Efficient Recombination in the Lin-Kernighan-Helsgaun Traveling Salesman Heuristic

Renato Tinós[1(✉)], Keld Helsgaun[2], and Darrell Whitley[3]

[1] Department of Computing and Mathematics, University of São Paulo, Ribeirão Preto, Brazil
rtinos@ffclrp.usp.br

[2] Department of Computer Science, Roskilde University, Roskilde, Denmark
keld@ruc.dk

[3] Department of Computer Science, Colorado State University, Fort Collins, USA
whitley@cs.colostate.edu

Abstract. The Lin-Kernighan-Helsgaun (LKH) algorithm is one of the most successful search algorithms for the Traveling Salesman Problem (TSP). The core of LKH is a variable depth local search heuristic developed by Lin and Kernighan (LK). Several improvements have been incorporated to LKH along the years. The best results reported in the literature were obtained by an iterative local search version known as multi-trial LKH. In multi-trial LKH, solutions generated by soft restarts of the LK heuristic are recombined using Iterative Partial Transcription (IPT). We show that IPT can be classified as a partition crossover. Partition crossovers use the features common to the parents to decompose the evaluation function. Recently, a new generalized partition crossover, known as GPX2, was proposed for the TSP. We investigate the use of GPX2 in multi-trial LKH and compare it to multi-trial LKH using IPT. Results of experiments with 11 large instances of the TSP indicate that LKH with GPX2 outperforms LKH with IPT in most of the instances, but not in all of them.

Keywords: Traveling Salesman Problem · Recombination operator
Heuristic search · Evolutionary combinatorial optimization

1 Introduction

The *Traveling Salesman Problem* (TSP) is one of the most investigated problems in Optimization [2]. Applications of the TSP can be found in the most diverse areas, such as Logistics, Bioinformatics, and Planning. Given a complete weighted graph $G(V, E)$, where V is a set of n vertices (cities) and E contains edges between every pair of vertices in V, the objective is to find the shortest Hamiltonian cycle. The evaluation of a solution (tour) $\mathbf{x}$ is given by:

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

A. Auger et al. (Eds.): PPSN 2018, LNCS 11101, pp. 95–107, 2018.
https://doi.org/10.1007/978-3-319-99253-2_8

where w_{x_i,x_j} is the weight of the edge between vertices v_{x_i} and v_{x_j} in V.

There are very good exact methods for the TSP, e.g., Concorde [2]. Concorde solves instances of the symmetric TSP with hundreds of cities in seconds. However, the TSP is NP-hard and, as a consequence, heuristic methods have been required for solving large TSP instances. One of the most successful heuristics for the TSP is the *Lin-Kernighan-Helsgaun* (LKH) algorithm [4,5]. LKH holds the record for several large instances of the TSP, some of them with more than 100,000 vertices. The best results of LKH reported in the literature were obtained by an iterative local search version known as multi-trial LKH. In multi-trial LKH, solutions generated by soft restarts of the LK heuristic are recombined using an efficient crossover operator, called *Iterative Partial Transcription* (IPT).

Recently, a new *generalized partition crossover*, known as GPX2, was proposed for the TSP [14]. Partition crossovers are deterministic recombination operators that use the features common to the parents to decompose the evaluation function [17]. The main contributions of this work are two. First, we show that IPT [10] is a kind of partition crossover. Second, we investigate the use of GPX2 in multi-trial LKH and compare it with multi-trial LKH using IPT. Unlike previous works with generalized partition crossovers [3,14], GPX2 is used here inside LKH. Before, generalized partition crossovers were used to recombine solutions generated by LKH but the offspring were not reinserted in LKH. Here, IPT is replaced by GPX2 inside LKH, which results in a different heuristic.

2 LKH Algorithm

LKH is an iterated local-search algorithm based on the Lin-Kernighan heuristic (LK) [9]. The local search performed by LK is based on k-opt moves. Given a tour $\mathbf{x}$, a k-opt move replaces k edges from $\mathbf{x}$ in order to create a solution $\mathbf{y}$ where $f(\mathbf{y}) < f(\mathbf{x})$. A k-opt move is incrementally obtained using basic moves, e.g., 2-opt, while the cumulative gain remains positive. Some heuristics (e.g., limiting the search to a subset of edges to the nearest neighbors of a node) are adopted in order to reduce the cost of the moves.

LK is an effective local search algorithm; implementations capable of finding solutions with typical cost 1–2% above the optimum cost were reported in the literature. A much more effective implementation of LK was reported in [4]. This implementation, called LKH, is able to find optimal solutions for large TSP instances with very high frequency [5]. Several improvements have been incorporated to LKH along the years. We present some of them in the following:

- **General k-opt moves:** In LK, moves are obtained by 2-opt or 3-opt moves followed by a sequence of 2-opt moves. Non-sequential moves are tried at the end if the sequential moves did not improve the original solution. LKH-1 [4] uses 5-opt sequential moves to create the sequence of basic moves. In LKH-2 [5], the basic moves are k-opt moves where k can be any integer greater than 1 and smaller than n. The moves are sequential, but non-sequential moves can also be tried during the search.

- **Partitioning:** Large instances of TSP are decomposed into smaller subproblems. Then, the solutions of the subproblems are used to improve the solutions of the original instance.
- **Candidate set criterion:** Instead of using the cost of an edge, the α-measure is used to evaluate the quality of an edge. The α-value of an edge e is computed as the increase of the cost of a minimum 1-tree when this tree is required to contain e. By restricting the search to a small number of neighbors of a node obtained according to a distance based on the α-measure, the time complexity is reduced.
- **Multiple trials:** In each run, the local optimum obtained is perturbed in order to generate a new initial solution for the LK strategy. Each run r of the multi-trial LKH is composed of t trials (Fig. 1). The use of multiple trials allows the use of strategies that explore information from different solutions in order to create a new solution. Three of them are presented in the following.
- **Backbone-guided search:** Edges of solutions previously obtained in different trials compose a set of candidate edges for the current trial.
- **Recombination of solutions:** Local optima share many partial solutions. The TSP has a multi-funnel structure, where many edges are common to the optima located in the same funnel [11]. Recombination operators are generally used in population meta-heuristics. However, recombination can also be used to merge solutions generated in different runs of an algorithm, in different trials of an iterated local search, or generated by different algorithms. In LKH-2, tours obtained in different trials and runs are recombined using IPT. Figure 1 shows how recombination is used in multi-trial LKH.
- **Genetic Algorithm:** Instead of storing only the best current solution, a population of solutions obtained in different runs can be stored. When the population size is different from one, a simple genetic algorithm is executed. For each run, the best solution is stored in the population if its fitness is different from the other solutions in the population. After each run, the stored solutions are selected and recombined using a variant of the Edge Recombination Crossover (ERX) [18]. It is important to observe that IPT is still used as shown in Fig. 1; ERX is used only after the end of each run to recombine the solutions of the population.

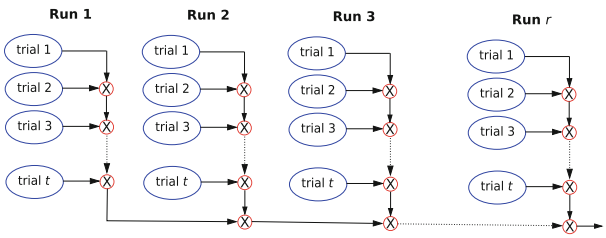

Fig. 1. Multi-trial LKH. The symbol $\otimes$ indicates a recombination operation.

3 Partition Crossover

Partition crossover (PX) is a deterministic recombination operator that can be applied in problems where the cost function, $f(\mathbf{x})$, is written as the sum of m subfunctions $f_i(\mathbf{x})$, i.e.:

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}) \tag{2}$$

where solution $\mathbf{x}$ is given by an n-dimensional vector and $m > 0$. Suppose that an offspring $\mathbf{z}$ is generated by recombining two solutions $\mathbf{x}$ and $\mathbf{y}$. If partition crossover is employed, then we can write:

$$f(\mathbf{z}) = \sum_{i \in S^x} f_i(\mathbf{x}) + \sum_{i \in S^y} f_i(\mathbf{y}) \tag{3}$$

where $|S^x| + |S^y| = n$. The subsets S^x and S^y contain the indexes of the decision variables inherited respectively from parents $\mathbf{x}$ and $\mathbf{y}$. The decomposition of the cost function is derived from two properties of partition crossover operators [12]: (i) the recombination is "respectful", i.e., the offspring inherits all features that are common to both parents; (ii) the recombination "transmits alleles", i.e., the offspring is composed only of features found in the parents. As a result of the properties of PX, the evaluation of the offspring is more correlated with the evaluation of the parents than in traditional recombination operators. Besides, if the parents are local optima with respect to a local search operator, then the offspring are guaranteed to be piecewise locally optimal under this local search operator. It was observed in different applications [13–16] that offspring are also very often true local optima when PX is employed.

The first step of PX is to remove all the features common to both parents. In the TSP, the features of a solution represented by $\mathbf{x}$ are the edges between two consecutive cities in $\mathbf{x}$. Define the union graph $G_u = G_x \cup G_y$, where the graphs G_x and G_y represent the parent solutions. The graph G'_u is obtained by removing the common edges from G_u.

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

Definition 2. *A* **recombining component** *is a candidate component 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) exactly* x *vertices that are portals serve as* "entry" *points, and* x *vertices that are portals serve as* "exit" *points to other recombining components; (3) the two parent solutions must enter and exit the recombining component at exactly the same entry and exit vertices.*

Inheriting one of the recombining components from one or another parent does not influence the evaluation of other recombining components. In other

words, the recombining components are subsets of features with independent evaluation. If p recombining components are found, there are 2^p different ways of combining the components to create an offspring. PX selects the best partial solution (from one or another parent) for each recombining component. Thus, the best of 2^p reachable offspring is found by PX.

Definition 3. *A* **partition crossover** *is a recombination operator that: (1) finds recombining components in the graph obtained by removing the common edges from the union graph G_u; (2) evaluates the cost of the partial solutions (for each parent) inside each recombining component; (3) generates the offspring by selecting the best partial solutions (from one or another parent) inside the recombining parents.*

GPX2 is a PX developed for the symmetric TSP. According to the definition of PXs, IPT can also be classified as a PX operator (see next section). IPT and GPX2 differ in the way the recombining components are found.

3.1 IPT

IPT [10] works directly on the sequence representation of the tours. IPT searches for subchains in parents $\mathbf{x}$ and $\mathbf{y}$ with: (i) the same initial and final cities; (ii) composed of the same cities, but in different order. According to the PX terminology, the subset of cities in a subchain composes a recombining component. Each subchain can be independently evaluated. By selecting the best subchains, the reachable offspring with the best cost is found. The three main steps of IPT, written according to the definition of recombining components, are:

- **Removal of cities connected only to common edges:** if a city is connected to the same neighbors in $\mathbf{x}$ and $\mathbf{y}$, then it can be removed from the tours, resulting in reduced sequences.
- **Finding recombining components in the reduced sequences:** suppose N_r is the size of the reduced sequences. Let $v_s(v, \mathbf{x})$ be a vertex located $s-1$ positions from vertex v in $\mathbf{x}$. Start with $s = 4$ (that is the minimum size of permutations that are different). For each vertex $v \in \mathbf{x}$, verify if $v_s(v, \mathbf{x}) = v_s(v, \mathbf{y})$, i.e., the subchains have the same initial and final cities. Subchains in both directions of $\mathbf{y}$ must be tested. If the subchains in $\mathbf{x}$ and $\mathbf{y}$ are composed of the same cities, then the subset of indices in the subchains define a recombining component. Repeat, increasing s by 1, while $s \leq N_r/2$.
- **Creating the offspring:** for 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.

An example of IPT is presented in Fig. 2. In this example, IPT first finds a recombining component with 4 cities. The cost of subchain $s_1^{\mathbf{x}}$ is smaller than the cost of $s_1^{\mathbf{y}}$. Thus, the offspring inherits $s_1^{\mathbf{x}}$. Then, it finds another recombining component with 5 cities. The cost of subchain $s_2^{\mathbf{y}}$ is smaller than the cost of $s_2^{\mathbf{x}}$. Thus, the offspring inherits $s_2^{\mathbf{y}}$. The implementation of IPT in LKH-2 is very efficient. Despite of the fact that the worst case complexity is $O(n^2)$, the average time is linear in n.

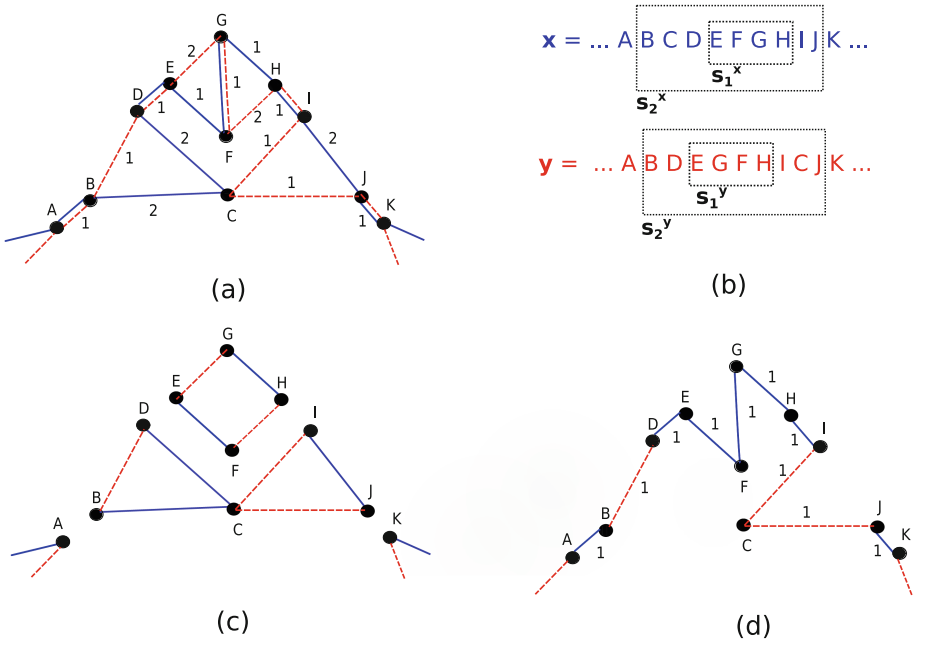

Fig. 2. Examples of recombination by IPT and GPX2. Only the paths between cities A and K are shown (suppose that, for IPT, $N_r > 20$). (a) Union graph composed of parents $\mathbf{x}$ (blue solid line) and $\mathbf{y}$ (red dashed line). (b) When IPT is applied, two subchains are identified for each parent. (c) When GPX2 is applied, candidate (connected) components are found after removing the common edges from the union graph. In this example, two candidate components are recombining components. (d) Offspring generated by IPT or GPX2. When compared to the parents, the offspring has better cost. (Color figure online)

3.2 GPX2

GPX1 [3] and GPX2 [14] work in the graph representation of the tours (Fig. 3). In GPX1, all candidate components linked to other parts of the graph G'_u by exactly two common edges are recombining components. Edges connecting a candidate component to other parts of the graph are *entries* for the tours in this candidate component. Thus, GPX1 finds only recombining components with two entries. The rest of the graph also composes a recombining component. Finally, the offspring is created by selecting, from one or another parent, the paths with the best cost inside each recombining component.

GPX2 presents 3 enhancements that allows to find much more recombining components than GPX1. Increasing linearly the number of recombining components, p, an exponentially larger number of reachable offspring are exploited. The enhancements are:

– **Exploring vertices of degree-4 as possible points for recombination:** GPX1 explores only common edges as possible connection points between

recombining components. In GPX2, a "ghost node" is created for every degree-4 vertex in G_u. The original and ghost nodes are linked by a common edge with weight 0. Thus, by removing the common edges in the new union graph, some vertices of degree-4 in G_u become potential points for recombination. The number of recombining component is further increased by exploring both directions for tour $\mathbf{y}$ and by using an efficient data structure (Extended Edge Table) for storing the direct and reverse tours for parent $\mathbf{y}$;
- **Exploring candidate components with more than two entries:** All candidate components with 2 entries are recombining components. However, not all candidate components with more than 2 entries are recombining components. In order to test the candidate components, simplified graphs are built for the path of each parent inside the candidate component. If the simplified graphs for both parents are equal, then exchanging the paths still results in a Hamiltonian cycle for the offspring. Another test is executed for the case where a recombining component is nested inside a candidate component. If, after removing the already identified recombining components, the number of entries of the candidate component becomes 2, then the candidate component is a recombining component;
- **Fusing candidate components:** Two candidate components that are not individual recombining components can be fused in order to create a recombining component. Two types of fusion are applied in GPX2. In fusion type 1, the fusion occurs between two candidate components that are neighbors. Then, the new candidate component is tested in order to verify if it is a recombining component or not. Cycles of fusion type 1 are repeated n_f times, obtaining each time larger candidate components. In fusion type 2, nested and intercalated candidate components are fused. Then, it is verified if the resulting component has 2 entries after removing the already identified recombining components. The procedure is repeated n_r times.

The time complexity for GPX2 is $O(n)$ [14]. Examples of GPX2 are presented in Figs. 2 and 3. In Fig. 3, IPT finds 3 recombining components, while GPX2 finds 4 recombining components. As a consequence, IPT finds the best of 2^3 reachable offspring, while GPX2 finds the best of 2^4 reachable offspring. The tour found by GPX2 in this example is shorter than the tour found by IPT.

4 Results

In the experiments, LKH with IPT (LKH+IPT) is compared to LKH with GPX2 (LKH+GPX2). The version of LKH used is 2.0.8[1]. Here, LKH runs with the default parameters, except for the number of runs (10 or 50), number of trials (10 or 1000), and population size (equal to the number of runs). It is important to observe that, in LKH, the best results found in different runs are not independent

[1] In LKH version 2.0.8, tours may be recombined by GPX2 instead of IPT. The code for LKH version 2.0.8, that allows to reproduce the results presented in this paper, can be downloaded at http://www.akira.ruc.dk/~keld/research/LKH/.

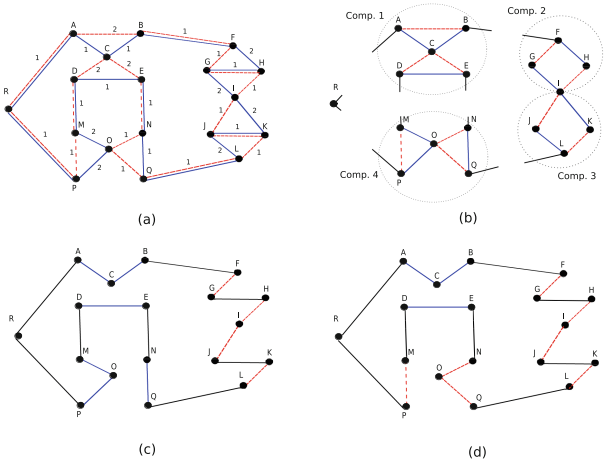

Fig. 3. Examples of recombination by IPT and GPX2. (a) Union graph composed of parents **x** (blue solid line) and **y** (red dashed line). (b) GPX2 identifies 4 recombining components. Components 1 and 4 have 4 entries each. (c) IPT identifies 3 of the recombining components found by GPX2: 2, 3, and union of 1 and 4. (c) The offspring generated by IPT has cost 20. (d) The offspring generated by GPX2 has cost 18. (Color figure online)

(see Fig. 1). When GPX2 is used in LKH, $n_f = 3$ and $n_r = 1000$ (parameters used in fusion).

Experiments with 11 instances of four classes of the symmetric TSP are presented. The 2 instances of Class 1 are artificial instances used in the 8th DIMACS Implementation Challenge [8]. In E31k0, the locations of 31,623 cities are uniformly generated in a square of 1,000,000 by 1,000,000 units. In C10k0, the locations of 10,000 cities consist of clustered points in the same square. LKH currently holds the records for these instances [7]. The records for the remaining problems are reported in [1]. The 3 instances in Class 2 (pia3056, dke3097, and xqe3891) are from the VLSI TSP Collection. The 5 instances in Class 3 (tz6117, ym7663, ar9152, usa13509, usa115475) are formed using (Euclidean) distance between cities of different countries. The size n in the instances of classes 2 and 3 is given in the name of the instances. Finally, monalisa100K is an instance of the Art TSP Collection with $n = 100,000$ vertices.

Due to the limitation of space, we only show the results for the experiments with 1000 trials and 50 runs (Table 2). Table 2 shows the percentage gap to the cost of the best solutions found in the literature [1,7]; when the cost of the best solution found by an algorithm is equal to the best result reported in the literature, the number of runs needed for finding the best result is shown in parenthesis. Smaller results are better for the percentage gap, number of runs, and average running time. A summary of the comparison of the best cost found by the algorithms in all experiments is presented in Table 1.

We also tested two versions of LKH where both operators are used. In the "First IPT and then GPX2" version, GPX2 is applied after IPT. In the "First GPX2 and then IPT" version, IPT is applied after GPX2. For strategy "First IPT and then GPX2", IPT is applied first to recombine the parents. If there is no improvement, GPX2 is then applied to recombine the parents. Otherwise, i.e., IPT generated an improvement, then GPX2 is applied to recombine the first parent with the offspring generated by IPT. The opposite occurs for strategy "First GPX2 and then IPT", i.e., GPX2 is applied first. If an operator A is applied first and does not improve the best solution, but operator B does, this means that operator B found recombination opportunities missed by operator A. The results for those versions for the experiments with 1000 trials are shown in Table 3. Some observations can be made about the results.

In the experiments with versions "First IPT and then GPX2" and "First IPT and then GPX2", GPX2 (applied after IPT) was able to find many recombination opportunities that improved the best solution in all experiments, except for instance ar9152 when the number of trials and runs is 10. For example, in the experiment with "First IPT and then GPX2" version applied to instance usa13509 when the number of trials and runs is 1000, GPX2 improved 713 times the best results (Table 3). IPT improved the best result 2358 times in this case. As IPT was applied first, it is clear that GPX2 found some recombination opportunities missed by IPT in this case. However, IPT (applied after GPX2) was not able to find recombination opportunities that improved the best solution, with exceptions for 2 instances in the experiments with 10 trials and 50 runs, and for 3 instances in the experiments with 1000 trials and 50 runs. The cases where IPT (applied after GPX2) was able to find recombination opportunities that improved the best solution can be explained by two main factors: the limit n_r used in fusion type 2 and the fact that there are different ways of finding the recombining components.

In general, GPX2 found more recombination opportunities that resulted in improvements of the best solution than IPT (see, for example, the number inside the parenthesis in Table 3). More efficient recombination generally results in better performance. When the number of trials and runs is 10, LKH+IPT found better solutions in experiments with 2 instances, while LKH+GPX2 found better solutions in experiments with 9 instances (Table 1). When the number of runs increased (and the number of trials was kept to 10), even better results were obtained by LKH+GPX2: it found better results for 10 out of 11 instances. Increasing the number of runs (from 10 to 50) resulted in more recombinations (Fig. 1); as a consequence, GPX2 found still more recombinations that resulted in improvements for the best solution.

LKH+GPX2 also resulted in better performance for the experiment with 1000 trials and 50 runs: LKH+GPX2 presented better performance in 8 out of 11 instances. However, LKH+IPT found better performance 3 times; for instance pia3056, LKH+IPT was able to find the literature best solution, while LKH+GPX2 was not. Thus, finding more recombination opportunities does not guarantee that LKH will perform better. A better solution obtained by recombination in a trial influences the solutions generated in the subsequent trials and

Table 1. Comparison between LKH+GPX2 and LKH+IPT (regarding the cost of the best solutions found by the algorithms). For each instance and experiment, the algorithm with better performance is indicated.

Problem	Experiment		
	10 trials, 10 runs	10 trials, 50 runs	1000 trials, 50 runs
pia3056	**LKH+GPX2**	**LKH+GPX2**	LKH+IPT
dke3097	**LKH+GPX2**	LKH+IPT	**LKH+GPX2**
xqe3891	**LKH+GPX2**	**LKH+GPX2**	LKH+IPT
tz6117	**LKH+GPX2**	**LKH+GPX2**	**LKH+GPX2**
ym7663	**LKH+GPX2**	**LKH+GPX2**	**LKH+GPX2**
ar9152	LKH+IPT	**LKH+GPX2**	**LKH+GPX2**
C10k0	LKH+IPT	**LKH+GPX2**	LKH+IPT
usa13509	**LKH+GPX2**	**LKH+GPX2**	**LKH+GPX2**
E31k0	**LKH+GPX2**	**LKH+GPX2**	**LKH+GPX2**
monalisa100K	**LKH+GPX2**	**LKH+GPX2**	**LKH+GPX2**
usa115475	**LKH+GPX2**	**LKH+GPX2**	**LKH+GPX2**

Table 2. Best cost gap (to the cost of the best results found in the literature) and average running time (in seconds) for LKH+GPX2 and LKH+IPT. The number of trials is 1000 and the number of runs is 50. The best results are in bold.

Problem	Best cost gap (%)		Average running time (s)	
	LKH+IPT	LKH+GPX2	LKH+IPT	LKH+GPX2
pia3056	**0 (run 5)**	0.0360	**56.01**	63.41
dke3097	0 (run 2)	**0 (run 1)**	**61.67**	75.83
xqe3891	**0 (run 2)**	0 (run 3)	**80.40**	104.65
tz6117	0 (run 29)	**0 (run 6)**	**188.36**	237.77
ym7663	0 (run 22)	**0 (run 4)**	**169.47**	194.24
ar9152	0.0140	**0.0130**	**723.72**	849.78
C10k0	**0 (run 16)**	0 (run 36)	**389.63**	406.23
usa13509	0 (run 34)	**0 (run 13)**	**331.73**	384.12
E31k0	0.0100	**0.0096**	1767.79	**1713.40**
monalisa100K	0.0220	**0.0110**	19901.08	**19061.51**
usa115475	0.0360	**0.0190**	13664.19	**13237.47**

runs. Recall that, in multi-trial LKH, the current best solution is employed to generate the soft restarts and the set of candidate edges. Thus, an initial solution that is not so good can be used by LK in order to generate a promising local optimum. For instances with 10,000 cities or more, LKH+IPT obtained better results for only one instance (C10k0). This is a clustered instance; in clustered

instances, most of the recombining components have two entries when two local optima are recombined. Recall that one of the most important properties of GPX2, when compared to IPT, is that it is able to find recombining components with more than two entries. If there are not many recombining components with more than two entries, e.g., in clustered instances, GPX2 and IPT generally have similar performance.

Despite the better results for the cost of the solutions, LKH+GPX2 generally resulted in higher mean running times (Tables 2 shows the results for the experiments with 1000 trials). For the experiments with 1000 trials, LKH+IPT presented smaller mean time in 8 out of 11 instances (Table 2). LKH+GPX2 presented smaller mean time in the 3 largest instances: E31k0, monalisa100K, and usa115475.

Table 3. Number of times that both crossovers improved the best solution or only the second crossover improved the best solution. The number of trials is 1000 and the number of runs is 50. The results in parenthesis indicate the total number of improvements generated by the first crossover.

Problem	"First IPT, then GPX2"	"First GPX2, then IPT"
	GPX2 (total for IPT)	IPT (total for GPX2)
pia3056	21 (271)	0 (328)
dke3097	44 (342)	0 (383)
xqe3891	68 (361)	0 (417)
tz6117	153 (1256)	0 (1330)
ym7663	221 (1175)	0 (1321)
ar9152	65 (2094)	2 (2003)
C10k0	69 (2277)	0 (2381)
usa13509	713 (2358)	0 (2710)
E31k0	3003 (4036)	0 (5934)
monalisa100K	10997 (3374)	1 (12520)
usa115475	8206 (9981)	12 (13631)

5 Conclusions

Previous versions of multi-trial LKH use only IPT to recombine solutions generated in different trials and runs. Here, we investigated the use of GPX2 in LKH. The experimental results indicated that GPX2 finds some recombination opportunities that are missed by IPT. This impacts the quality of solutions generated by recombination. However, finding better offspring by recombination does not necessarily guarantee better performance for the iterative local search; solutions generated by recombination impacts the soft restarts of multi-trial LKH.

In the experiments with 10 trials and 10 runs, LKH+GPX2 obtained better results for 9 instances, while LKH+IPT obtained better results for 2 instances.

When the number of runs increased to 50 (and the number of trials was kept to 10), LKH+GPX2 obtained even better results: LKH+GPX2 obtained better results for 10 out of 11 instances. Finally, in the experiments with 50 runs and 1000 trials, LKH+GPX2 obtained better results for 8 instances, while LKH+IPT obtained better performance for 3 instances. Despite the better results for the cost of the solutions, LKH+IPT generally resulted in smaller mean running time. For the experiments with 50 runs and 1000 trials, LKH+GPX2 resulted in smaller mean time only for the three largest instances.

As a consequence of this work, solutions may be now recombined by GPX2 instead of IPT in LKH version 2.0.8 [7]. GPX2 can be easily adapted to asymmetric TSP. However, when used with LKH, no modifications are needed; the asymmetric instance is transformed into a symmetric instance twice the size. Optimizing the running time of LKH+GPX2 is a possible future work. There is room for improvement: for example, data structures that explore the implementation of LKH can be proposed. Another future work can be the investigation of LKH+GPX2 in different routing problems. LKH-3 [6] is a recent extension of LKH-2 that can be used in constrained TSP and other vehicle routing problems, e.g., multiple traveling repairman problem and vehicle routing problem with pickups and deliveries.

References

1. Cook, W.: TSP test data (2009). http://www.math.uwaterloo.ca/tsp/data/index.html
2. Cook, W.: In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton University Press, Princeton (2011)
3. Hains, D., Whitley, D., Howe, A.: Revisiting the big valley search space structure in the TSP. J. Oper. Res. Soc. **62**(2), 305–312 (2011)
4. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. J. Oper. Res. **126**(1), 106–130 (2000)
5. Helsgaun, K.: General k-opt submoves for the Lin-Kernighan TSP heuristic. Math. Program. Comput. **1**(2–3), 119–163 (2009)
6. Helsgaun, K.: An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Roskilde University, Technical report (2017)
7. Helsgaun, K.: LKH (2018). http://www.akira.ruc.dk/~keld/research/LKH/
8. Johnson, D., McGeoch, L., Glover, F., Rego, C.: 8th DIMACS implementation challenge: the traveling salesman problem (2013). http://dimacs.rutgers.edu/Challenges/TSP/
9. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. Oper. Res. **21**(2), 498–516 (1973)
10. Möbius, A., Freisleben, B., Merz, P., Schreiber, M.: Combinatorial optimization by iterative partial transcription. Phys. Rev. E **59**(4), 4667–4674 (1999)
11. Ochoa, G., Veerapen, N., Whitley, D., Burke, E.K.: The multi-funnel structure of TSP fitness landscapes: a visual exploration. In: Bonnevay, S., Legrand, P., Monmarché, N., Lutton, E., Schoenauer, M. (eds.) EA 2015. LNCS, vol. 9554, pp. 1–13. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31471-6_1

12. Radcliffe, N., Surry, P.: Fitness variance of formae and performance predictions. In: Whitley, D., Vose, M. (eds.) Foundations of Genetic Algorithms, vol. 3, pp. 51–72. Morgan Kaufmann, Burlington (1995)
13. Tinós, R., Whitley, D., Chicano, F.: Partition crossover for pseudo-Boolean optimization. In: Proceedings of FOGA XIII, pp. 137–149 (2015)
14. Tinós, R., Whitley, D., Ochoa, G.: A new generalized partition crossover for the traveling salesman problem: tunneling between local optima. Submitted to Evolutionary Computation (2018)
15. Tinós, R., Zhao, L., Chicano, F., Whitley, D.: NK hybrid genetic algorithm for clustering. IEEE Trans. Evol. Comput., 13 p. (2018). https://doi.org/10.1109/TEVC.2018.2828643
16. Veerapen, N., Ochoa, G., Tinós, R., Whitley, D.: Tunnelling crossover networks for the asymmetric TSP. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 994–1003. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_93
17. Whitley, D., Hains, D., Howe, A.: Tunneling between optima: partition crossover for the TSP. In: Proceedings of GECCO 2009, pp. 915–922 (2009)
18. Whitley, D., Starkweather, T., Fuquay, D.: Scheduling problems and traveling salesmen: the genetic edge recombination operator. In: Proceedings of ICGA 1989, pp. 133–140 (1989)