Improving an Exact Solver for the Traveling Salesman Problem using Partition Crossover

Danilo Sanches
Department of Computer Science
Federal University of Technology of Paraná
Cornélio Procópio, PR, Brazil
danilosanches@utfpr.edu.br

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

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

ABSTRACT

The best known exact solver for generating provably optimal solutions to the Traveling Salesman Problem (TSP) is the Concorde algorithm. Concorde uses a branch and bound search strategy, as well as cutting planes to reduce the search space. The first step in using Concorde is to obtain a good initial solution. A good solution can be generated using a heuristic solver outside of Concorde, or Concorde can generate its own initial solution using the Chained Lin Kernighan (LK) algorithm. In this paper, we speed up Concorde by improving the initial solutions produced by Chained LK using Partition Crossover. Partition Crossover is a powerful deterministic recombination operator that is able to tunnel between local optima. In every instance we examined, the addition of recombination resulted in an average speed-up of Concorde, and in the majority of cases, the difference in the runtime costs was statistically significant.

CCS CONCEPTS

•**Computing methodologies** →*Heuristic function construction; Discrete space search; Randomized search;*

KEYWORDS

Traveling Salesman Problem; Concorde, branch and cut; recombination

ACM Reference format:
Danilo Sanches, Darrell Whitley, and Renato Tinós. 2017. Improving an Exact Solver for the Traveling Salesman Problem using Partition Crossover. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 9 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071304

1 INTRODUCTION

The Traveling Salesman Problem (TSP) can be defined with respect to a complete weighted graph $G(V,E)$, where every vertex in $V = \{v_1, v_2, ..., v_n\}$ is linked to the other vertices. Each edge $e_{i,j} \in E$ between vertices v_i, $v_j \in V$ is associated with a weight $w_{i,j} \in R^+$. The feasible solutions of the TSP are Hamiltonian circuits in the graph G. The evaluation of a particular solution $x = [x_1, x_2, ..., x_n]^\intercal \in X$, specifying a permutation on $n-1$ vertices, is given by:

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

where the node v_{x_1} denotes the first vertex in the permutation and the edge e_{x_1,x_n} closes the circuit. The objective is to find $x \in X$ with the minimum evaluation $f(x)$. The weight matrix $W = [w_{ij}]$ can be symmetric or asymmetric. The TSP is one of the best studied problems in combinatorial optimization. Many exact and approximate algorithms have been developed over the last 70 years.

The best known of the exact solvers is the Concorde algorithm, which implements a sophisticated form of branch and bound to generate solutions to TSP instances that are provably globally optimal. In addition to generating and exploiting bounds, Concorde also uses cutting planes to narrow and focus the search space. Of course, since the TSP is NP-Hard, the cost of executing an exact solver scales exponentially. Thus, an algorithm such as Concorde will fail to execute in a reasonable time for sufficiently large problems instances. While we cannot solve this scaling problem in general, it is still desirable to improve the expected running time of branch and bound algorithms. A significant improvement in the running time can allow the algorithm to be applied to somewhat larger problem instances.

The running time of branch and bound algorithms is affected by the availability of a good initial solution. Classic branch and bound algorithms uses this initial solution to eliminate parts of the search space that provably cannot yield a better solution. The initial solution establishes an aspiration level, and provides an initial upper bound on the global optimum. As better bounds are discovered, it becomes possible to prune away more of the search space. In principle, a better initial solution should result in a improvement in runtime.

When run in its default configuration, Concorde uses the Chained Lin Kernighan (Chained LK) algorithm to find the initial solution [2]. The Chained LK algorithm is a form of iterated local search. When Lin Kernaghan is run in its non-iterated form, it is reasonable to assume that the solutions found on each run are locally optimal under the 3-opt operator. Thus, we will imprecisely refer to the solutions as "local optima." This usage is imprecise because Lin Kernighan uses a variable k-opt neighborhood, and therefore the notion of a local optimum may not be precisely defined. After

GECCO '17, Berlin, Germany

978-1-4503-4920-8/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3071178.3071304

Chained LK fails to find an improving move after some time, it uses a soft restart in the form of a double bridge move that perturbs the current solution. From this perturbed solution, the Lin Kernaghan algorithm is applied iteratively until the search stagnates.

If an algorithm such as Chained LK uses 30 soft restarts, then it also generates a set of 30 "locally optimal" solutions that are not easily improved by further local search. We can treat these solutions as an emergent population and apply recombination to these solutions with the goal of finding a better initial solution for Concorde. To be a viable strategy for improving Concorde, the recombination operator must have very low cost, and it must have a very high probability of finding an improved solution. In this work, we propose the use of Partition Crossover to speed up Concorde by recombining the set of "local optima" generated by Chained LK.

Partition crossover is a recombination operator for the TSP that produces offspring from a graph constructed from the union of two parent solutions. Different versions of partition crossover have been introduced: Partition Crossover [18], Generalized Partition Crossover (GPX) [19] and GPX2 [17]. It has been shown that these partition crossover operators have a high probability of producing locally optimal offspring if the parents are local optima [18]. The cost of applying partition crossover is also very low: it has $O(n)$ complexity with a very small constant. Tinós [17] introduced enhancements to GPX2 that allow it to find many more recombining partitions than previous versions of Partition Crossover. If Partition Crossover generates q recombination partitions, it is guaranteed to return the best of 2^q possible offspring. Theoretical and empirical results show that the set of 2^q possible offspring are largely composed of local optima (e.g., under 3 opt). Thus, Partition Crossover operators can quickly generate very high quality solutions.

We present a set of experiments that shows how GPX2 can be used to improve the runtime behavior of Concorde. When run on a set of 22 benchmark instances using dedicated machines, Concorde using the GPX2 crossover operator reduces the average running time in every single case, and in approximately half of the cases the difference in the running time is statistically significant.

Our results also reveal that the running time of Concorde is very unpredictable. Concorde is not only sensitive to the *evaluation* of the initial solution, it is also sensitive to the *structure* of the initial solution. It is also sensitive to machine resources such as memory and memory allocation. We also present preliminary results on shared machines, and include additional enhancements using the Lin-Kernighan-Helsgaun (LKH) algorithm and GPX2 to improve initial solutions for Concorde.

Section 2 covers the basics of the Concorde algorithm. Section 3 reviews partition crossover operators and recent enhancements to the operator. Section 4 presents experiments aiming at speeding up Concorde using dedicate machines. Section 4 also present experimental results using both LKH and partition crossover to speed up Concorde using shared machines. Conclusions appear in Section 5.

2 THE BASICS OF CONCORDE

The Traveling Salesman Problem can be posed as a pseudo-Boolean optimization problem with constraints. Given n vertices, a Hamiltonian circuit can be defined as a bit vector with length $n(n-1)/2$, with exactly n bits with value 1. Bits with value 1 selects n edges with the goal of finding a minimal Hamiltonian circuit. Note that $n(n-1)/2$ is also the number of edges in the cost matrix for the TSP, and the cost matrix can also be represented as a cost vector denoted by c. Let the set of solutions that are also Hamiltonian circuits be denoted by S.

Thus a solution can be defined by

$$\text{minimize } c^T x \text{ subject to } x \in S \tag{2}$$

To be a Hamiltonian circuit, the set of n selected edges are also such that the number of edges that are incident on every vertex is exactly 2. To convert the TSP into a linear programming problem, we instead need a problem of the form

$$\text{minimize } c^T x \text{ subject to } Ax \leq b \tag{3}$$

for some appropriate system $Ax \leq b$ of linear inequalities. Equation 3 is a relaxation of the problem posed in equation 2. If a solution is found that satisfies both equation 2 and 3, then it is an optimal solution to the TSP. However, if a solution is found for equation 3 that is not a solution to equation 2, we still obtain a new lower bound on the optimal solution, and we can add additional constraints (e.g., cutting planes) to try to move the search space "closer" to the optimal solution.

Concorde, Version 03.12.19 [3], is the best performing exact algorithm currently known for the symmetric TSP. To solve the linear programming problems that arise during the sophisticated branch and cut search process, Concorde uses an LP solver specifically designed for this purpose.

As is typical when using branch and bound (or branch and cut) algorithms, Concorde can use a good initial solution to generate constraints on the search space.

Concorde can be run in two modes; in one mode, Concorde generates its own initial solution using the Chained LK algorithm, which is a form of Iterated (i.e., "chained") Local Search. Lin Kernighan (LK) [9], is a powerful and well-known heuristic for solving the TSP. For about two decades, it was the best local search method, and it is a key component of state-of-the-art TSP solvers [12] based on local search. LK search explored k-exchanges to find improving moves, i.e, take the current tour and remove k different links from it, which are then reconnected in a new way to achieve a legal tour. LK applies 2, 3 and higher-order k-exchanges. The overall strategy used by LK-search repeatedly starts the basic LK routine from different starting points keeping the best solution found. Martin et al. [10] proposed the idea of perturbing the LK tour and reapplying the algorithm. This new strategy was named Chained LK [2]. The perturbation operator in Chained LK is a type of 4-exchange, known as a double-bridge move [10].

Chained LK applies LK-search to a single tour and then runs until it 1) is trapped by a local optimum, or 2) the search stagnates because it cannot find an improving move. Chained-LK then iteratively reapplies LK local search. The Chained-LK implementation in the Concorde software package [3] also uses "do not look bits" and candidate lists. The "do not look" bits exploit the fact that some moves which are not improving moves do not need to be checked again (hence, "do not look") until there is an event that changes the potential contribution of that move. Candidate lists are used to heuristically restrict neighborhood size. In particular, candidate

lists focus attention on the shortest edges that are incident on a particular vertex.

Chained LK has the desirable property that it can generate good solutions with minimal runtime cost. Nevertheless, when generating a good initial solution, there is a trade-off between the quality of the initial solution and the amount of time that is spent searching for a good initial solution before starting the branch and cut algorithm.

The other way in which Concorde can be used is to confirm an optimal solution generated by another method. Concorde can be initialized with a solution that is thought to be optimal or near optimal that was found by a heuristic solver using a significant amount of computation. When used in this mode, Concorde does not generate its own initial solution. In principle, Concorde's running time should be the smallest when the initial solution is actually a globally optimal solution, since this allows Concorde to prune away more of the search space as fast as possible. For very large TSP instances where the global optimum is known, the optimal solution was usually found by a heuristic solver, and then the optimal solution was confirmed using an exact solver such as Concorde.

In this paper, we improve the performance of Concorde by using genetic recombination to find a better initial solution to pass into the branch and bound algorithm. We will first do this by improving on the set of local optima (or approximate local optima) that are already found by the Chained LK local search algorithm. We capture the "local optima" visited just before every soft restart. We then recombine all of these local optima to seek an additional improvement before passing the initial start solution to the branch and bound algorithm.

Other researchers have also looked at ways of improving the runtime cost of Concorde [1, 8]. Hougardy [8] presents a graph theoretic result that makes it possible to prove that select edges of a TSP instance cannot occur in any globally optimal TSP tour. A combinatorial algorithm was developed to identify and eliminate these edges during the branching process.

3 PARTITION CROSSOVER OPERATORS

Two desirable properties of recombination operators that were identified by Radcliffe et al. [13] are the ability to be 1) respectful and 2) to transmit "alleles." What this means in practical terms for the TSP is that 1) all edges shared in common by the parents are inherited and 2) all edges are inherited from one of the two parents. It should be noted, however, that such recombination operators are highly exploitive and can be very powerful, but they are not particular good at exploration. This is because operators for the TSP that only transmit edges cannot introduce new edges in the population.

Möbius et al. [11] introduced the first recombination operator for the TSP that always transmitted edges. The operator is called "Iterated Partial Transcription" (IPT). The IPT operator is not well known in the evolutionary computation community, but IPT was included as an operator in the very well known Lin-Kernighan-Helsgaun (LKH) local search algorithm for the TSP. In effect, IPT is also a from of partition crossover.

Whitley et al. [18, 19] independently introduced a "partition crossover" operator for the TSP that is respectful and transmits edges. There have been multiple enhancements to partition crossover for the TSP as well as a detailed theoretical analysis [6, 17–19]; there are no theoretical or analytical results for the IPT operator. IPT is more powerful that the original Partition Crossover (PX) operator, but less powerful than GPX2. All partition crossover operators define a *reachable set* of offspring, and then return the best of all of these offspring. One partition crossover is more powerful than another if the reachable set of offspring is (exponentially) larger.

Partition Crossover works by locally decomposing the parents (which are Hamiltonian circuits) into subgraphs [16]. To be useful for partition crossover, the resulting subgraphs must also decompose the evaluation function into linearly separable subfunctions. We refer to subgraphs that also decompose the evaluation function into linearly separable subfunctions as *recombining components*. We also want to maximize the number of recombining components. If the number of recombining components is q, then the number of reachable offspring is of size $2^q - 2$, assuming the set of offspring does not contain the parents. It has been shown that partition crossover operators have a high probability of producing locally optimal offspring if the parents are local optima [18, 19]. The offspring is always locally optimal in the largest hyperplane subspace that contains both parents [17].

All partition crossover operators first construct a graph $G_u = (V, E_u)$ from two solutions, S_{P1} and S_{P2} , where V is the set of cities in the TSP instance and E_u is the union of edges in the two solutions (see Fig. 1(a)). To partition G_u, the edges in G_u that are shared by both solutions are removed to create a graph G'_u (see Fig. 1(b)). Breadth first search is then applied to G'_u to locate the connected subgraphs. Sometimes the connected subgraphs are not recombining components. Degenerate cases must be recognized to determine when crossover is (or is not) feasible. GPX2 is able to "fuse" connected subgraphs and can determine is the fusion of two connected subgraphs can be used for partition crossover.

In the simplest case, a recombining component has one entry and exit (e.g., vertices 6 and 12 in Fig. 1(b)). In general, if a graph is broken into two subgraphs, and one subgraph is a recombining components, then both subgraphs must be recombining components. This idea can be applied recursively to subgraphs. In more complex cases (e.g. Fig. 2) a recombining component can have multiple entries and exits. Under partition crossover the recombining components are found in $O(n)$ time.

In the original description of IPT crossover operator by Möbius et al. [11] the implementation of IPT had $O(n^2)$ complexity. This is due to the fact that the IPT operator explicitly searched for subtours that include the same subset of cities. Checking all possible subtours has $O(n^2)$ costs. However, we now know we do not need to check all possible subtours. There are two major advantage of partition crossover operators over IPT: 1) all of the partition crossover operators were designed to have $O(n)$ runtime complexity to achieve fast execution times, and 2) we can find more recombining components with multiple entries and exits and (exponentially) increase the number of reachable offspring.

3.1 Partition Crossover: GPX2

Fig. 1 illustrates how recombination exploits partitions of the graph G'_u. Let the red (dashed) lines represent parent one and blue (solid)

lines represent parent two. The first observation is that a partition that cuts only 2 common edges always produces 2 recombining components. This creates a single entry and exit from each subgraph. Given one entry and one exit, a tour can be broken at these "cuts" and recombination will always yield new distinct and feasible offspring [18]. If there are multiple partitions that cut 2 common edges, this process can be repeated.

Tinós et al. [17] present two enhancements to increase the number of recombining components found by the partition crossover, GPX2. First, GPX2 exploits cutting points that break nodes of degree 4 of the union graph G_u as sites for crossover. (IPT also finds single entry, single exit subtours with breaks at vertices of degree 4, but does so at higher computational cost.) Second, GPX2 identifies recombining partitions with any even number of entries and exits. It does this by first decomposing the graph G_u, and checking to determine if the two parents enter and exit the subgraph at the same vertices.

If the subgraphs are not recombining components, GPX2 also can merge neighboring connected subgraphs that do not result in feasible recombinations. Two unfeasible subgraphs are neighbors if they are linked by at least two paths composed only of common edges. If the fusion of two connected partitions result in a recombining component, the offspring can inherits the partial solution inside the merged partitions from only one parent. The fusion procedure can be repeated for a constant number of times. The implementation of GPX2 is described in Algorithm 1.

Fig. 2 illustrates an example of the GPX2 operator. Each vertex v of degree 4 is split into vertices v and v' , with a zero cost on the edge between them. We will denote v' as a ghost vertex. The edge between v and the ghost vertex v' is a common edge and therefore removed. In the simplest case, a connected subgraph has a single entry and exit point, then recombination can occur and will yield a new Hamiltonian circuit. In more complex cases, there are multiple entries and exits.

GPX2 generates offspring in the following way. 1) Pick a direction to traverse the parents and then split vertices of degree 4 by inserting ghost vertices after the original vertex; denote the new parents by G'_1 and G'_2. 2) Generate graph my merging the parents so that $G_u = G'_1 \cup G'_2$. 3) Delete the common edges from G_u to create G'_u and identify the connected subgraphs; copy the common edges to the offspring. 4) Determine if the connected subgraphs are feasible recombining components. 5) Copy the edges from the shortest subpath for each recombining component.

These steps must be repeated for 2 difference traversal of the parents. If both parents are traversed from left to right, then a second traversal must be done where one parent is traversed from left to right, and the other parent is traversed from right to left. This idea can be simplified into a traversal in the *same direction* and a traversal in the *opposite direction*. This is because, for example, a recombining component might span from vertice v_α to v_ω in both parents, or a recombining component might span from span from vertice v_α to v_ω in parent 1, but span from vertice v_ω to v_{alpha} in parent 2. Only by scanning parents it the same direction and then in opposite directions can this be detected. This is particularly important because how vertices are split is also determined by the direction of the scan. (The IPT operator must also scan in both directions to locate recombining components, for exactly the same reason.)

Algorithm 1 GPX2

1: **Step 1:** Pick a direction in which to trace each parent. Insert a "ghost vertex" immediately after each vertex with degree 4 in both Parents. The common edge between the original vertex and the corresponding ghost vertex has weight 0.
2: **Step 2:** Create a union graph G_u.
3: **Step 3:** Remove all common edges of G_u to create G'_u. This includes removing the common "ghost edges" that result from splitting vertices of degree 4.
4: **Step 4:** Use Breadth First Search to find the connected components of G_u.
5: **Step 5:** Determine if each connected component is a recombining component. A subgraph is a recombining component if it has a equal number of entry and exit vertices, and both parents enter and exit the subgraph at the same vertices.
6: **Step 6:** If two adjacent connected components are not feasible recombining components, fuse the two adjacent connected components, then determine if the merger results in a new recombining component. Repeat the fusing process for a constant number of times.
7: **Step 7:** Apply crossover by finding the shortest path inside each recombining component.

GPX2 can also identify recombining components with multiple entries and exists [15, 17]. All of the vertices internal to a recombining component must be reachable by traversing single edges (not common edges). Common edges that connect the disconnected subgraphs are potential entries or exits. A recombining component is one where both tours enter at the same vertices, both tours visit all of the vertices of the subgraph, and both tours exit at the same vertices. In Figure 2, there are four recombining components, and two of the recombining components have two entry points and two exit points, and thus four edges in total that connect to other recombining components. Finally, GPX2 will merge connected subgraphs that are not recombining components to determine if the merger will result in a recombining component. To maintain $O(n)$ complexity, GPX2 uses only a constant number of mergers when looking for this type of recombining component.

3.2 Chained LK and Partition Crossover

The modifications we made to Chained LK are very simple. After every soft restart we captured the local optimum (or approximate local optimum) that Chained LK has found just before the restart. If m restarts occur, we construct a population of $m + 1$ solutions: the m solutions found just before the restarts and the final solution. We then apply GPX2 to all of these solutions in a pairwise fashion. This yields very rapid improvement at very low cost.

We also introduced a specific runtime limit on the Chained LK algorithm inside of Concorde; this runtime limit is denoted by T. By forcing Chained LK to run a specific amount of time, we also ensured that Chained LK would generate enough local optima to

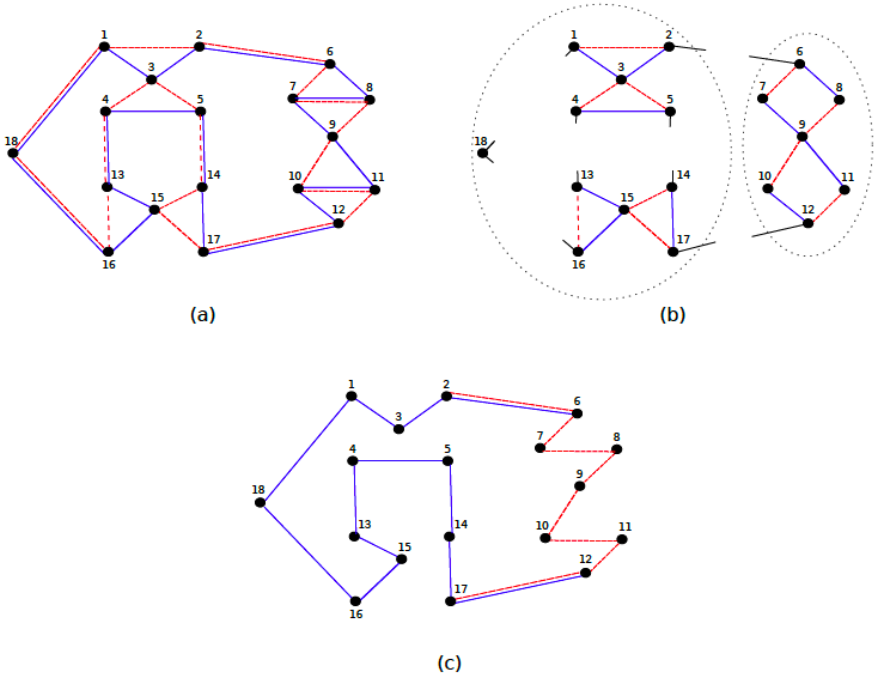

Figure 1: An example of a simple application of Partition Crossover. (a) The graph is constructed from the union of two solutions, S_{P1} (dashed edges) and S_{P2} (solid blue edges). (b) The removal of shared edges creates connected subgraphs; this makes it easy to identifies two recombining components. (c) The best offspring is constructed from the shared edges in (a) and the dashed or solid edges from the subgraphs in (b).

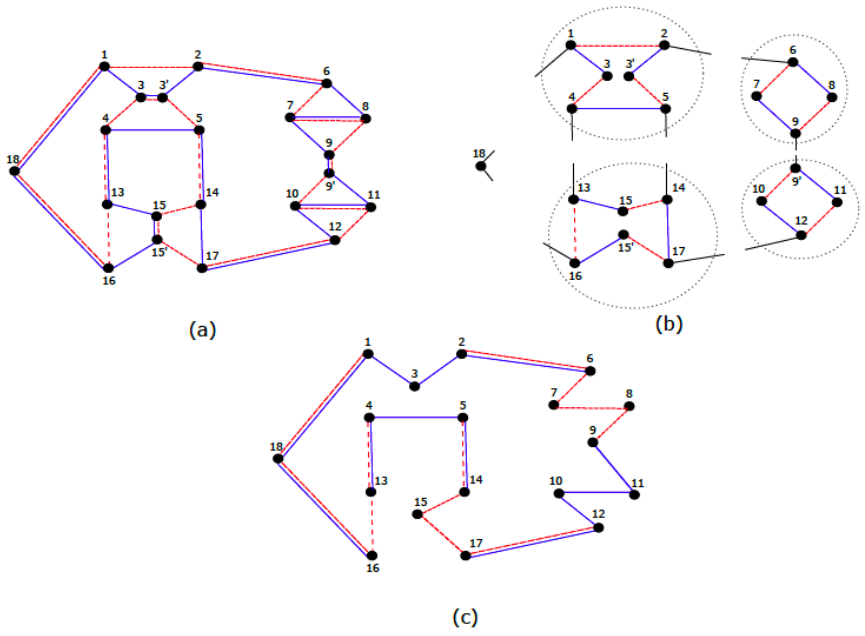

Figure 2: The parents are the same as in Figure 1. GPX2 is able to identify more recombining components with multiple entries and exits. a) Parent solutions after ghost vertices are inserted after all vertices of degree 4. b) After removing the common edges, the connected subgraphs are obvious. In this case, there exists *recombining components* that have a total of 4 entry and exit vertices. c) An offspring using the four recombining components. GPX2 returns the best of $2^4 - 2 = 14$ possible offspring.

make partition crossover viable. The addition of GP2 to the Chained LK algorithm adds much less than 1 second to the running time.

This strategy is also similar to "tour merging via branch decomposition" proposed by Cook and Seymour [4], but it appears to have much lower cost. In tour merging a form of LK (Chained LK or LKH) is run multiple times. Then the resulting set of solutions are merged after being decomposed using branch decomposition. "Tour merging" is really just a form of multi-parent crossover that transmits edges, and it is unclear if it is any more effective than partition crossover operators such as GPX2.

4 EXPERIMENTS

We ran experiments using a number of larger TSP instances drawn from TSPLIB [14], as well as the National and VLSI instances [5]. The National instances are based on the locations of cities within countries, while the VLSI instances stem from applications in VLSI circuit design. The three National instances range in size from 980 to 4,663 cities while the twelve VLSI instances have 1,376 to 2,086 vertices. Most of these instances are known to be particularly hard for Concorde. We also used larger instances from the DIMACs test set, specifically instances E1k.0, E3k.0 and C3k.0.

4.1 Results with Dedicated CPU's

In this section we look at how the addition of the GPX2 partition crossover operator impacts the running time of the Concorde algorithm. One might assume that, in theory, we can never do worse that the Concorde algorithm. However, the Concorde algorithm makes a number of stochastic search decisions and we have found that the running time of Concorde can be highly variable. Factors that impact the running time of Concorde include: 1) the random seed, 2) the amount of available computer memory, 3) the evaluation of the initial solutions, and 4) the sequence of the initial solution. Surprisingly, reversing the order of the permutation representing the initial solution can dramatically change the running time.

This first set of experiments were carried out with dedicated machines. With no other processes running on these machines, we can control for the effects of memory access on the running time.

The steps of the proposed approach are described in algorithm 2. In the first set of experiments the "Iterated LK algorithm" refers to Chained LK. In later experiments, the "Iterated LK algorithm" will be the Lin Kernighan Helsguan (LKH) algorithm.

The running time T allocated to Chained LK in the experiments was based on the running time of Concorde with standard settings. For that reason, we considered $T = 5$ seconds for datasets where Concorde normally spends less than 20,000 seconds. Otherwise, we have set $T = 30$. Also, Concorde's standard parameters were set to their normal default settings. In every case, we ran 30 experiments (with random seeds 1 to 30).

In Table 1 we present results using dedicated machines, meaning that Concorde was the only programming running on these machines. We used 8 identical HP xw6600 workstations, each equipped with quad-core 3.0 GHz Intel Xeon E5450 processors and 32GB main memory. All experiments were run using Fedora Linux, version 24 and compiled with gcc-6.2.1 with optimization flag settings -O3. Running times were measured based on CPU times reported by Concorde.

Algorithm 2 Proposed Approach

```
Set number of experiments S = 30;
for ¡seed = 1; seed < S; seed + +¿ do
    Set Running time in seconds T;
    //Create a vector K;
    K = [];
    while (runningTime < T) do
        Run the Iterated LK algorithm;
        Save all of the Iterated LK "local optima" in K;
    end while

    Save the best tour from K in Best.Tour;

    //Apply GPX2, recombining all of the "local optima"
    //in vector K to find an Improved Tour

    Call GPX2(K) and generate Improved.Tour;

    //Run the Concorde solver

    Run Concorde with -s seed and Best.Tour
    Run Concorde with -s seed and Improved.Tour;
end for
```

In every single case, GPX2 was able to improve the initial solution. It is reasonable to assume that an improved initial solution should result in faster runtime on average, and indeed this is the case. On average, in every single case, Concorde using GPX2 partition crossover displayed faster running times compared to the standard Concorde configuration. In three cases, the difference was less than 10%. But across all problems, Concorde using GPX2 partition crossover displayed 15% faster running times compared to the standard Concorde configuration. In 9 of 22 cases, the differences are statistically significant ($p < 0.05$) using a two-tailed t-test. If we had used a one-tail t-test (based on the fact that adding crossover can only improve the initial solution), then 12 of the 22 cases are significant ($p < 0.05$).

Our results suggest that Concorde should always be run using the addition of GPX2 partition crossover. In no case was the mean runtime of Concorde using GPX2 worse that the runtime of Concorde using the standard setting.

4.2 LKH and Results on Shared CPU's

In a second set of experiments, we asked what happens if we use the Lin Kernighan Helsgaun (LKH) algorithm instead of Chained LK to find the initial solution, again improving the LKH using partition crossover.

We did these experiments in a shared machine environment. This is partly because using dedicated machines is costly; for example, the experiments on just instance ca4663 took 4163.6 CPU hours to execute. Using dedicated machines requires blocking the access of other users from machines. Our other motivation is therefore related: some users will likely run Concorde on shared machines. Still, from a scientific point of view this is not ideal, and some results will be biased by the load on the machines. We can still capture

Table 1: Concorde with Standard settings and CLK+GPX2 results using dedicated machines. Under "t-test" the * symbol indicates the difference is significant at the $p \leq 0.05$ level using a two-tailed t-test. The ~ symbol indicates the difference would be significant if a one-tailed t-test were used.

	Concorde Standard		CLK+GPX2		Runtime		
Dataset	Mean	Std. Dev.	Mean	Std. Dev.	Ratio	t-test	P-value
u1432.tsp	284.10	316.56	48.14	19.53	5.90	*	0.0001
dcb2086.tsp	820.98	343.86	617.13	181.78	1.33	*	0.0057
bva2144.tsp	944.66	283.03	841.98	237.77	1.12	=	0.1336
fl1577.tsp	1312.53	467.66	1200.23	437.41	1.09	=	0.3408
dkd1973.tsp	2642.42	1032.99	2580.10	1218.63	1.02	=	0.8316
nu3496.tsp	4221.73	2263.91	3137.26	1312.03	1.34	*	0.0269
dcc1911.tsp	4617.14	1974.82	3863.69	1188.07	1.20	~	0.0786
d1291.tsp	5763.42	1626.28	4566.53	1055.38	1.26	*	0.0013
dcj1785.tsp	7715.09	2539.39	6672.38	2720.34	1.16	=	0.1303
fnl4461.tsp	42149.59	17816.86	36021.43	14048.87	1.17	=	0.1445
ca4663.tsp	108332.00	50080.18	76388.80	41209.83	1.42	*	0.0091
pr2392.tsp	16.10	2.77	14.39	1.90	1.12	*	0.0072
rl1304.tsp	60.97	20.24	50.31	16.96	1.21	*	0.031
nrw1379.tsp	60.88	12.05	44.66	8.86	1.36	*	0.0001
E1k.0	81.45	21.26	72.82	18.17	1.12	~	0.0964
C3k.0	620.92	263.09	557.43	179.21	1.11	=	0.2792
rl1323.tsp	846.51	290.43	766.08	258.31	1.10	=	0.2617
rl1889.tsp	1917.10	597.17	1664.17	506.18	1.15	~	0.082
u2152.tsp	19841.96	5588.40	17819.83	4511.55	1.11	=	0.1285
fl3795.tsp	21855.47	11735.01	19339.86	10658.49	1.13	=	0.3883
E3k.0	24999.61	5531.54	24217.73	5823.30	1.03	=	0.5959
u1817.tsp	152499.08	73609.24	112017.67	35790.79	1.36	*	0.0089

Table 2: Average running time in seconds of the Concorde solver with: standard settings; CLK+GPX2, LKH2+IPT and LKH2+GPX2 initial solutions. These results were generate on shared machines. The symbol † indicates the results are significantly different than Standard Concorde using the Wilcoxon statistic. The symbol * indicates the results are significantly different than Standard Concorde using a t-test.

Problem	Concorde Standard mean+std	CLK+GPX2 mean+std	LKH2+IPT mean+std	LKH2+GPX2 mean+std	Runtime Ratio
u1432	377.8 ± 430.6	284.0 ± 183.7	73.7*† ± 43.3	**63.6*† ± 31.5**	5.98
dcb2086	860.1 ± 360.9	816.5 ± 289.5	**693.0* ± 314.1**	710.4 † ± 362.1	1.21
bva2144	1003.8 ± 321.0	961.3 ± 298.7	841.2*† ± 230.4	**820.2* ± 202.4**	1.22
fl1577	1702.8 ± 630.5	1692.0 ± 676.4	1683.0 ± 595.5	**1423.9 ± 481.1**	1.19
dkd1973	2667.9 ± 1051.2	2567.7 ± 987.3	2636.2 ± 1082.3	**2559.6 ± 813.9**	1.04
nu3496	4297.8 ± 2341.32	3437.9 ± 1846.9	3253.5 † ± 1439.7	**3084.7*† ± 1719.9**	1.39
dcc1911	5234.9 ± 2429.7	4138.1* ± 2288.8	**3139.7*† ± 1389.2**	3520.5*† ± 1302.3	1.48
dcj1785	8126.1 ± 2684.0	7981.8 ± 3899.1	6776.4* ± 2852.1	**6493.2*† ± 2794.6**	1.25
d1291	10949.1 ± 3358.8	11143.9 ± 4317.4	9965.8* ± 4413.8	**9948.2† ± 3495.8**	1.10
djb2036	29012.6 ± 24413.3	31078.7 ± 22748.5	28099.7 ± 14727.9	**27840.5 ± 12402.4**	1.04
u2319	55269.1 ± 85546.9	47662.6 ± 37491.9	222.6*† ± 393.6	**141.6*† ± 168.3**	391.9
fnl4461	57961.4 ± 24728.1	52577.5 ± 17073.9	49337.7 ± 19990.8	**43546.7*† ± 11061.3**	1.33
ca4663	140660.8 ± 64290.3	119425.5 ± 45742.4	112668.6 ± 48133.8	**111908.9*† ± 35192.24**	1.25

the actual running time, but the running time can be affected by memory usage and swapping between processes.

We again used HP xw6600 workstations equipped with quad-core 3.0 GHz Intel Xeon E5450 processors and 16GB main memory. All experiments were run using Fedora Linux, version 24 and compiled with gcc-6.2.1 with optimization flag settings -O3. Running times were the CPU times reported by Concorde.

Lin-Kernighan-Helsgaun 2 (LKH-2) [7] is considered to be the state-of-the-art local search heuristic for the TSP based on the variable depth local search of Lin and Kernighan (LK-search) [9]. In LKH-2, a number of improvements have been added to the original LK heuristic, like the use of sensitivity analysis to estimate the probability of an edge appearing in an optimal solution. LKH-2 is also a form of iterated local search which uses soft restarts. However, the kick operator used by LKH-2 is more disruptive than the double bridge move used by Chained LK.

We ran the same experiment outlined in **Algorithm 2** again, except now in a shared machine environment. This again gave us baseline results for Concorde, and then results with an Improved Tour by using GPX2.

In this second set of experiments we also replaced Chained LK with LKH2. Thus, we ran **Algorithm 2** again, but this time with LKH-2 as the Iterated LK algorithm. This gave us two more sets of results, one baseline using LKH2 and then an Improved Tour generated using GPX2. In total, this gives us four combinations: 1) Standard Concorde with Chained LK, 2) Chained LK with GPX2 (CLK+GPX2), 3) LKH-2, and 4) LKH-2 with GPX2 (LKH-2+GPX2). The experiments were executed 30 times, each time with a different random seed for CLK, LKH-2 and Concorde. For each execution, LKH-2 and CLK were ran for a specific duration of time T. In most cases, $T = 5$ seconds. On the most difficult problems (with running time more than 20,000 seconds), $T = 30$ seconds. After saving the best "local optima" from each restart of CLK and LKH-2, the GPX2 recombination operator is applied to search for additional improvements. In this way, we do not lose the best solution found by CLK and LKH-2, and the initial solution generated by GPX2 can only be better. Note that LKH already includes the IPT recombination operator. Thus, improvements found by GPX2 represent additional improvements that were missed by IPT.

Table 2 contains the running times used by Concorde to solve instances generated by the proposed approaches. The best running-time found are highlighted in bold font. We ran two forms of statistical tests: we ran both the Mann-Whitney-Wilcoxon test and a t-test. These two statistics make different assumptions and are sensitive to different properties of the data, but the outcomes match in most cases. When testing the null hypothesis that the runtime for standard Concorde and LKH2+GPX2 are drawn from the same population, both statistics indicate a significant difference (p-valve < 0.05) in 7 out of 13 experiments. In every case, the mean runtime is less for LKH2+GPX2 compared to standard Concorde.

5 CONCLUSIONS

In this paper the GPX2 partition crossover operator was used to improve the initial solution used by the Concorde branch and cut algorithm. This improvement was generate by capturing different "local optima" visited by iterated local search, and then recombining the set of local optima to find a better initial solution. The GPX2 partition crossover operator is ideally suited to this task, since it finds an improvement with high probability and at low cost. Generating an improved initial solution should result in a faster running time, and on average this proved to be true when experiments are run on dedicated machines.

We also replaced the Chained LK algorithm with the LKH-2 iterated local search algorithm, and then additionally improved the solution found by LKH-2 using the GPX2 partition crossover operator. Again, this improved the performance of Concorde. Using recombination to improve the initial solution generated by Chained LK (or LKH-2) is likely to have the most benefit on very large TSP instances where the running time of Concorde can grow very large.

The running time of the Concorde algorithm can be highly variable, as indicated by the standard deviations associated with the running times. Clearly, there are significant stochastic elements to the runtime behavior of Concorde. It is important to ask if there are ways to reduce the runtime variance of Concorde. But this is an extremely complex question given the complexity of Concorde.

6 ACKNOWLEDGMENTS

R. Tinós was supported by FAPESP (under grant 2015/06462-1) and CNPq (under grant 304400/2014-9). D. Sanches was supported by CNPq (under grant 304400/2014-9) (under grant number 458598/2014-3). D. Whitley was supported by AFOSR (under grant FA9550-11-1-0088).

REFERENCES

[1] F. Ahammed and P. Moscato. Evolving l-systems as an intelligent design approach to find classes of difficult-to-solve traveling salesman problem instances. In *Applications of Evolutionary Computation: EvoApplications*, pages 1–11. Springer, 2011.
[2] D. Applegate, W. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS J. on Computing*, 15(1):82–92, Jan. 2003.
[3] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *Concorde-03.12.19*, 2003. http://www.math.uwaterloo.ca/tsp/data/index.html.
[4] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.
[5] W. J. Cook. *TSP test data*, accessed October 24, 2016. http://www.math.uwaterloo.ca/tsp/data/index.html.
[6] D. Hains, D. Whitley, and A. Howe. Improving lin-kernighan-helsgaun with crossover on clustered instances of the TSP. In *Parallel Problem Solving from Nature*, PPSN'12, pages 388–397, 2012. Springer.
[7] K. Helsgaun. General k-opt submoves for the lin-kernighan tsp heuristic. *Mathematical Programming Computation*, 1(2):119–163, 2009.
[8] S. Hougardy and R. T. Schroeder. Edge elimination in tsp instances. In D. Kratsch and I. Todinca, editors, *Graph-Theoretic Concepts in Computer Science*, pages 275–286. Springer, 2014.
[9] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, 21(2):498–516, Apr. 1973.
[10] O. Martin, S. Otto, and E. Felten. Large-step markov chains for the tsp incorporating local search heuristics. *Operations Research Letters*, 11:219 – 224, 1992.
[11] A. Möbius, B. Freisleben, P. Merz, and M. Schreiber. Combinatorial optimization by iterative partial transcription. *Physycal Review E*, 59(4):4667–4674, 1999.
[12] G. Ochoa and N. Veerapen. Additional dimensions to the study of funnels in combinatorial landscapes. In *Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 373–380, 2016. ACM.
[13] N. Radcliffe and P. Surry. Fitness variance of formae and performance predictions. In D. Whitley and M. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 51–72. Morgan Kaufmann, 1995.
[14] G. Reinelt. *TSPLIB 95*, accessed October 24, 2016. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.
[15] D. Sanches, D. Whitley, and R. Tinós. Building a better heuristic for the traveling salesman problem: Combining edge assembly crossover and partition crossover. In *Genetic and Evolutionary Computation Conference GECCO*, 2017. ACM.
[16] R. Tinos, D. Whitley, and F. Chicano. Partition crossover for pseudo-boolean optimization. In *Foundations of genetic algorithms, (FOGA-15)*, pages 137–149, 2015. ACM.
[17] R. Tinós, D. Whitley, and G. Ochoa. Generalized asymmetric partition crossover (GAPX) for the asymmetric tsp. In *Genetic and Evolutionary Computation Conference*, GECCO '14, pages 501–508, 2014. ACM.
[18] D. Whitley, D. Hains, and A. Howe. Tunneling between optima: Partition crossover for the traveling salesman problem. In *Genetic and Evolutionary Computation Conference*, GECCO '09, pages 915–922, 2009. ACM.
[19] D. Whitley, D. Hains, and A. Howe. A hybrid genetic algorithm for the traveling salesman problem using generalized partition crossover. In *Parallel Problem Solving from Nature*, PPSN'10, pages 566–575, 2010. Springer.