 ±0.5	3.7 ±0.6	4.3 ±0.7	12
C10k1	1	2.6 ±0.6	11	2.1 ±0.4	4.9 ±1.1	6.5 ±1.4	6.9 ±1.7	7.4 ±1.7	26
	20	2.7 ±0.4	9	2.2 ±0.5	5.0 ±1.3	6.3 ±1.4	6.7 ±1.5	7.0 ±1.5	24
	1000	2.1 ±0.3	7	1.6 ±0.5	3.7 ±0.7	4.3 ±0.7	4.8 ±0.8	5.4 ±0.8	13
pia3056	1	32.5 ±2.3	48	35.1 ±2.2	36.6 ±2.3	36.7 ±2.3	36.7 ±2.3	38.5 ±2.2	55
	20	34.5 ±1.7	50	37.6 ±2.2	39.3 ±2.1	39.4 ±2.1	39.4 ±2.1	41.6 ±2.2	58
	1000	37.1 ±3.0	54	39.3 ±2.6	41.2 ±2.7	41.3 ±2.6	41.3 ±2.6	43.1 ±3.0	61
dke3097	1	30.9 ±2.4	51	33.1 ±2.6	34.4 ±2.7	34.6 ±2.8	34.6 ±2.8	36.6 ±3.0	57
	20	32.8 ±1.3	51	36.6 ±1.7	38.0 ±1.8	38.1 ±1.8	38.1 ±1.8	40.6 ±1.6	58
	1000	35.1 ±2.4	53	37.4 ±2.0	38.8 ±2.0	38.9 ±1.9	38.9 ±1.9	41.0 ±2.1	57
xqe3891	1	35.3 ±1.6	48	41.4 ±2.8	42.9 ±2.9	43.1 ±2.8	43.1 ±2.8	44.6 ±2.9	61
	20	37.4 ±3.2	55	43.1 ±3.2	44.7 ±3.4	44.8 ±3.3	44.8 ±3.3	46.4 ±3.6	67
	1000	38.0 ±2.1	55	44.3 ±2.3	45.7 ±2.3	45.7 ±2.3	45.7 ±2.3	47.1 ±2.1	70
tz6117	1	19.5 ±1.9	29	24.5 ±2.3	26.3 ±2.5	27.2 ±2.6	27.3 ±2.6	29.9 ±4.0	78
	20	19.8 ±2.2	33	24.8 ±2.7	26.9 ±3.0	27.6 ±2.9	27.6 ±2.9	30.4 ±3.4	47
	1000	25.0 ±0.6	39	29.9 ±1.3	33.1 ±1.0	33.5 ±0.9	33.5 ±0.9	37.1 ±1.0	49
ym7663	1	31.9 ±2.4	45	35.5 ±2.9	37.1 ±3.0	37.7 ±2.9	37.8 ±3.0	39.4 ±3.0	53
	20	34.9 ±1.3	47	38.3 ±1.6	39.9 ±1.8	40.6 ±1.8	40.7 ±1.9	42.4 ±1.7	59
	1000	37.9 ±1.8	51	39.9 ±1.8	41.1 ±1.8	41.3 ±1.8	41.3 ±1.8	42.0 ±1.9	57
ar9152	1	561.9 ±22.4	651	740.1 ±27.3	744.9 ±27.5	745.4 ±27.6	745.4 ±27.6	747.4 ±27.6	833
	20	606.3 ±60.1	722	770.2 ±79.1	774.9 ±79.6	775.4 ±79.7	775.4 ±79.7	778.5 ±79.9	884
	1000	675.0 ±22.6	762	820.6 ±28.1	825.8 ±28.3	826.3 ±28.3	826.3 ±28.3	829.7 ±28.4	925
usa115475	1	66.6 ±24.6	109	79.9 ±30.1	88.6 ±32.5	91.8 ±33.6	92.3 ±33.9	94.4 ±34.6	147
monalisa100K	1	24.0 ±4.2	37	46.5 ±9.8	60.3 ±11.6	64.0 ±11.8	64.2 ±11.9	67.1 ±15.1	436

This is partially explained by the fact that GPX is generating many offspring equal to the parents (see percentage of improvement in Table 5). This also results from the use of the extended edge table, which makes GPX2 more efficient. Another cause is the fact that GPX2 generates more recombination components and, as a consequence, smaller components. The 2-opt operator cannot improve the offspring by exchanging the edges of two nodes inside the recombination components. However, it can generate a successful move by exchanging the edges of nodes that belong to two different recombination components. Therefore, more recombination components can result in a better offspring, but ironically it can also result in a lower probability of generating local optima. This phenomenon of course depends on the disposition of vertices in the coordinate space.

Table 5: Average percentage of: successful applications, improvements, and local optima generated when GPX or GPX2 is applied. A successful application of GPX or GPX2 is when the operator found at least 2 recombining components when two parents are recombined. An improvement occurs when the evaluation of the offspring is better than the evaluation of both parents. Here, a local optimum is detected when 2-opt local search is not able to improve the offspring.

	GPX			GPX2		
Problem	% Successful	% Improving	% Local Optima	% Successful	% Improving	% Local Optima
rd100	48.4 ±6.3	27.3 ±7.6	98.5 ±3.5	86.0 ±7.0	61.8 ±4.0	92.8 ±4.5
rd400	52.2 ±9.5	28.4 ±5.8	98.0 ±4.2	99.8 ±0.7	96.2 ±3.9	73.1 ±4.0
xqf131	62.4 ±8.6	20.4 ±6.7	75.3 ±15.0	95.6 ±4.1	65.6 ±9.6	58.6 ±14.3
pbm436	95.1 ±2.9	38.4 ±13.1	35.7 ±5.9	100.0 ±0.0	91.3 ±4.2	6.7 ±3.3
xql662	85.1 ±7.3	35.3 ±7.7	96.2 ±2.8	100.0 ±0.0	94.7 ±3.8	72.7 ±5.8
u574	70.0 ±7.9	42.7 ±10.3	98.7 ±2.3	100.0 ±0.0	98.4 ±1.8	68.9 ±6.9
qa194	64.7 ±5.4	34.9 ±4.2	99.7 ±1.0	99.6 ±0.9	90.0 ±4.0	87.7 ±6.5
att532	92.7 ±5.1	61.3 ±8.5	98.5 ±2.2	100.0 ±0.0	98.4 ±1.8	70.7 ±6.6
uy734	85.3 ±8.1	60.0 ±8.4	97.8 ±2.7	100.0 ±0.0	99.3 ±1.1	62.2 ±8.8

4.3 Using GPX2 inside LKH

In the experiments presented in previous sections, GPX2 is used to improve solutions generated by LKH with IPT. Thus, the dynamics of LKH is not changed because solutions generated by GPX2 are not re-inserted in LKH. Here, we present results of experiments where IPT is replaced by GPX2 inside LKH, which results in a different heuristic. In the experiments presented here[4], LKH with IPT is compared to LKH with GPX2. The parameters of GPX2 are the same presented in Section 4.1.

The parameters of LKH used in the experiments are presented in Appendix B. The number of runs is 50 and the number of trials is 10 or 1000. Unlike the experiments in previous sections, the size of the population is 50. When the population size is 50, LKH also executes a simple genetic algorithm. In each run, the best solution is stored in a population if its fitness is different from the fitness of other solutions in the population. After each run, the stored solutions are selected and recombined using a variant of the Edge Recombination Crossover (EX) (Whitley et al., 1989). It is important to observe that IPT (or GPX2 when selected) is still used as described before; EX is used only after the end of each run to recombine the solutions of the population.

Tables 6 and 7 show the results of percentage excess over the HK bounds for LKH with IPT and LKH with GPX2. When the cost of the best result reported in the literature is equal to the best solution found by an LKH with IPT or GPX2, the number of runs needed for finding the best result is shown in parenthesis.

LKH with GPX2 obtained better performance than LKH with IPT for 13 out of 14 instances in the experiments with 10 trials (Table 6) and for 9 out of 14 instances in the experiments with 1000 trials (Table 7). GPX2 resulted in better performance because it was able to find many more recombination opportunities that improved the current best solutions. When GPX2 is applied after IPT in LKH, it finds many recombination opportunities missed by IPT (see (Tinós et al., 2018a)). When IPT is applied after GPX2, IPT usually cannot find recombination opportunities missed by GPX2. In the few cases where IPT applied after GPX2 was able to find additional recombination opportunities this can be explained by two factors: the limit n_r used in fusion type 2, and the order in which GPX2 applies fusions which results in different ways of finding recombining components.

[4]Some of the results presented in this section were previously presented in (Tinós et al., 2018a).

Table 6: Percentage excess over the HK bounds for LKH with IPT and LKH with GPX2 in the experiments with multi-trial LKH with 10 trials. The best results are in bold. When the cost is equal to the best result reported in the literature, the number of runs of LKH needed for finding the best result is shown in parenthesis.

Problem	LKH with IPT		LKH with GPX2		literature
	mean	best	mean	best	
E10k0	0.7785	0.7337	0.7387	**0.7121**	0.7056
E10k1	0.7033	0.6763	0.6902	**0.6617**	0.6514
E31k0	0.7386	0.7220	0.7174	**0.7014**	0.6383
E31k1	0.7454	0.7288	0.7211	**0.6981**	0.6357
C10k0	1.1323	0.9686	1.1383	**0.8843**	0.6677
C10k1	1.4177	0.9441	1.1430	**0.9294**	0.6897
pia3056	1.1912	1.1222	1.1805	**1.1100**	1.0610
dke3097	1.3854	**1.3239 (run 23)**	1.3952	1.3239 (run 26)	1.3239
xqe3891	1.3298	1.2114	1.2756	**1.1861 (run 23)**	1.1861
tz6117	0.0932	0.0469	0.0642	**0.0395**	0.0276
ym7663	0.0638	0.0386	0.0582	**0.0357**	0.0302
ar9152	0.0936	0.0564	0.1115	**0.0502**	0.0122
usa115475	0.8544	0.8474	0.8246	**0.8110**	0.7359
monalisa100K	0.0311	0.0293	0.0292	**0.0263**	0.0019

Finding more efficient recombination opportunities generally results in better performance. However, Tables 6 and 7 show that LKH with IPT can sometimes yield better results even when it finds fewer opportunities for recombination. Solutions obtained by recombination influence the future direction of the search, and sometimes LKH with IPT finds just the right mix of recombination, local search and soft restarts which results in better performance.

Table 8 shows the mean time for the runs for LKH with IPT and LKH with GPX2. Despite better results for the cost of the solutions, LKH with GPX2 generally resulted in higher mean running times: LKH with IPT presented smaller mean time for the runs for 8 out of 14 instances for the experiments with 10 trials and 10 out of 14 instances for the experiments with 1000 trials. This is also partly due to the use of fusions and is also partly a side-effect of the optimized implementations. Despite of the fact that the worst case complexity for IPT is $O(n^2)$, the implementation of IPT in LKH is highly optimized and the average time is linear in n (Tinós et al., 2018a). The experimental results indicate that LKH with GPX2 yields lower running time in very large instances. Table 8 shows that GPX2 resulted in lower mean time for runs in the four largest instances (E31k0, E31k1, usa115475, and monalisa100K) in the experiments with 1000 trials.

5 Conclusions

GPX2 is a a new form of efficient deterministic crossover which improves on earlier forms of partition crossover for the TSP. Partition crossover can be used to dramatically speed up search when used in combination with heuristics and metaheuristics. Our empirical results demonstrated how GPX2 can be used to improve the well-known LKH algorithm. In the experiments, GPX2 resulted in better performance when compared to IPT and GPX. Better performance is explained because GPX2 was able to find many more recombination opportunities that IPT and GPX.

We have also worked with K. Helsgaun to understand the differences between the implementation of our GPX2 code and his implementation of IPT. Prof. Helsgaun developed and maintains the LKH algorithm and software. Prof. Helsgaun has also developed a new version of LKH where GPX2 can be used instead of IPT. In LKH (version 2.0.8), IPT is the default but GPX2 can be selected by defining a specification

Table 7: Percentage excess over the HK bounds for LKH with IPT and LKH with GPX2 in the experiments with multi-trail LKH with 1000 trials.

Problem	LKH with IPT		LKH with GPX2		literature
	mean	best	mean	best	
E10k0	0.7161	0.7089	0.7125	**0.7056 (run 14)**	0.7056
E10k1	0.6605	**0.6514 (run 41)**	0.6563	0.6517	0.6514
E31k0	0.6593	0.6484	0.6543	**0.6480**	0.6383
E31k1	0.6583	0.6450	0.6481	**0.6411**	0.6357
C10k0	0.6866	**0.6677 (run 16)**	0.6942	0.6677 (run 36)	0.6677
C10k1	0.7379	**0.7134**	0.7473	0.7171	0.6897
pia3056	1.1159	**1.0610 (run 5)**	1.1220	1.0977	1.0610
dke3097	1.3256	1.3239 (run 2)	1.3266	**1.3239 (run 1)**	1.3239
xqe3891	1.2331	**1.1861 (run 2)**	1.2274	1.1861 (run 3)	1.1861
tz6117	0.0374	0.0276 (run 29)	0.0354	**0.0276 (run 6)**	0.0276
ym7663	0.0326	0.0302 (run 22)	0.0337	**0.0302 (run 4)**	0.0302
ar9152	0.0388	0.0263	0.0379	**0.0252**	0.0122
usa115475	0.7767	0.7725	0.7586	**0.7550**	0.7359
monalisa100K	0.0263	0.0239	0.0145	**0.0124**	0.0019

Table 8: Mean time for the runs (in seconds) for LKH with IPT and LKH with GPX2. The best results are in bold.

Problem	10 trials		1000 trials	
	LKH with IPT	LKH with GPX2	LKH with IPT	LKH with GPX2
E10k0	8.21	**7.86**	**263.96**	299.15
E10k1	**7.51**	7.90	**276.23**	304.34
E31k0	29.95	**29.10**	1767.79	**1713.40**
E31k1	**32.20**	33.45	1858.27	**1744.36**
C10k0	**7.45**	8.18	**389.63**	406.23
C10k0	7.61	**7.30**	**249.24**	280.47
pia3056	1.32	**1.29**	**56.01**	63.41
dke3097	**1.47**	1.70	**61.67**	75.83
xqe3891	**1.95**	2.00	**80.40**	104.65
tz6117	**3.23**	4.14	**188.36**	237.77
ym7663	4.31	4.31	**169.47**	194.24
ar9152	**9.55**	10.65	**723.72**	849.78
usa115475	186.33	**179.02**	13664.19	**13237.47**
monalisa100K	**201.14**	203.23	19901.08	**19061.51**

in the parameter file[5].

This paper has also explored how partition crossover operators such as GPX, IPT and GPX2 are able to tunnel between local optima: given two parents that are local optima, they generate offspring that are local optima with high frequency. These operators are all "respectful" and they "transmit edges" so that offspring are composed entirely of edges found in parents. This is both an advantage and a disadvantage. It allows partition crossover to be highly exploitative. Given a decomposition of the parents into k recombining components, recombination is guaranteed to yield the best of 2^k reachable offspring. Nevertheless, partition crossover has the disadvantage that recombination never generates or discovers "new" edges. Thus, some other mechanism must be used to introduce new edges into the search.

It is natural to ask how GPX2 and IPT compare to Edge Assembly Crossover (EAX). When EAX is used as part of an Evolutionary Algorithm it has also proven to be a very powerful search method for finding competitive solutions for very large TSP instances (Nagata and Kobayashi, 1997; Honda et al., 2013; Nagata and Kobayashi, 2013). EAX is highly explorative. GPX2 is highly exploitive. They serve completely different purposes. Our preliminary research suggests that using EAX and GPX2 together is better than using either recombination operator in isolation (Sanches et al., 2017).

[5]LKH is available at `http://www.akira.ruc.dk/~keld/research/LKH/`.

We have previous developed a version of GPX for the asymmetric TSP called Generalized Asymmetric Partition Crossover (GAPX). When LKH is applied to an instance of the asymmetric TSP, the original instance is transformed to an instance of the symmetric TSP with the double of its original size. GAPX is able to work directly in the asymmetric representation, using a directed graph to represent the problem. In fact, GAPX (Tinós et al., 2014) introduced two of the innovations employed by GPX2: i) finding potential components by splitting vertices of degree 4, and ii) finding recombining components with more than 2 portals. Recently, we added fusion type 1 to GAPX (Tinós and Whitley, 2018). Future work will investigate how to include the other innovations presented in GPX2 (e.g., the Extended Edge Table and fusion type 2) into the implementation of GAPX.

6 Acknowledgments

Renato Tinós was supported by FAPESP (under grants 2015/06462-1, 2013/07375-0, and 2016/18615-0) and CNPq. In U.S.A., 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. We would like to thank K. Helsgaun for the discussion of results comparing IPT and GPX2. The discussion helped us to develop some enhancements for GPX2. We are very grateful to K. Helsgaun by adapting GPX2 for LKH; this allowed us to generate the results presented in Section 4.3. We also would like to thank D. Hains for implementing the original version of GPX and D. Sanches for testing an intermediate implementation of GPX2.

References

Chen, W., Whitley, D., Tinós, R., and Chicano, F. (2018). Tunneling between plateaus: improving on a state-of-the-art MAXSAT solver using partition crossover. In *Proc. of GECCO'2018*, pages 921–928.

Cook, W. (2009). TSP test data. `http://www.math.uwaterloo.ca/tsp/data/index.html`. Updated in February, 2009.

Cook, W. (2011). *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton Univ. Press.

Cook, W. and Seymour, P. (2003). Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248.

Eremeev, A. V. and Kovalenko, J. V. (2014). Optimal recombination in genetic algorithms for combinatorial optimization problems: Part ii. *Yugoslav Journal of Operations Research*, 24(2):165–186.

Eremeev, A. V. and Kovalenko, Y. V. (2017). Genetic algorithm with optimal recombination for the asymmetric travelling salesman problem. In *International Conference on Large-Scale Scientific Computing*, pages 341–349.

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

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

Helsgaun, K. (2009). General k–opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2-3):119–163.

Helsgaun, K. (2014). DIMACS TSP challenge results: Current best tours found by LKH. http://www.akira.ruc.dk/~keld/research/LKH/DIMACS_results.html. Updated in October 6, 2014.

Helsgaun, K. (2018). LKH version 2.0.9. http://www.akira.ruc.dk/~keld/research/LKH/. Updated in July, 2018.

Honda, K., Nagata, Y., and Ono, I. (2013). A parallel genetic algorithm with edge assembly crossover for 100,000-city scale TSPs. In *Proc. of the 2013 IEEE Congress on Evolutionary Computation*, pages 1278–1285.

Hoos, H. H. and Stützle, T. (2014). On the empirical scaling of run-time for finding optimal solutions to the travelling salesman problem. *European Journal of Operational Research*, 238(1):87–94.

Johnson, D., McGeoch, L., Glover, F., and Rego, C. (2013). 8th DIMACS implementation challenge: The traveling salesman problem. http://dimacs.rutgers.edu/Challenges/TSP/. Updated in November 19, 2013.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2):498–516.

Möbius, A., Freisleben, B., Merz, P., and Schreiber, M. (1999). Combinatorial optimization by iterative partial transcription. *Physycal Review E*, 59(4):4667–4674.

Nagata, Y. and Kobayashi, S. (1997). Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem. In Bäck, T., editor, *Proc. of the Seventh International Conference on Genetic Algorithms (ICGA97)*.

Nagata, Y. and Kobayashi, S. (2013). A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing*, 25(2):346–363.

Radcliffe, N. and Surry, P. (1995). Fitness variance of formae and performance predictions. In Whitley, D. and Vose, M., editors, *Foundations of Genetic Algorithms, 3*, pages 51–72. Morgan Kaufmann.

Radcliffe, N. J. (1994). The algebra of genetic algorithms. *Annals of Maths and Artificial Intelligence*, 10:339–384.

Sanches, D., Whitley, D., and Tinós, R. (2017). Building a better heuristic for the traveling salesman problem: combining edge assembly crossover and partition crossover. In *Proc. of GECCO'2017*.

Tinós, R., Helsgaun, K., and Whitley, D. (2018a). Efficient recombination in the lin-kernighan-helsgaun traveling salesman heuristic. In *Proc. of PPSN XV*, pages 95–107. Springer.

Tinós, R. and Whitley, D. (2018). A fusion mechanism for the generalized asymmetric partition crossover. In *Proc of the 2018 IEEE Congress on Evolutionary Computation (CEC)*.

Tinós, R., Whitley, D., and Chicano, F. (2015). Partition crossover for pseudo-boolean optimization. In *Proc. of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*, pages 137–149.

Tinós, R., Whitley, D., and Ochoa, G. (2014). Generalized asymmetric partition crossover (GAPX) for the asymmetric TSP. In *Proc. of GECCO'2014*, pages 501–508.

Tinós, R., Zhao, L., Chicano, F., and Whitley, D. (2018b). NK hybrid genetic algorithm for clustering. *IEEE Transactions on Evolutionary Computation*, 22(5):748–761.

Veerapen, N., Ochoa, G., Tinós, R., and Whitley, D. (2016). Tunnelling crossover networks for the asymmetric TSP. In *Parallel Problem Solving from Nature – PPSN XIV. PPSN 2016. Lecture Notes in Computer Science, vol 9921*, pages 994–1003.

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

Whitley, D., Hains, D., and Howe, A. (2010). A hybrid genetic algorithm for the traveling salesman problem using generalized partition crossover. In *Proc. of PPSN XI*, pages 566–575. Springer.

Whitley, D., Starkweather, T., and Fuquay, D. (1989). Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proc. of ICGA'1989*, pages 133–140.

A IPT

Unlike PX, IPT works on the sequence representation of the tours(Möbius et al., 1999). Subchains in parents P_1 and P_2 with the same initial and final cities, and composed of the same cities, but in different order, are first searched We show in (Tinós et al., 2018a) that IPT can be classified as a partition crossover. The cities in a subchain found by IPT composes a recombining component. Because each subchain can be independently evaluated, the reachable offspring with the best cost can be found by selecting the best subchains. Using partition crossover terminology, the three main steps of IPT can be written as:

- **Removal of cities connected only to common edges:** when a city is connected to the same neighbors in P_1 and P_2, it is removed from the parents, resulting in reduced sequences.
- **Finding recombining components in reduced sequences:** suppose N_r is the size of the sequences after removing the cities connected to common edges. Let $v_s(v, P_1)$ be a city located $s-1$ positions from city v in P_2. Start with $s = 4$. For each city $v \in P_1$, verify if $v_s(v, P_1) = v_s(v, P_2)$, i.e., the subchains have the same initial and final cities. Subchains in both directions of P_2 must be tested. If the cities in the subchains of P_1 and P_2 are equal, then the indices in the subchains define a recombining component. Repeat, increasing s by 1, while $s \leq N_r/2$.
- **Creating the offspring:** select the best subchains in each recombining component and copy the cities connected only to common edges from one of the parents.

B Parameters of LKH

Parameters of LKH are specified in a parameters file. In the experiments where GPX2, or GPX, was used to improve the results of LKH with IPT (Section 4.1), the parameter MAX_TRIALS was 1, 20 or 1000. The other specified parameters were:

PATCHING_C = 3

PATCHING_A = 2

RUNS = 20

The other parameters were default and LKH version was 2.0.7. In LKH version 2.0.8, tours can be recombined by GPX2 instead of IPT by setting parameter RECOMBINATION = {IPT | GPX2}. In the experiments where GPX2 was used inside LKH (Section 4.3), the parameter MAX_TRIALS was 10 or 1000. The other specified parameters were:

POPULATION_SIZE = 50

RUNS = 50