Building a Better Heuristic for the Traveling Salesman Problem: Combining Edge Assembly Crossover and 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

A genetic algorithm using Edge Assemble Crossover (EAX) is one of the best heuristic solvers for large instances of the Traveling Salesman Problem. We propose using Partition Crossover to recombine solutions produced by EAX. Partition Crossover is a powerful deterministic recombination that is highly exploitive. When Partition Crossover decomposes two parents into q recombining components, partition crossover returns the best of 2^q reachable offspring. If two parents are locally optimal, all of the offspring are also locally optimal in a hyperplane subspace that contains the two parents. One disadvantage of Partition Crossover, however, is that it cannot generate new edges. By contrast, the EAX operator is highly explorative; it not only inherits edges from parents, it also introduces new edges into the population of a genetic algorithm. Using both EAX and Partition Crossover together produces better performance, with improved exploitation and exploration.

CCS CONCEPTS

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

KEYWORDS

Traveling Salesman Problem; Recombination Operators; Partition Crossover; Edge Assembly Crossover; Combinatorial Optimization

ACM Reference format:

Danilo Sanches, Darrell Whitley, and Renato Tinós. 2017. Building a Better Heuristic for the Traveling Salesman Problem: Combining Edge Assembly Crossover and Partition Crossover. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017*, 8 pages.
DOI: <http://dx.doi.org/10.1145/3071178.3071305>

1 INTRODUCTION

Since it was formally described more than 70 years ago, the Traveling Salesman Problem (TSP) is one of the most studied problems in

optimization. Applications of the TSP can be found in many diverse areas such as bioinformatics, circuit layout and logistics [1].

An instance of the TSP is defined by a complete weighted graph $G(V, E)$, where $V = \{v_1, v_2, \dots, v_n\}$ is a set of n vertices and E contains edges between every pair of vertices in V . Each edge of E is associated with a weight $w_{i,j} \in \mathbb{R}^+$. The objective in the TSP is to find the Hamiltonian cycle with minimal evaluation:

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

where $\mathbf{x} = [x_1, x_2, \dots, x_n]^T \in X$ specifies a permutation on the vertices of $G(V, E)$. If the weight matrix $W = [w_{ij}]$ is symmetric, the problem is denoted as a *Symmetric TSP*; otherwise, the problem is denoted as an *Asymmetric TSP*.

Some exact methods are capable of solving the TSP with hundreds of vertices in seconds [1]. However, the TSP is NP-Hard and, as a consequence, exact methods result in prohibitive computational times for large instances. For this reason, several heuristic search methods have been proposed for solving large TSP instances. One of the most successful heuristics for TSPs is the Lin-Kernighan-Heuristics (LKH) heuristic [4], that holds the record for several large instances of the TSP [2]. LKH incorporates a number of different mechanisms to the variable depth local search heuristic developed by Lin and Kernighan (LK). A crossover operator, called Iterative Partial Transcription [5], is also used to recombine local optima generated in different trials of the LKH heuristic.

Another very efficient heuristic for the TSP is the genetic algorithm with Edge Assemble Crossover (EAX), that also holds the record for the best known solution to some very large instances of the TSP [8]. The EAX was introduced by Nagata and Kobayashi in 1997 as a way of inheriting many edges from parents, while at the same time also introducing new edges of high quality [10]. The name “EAX” is sometimes informally used to refer to both the EAX operator and the genetic algorithm that uses EAX [9, 11]. Nagata and Kobayashi [11] provide empirical evidence that EAX often, but not always, outperforms LKH on several sets of commonly studied Euclidean TSP instances in terms of the solution qualities reached within similar or shorter running times. The EAX operator is described in Section 3.

As individual solvers, LKH and EAX represent the state of the art for solving large TSP instances. Mu et al. [6] use automated algorithm configuration methods to improve both LKH and EAX, matching particular TSP instances to particular algorithm configurations. In this paper, we combine a form of LK local search and EAX, and add a Partition Crossover operator to EAX.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GECCO '17, Berlin, Germany

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. 978-1-4503-4920-8/17/07...\$15.00

DOI: <http://dx.doi.org/10.1145/3071178.3071305>

A crossover for the TSP is both 1) respectful and 2) transmits edges if: 1) all edges shared by the parents are inherited by the offspring; 2) offspring are composed only by edges that are present in the parents. EAX is very efficient, but it is not respectful and does not transmit all edges. Whitley *et al.* [18] proposed a deterministic crossover operator, named Partition Crossover, that is both respectful and transmits genes. Partition Crossover works by finding the connected components in the graph created by the removal of the shared edges in the graph formed by taking the union of the edges found in the two parents. Some of the connected components identified in the resulting graph are *recombining components*. *Recombining components* can be deterministically identified and recombined to produce a new Hamiltonian circuit. If q recombining components are found, Partition Crossover returns the best of 2^q reachable offspring at cost $O(n)$. An interesting consequence of the decomposition of the parents is that, given two parents that are locally optimal, all of the offspring are also locally optimal in a hyperplane subspace that contains the two parents. A form of Partition Crossover called GPX2 is presented in Section 2.

Partition Crossover is highly exploitive; it finds the best of an exponential number of local optima at cost $O(n)$. However, Partition Crossover operators are not good at exploration. When Partition Crossover is used as part of a genetic algorithm, a method is still needed to introduce diversity because Partition Crossover operators can never generate new edges. By contrast, the EAX operator is highly explorative; it not only inherits edges from parents, it also introduces new edges into the population of a genetic algorithm. Therefore, the EAX operator and Partition Crossover operators complement each other and should be used together. In particular, Partition Crossover can be used to find improvements when EAX is not able to generate an improvement.

The algorithms proposed in Section 4 are the first algorithms to use both EAX and Partition Crossover. Using both EAX and Partition Crossover together produces better performance, with improved exploitation and exploration (Section 5).

2 GENERALIZED PX AND GPX2

Generalized Partition Crossover 2 (GPX2) is an efficient deterministic recombination operator proposed by Tinós *et al.* [16] for the Symmetric TSP. GPX2 incorporates three enhancements to the *Generalized Partition Crossover* (GPX), developed by Whitley *et al.* [18]. Two of the three enhancements of GPX2 were first proposed in a version of GPX for the Asymmetric TSP, called Generalized Asymmetric Partition Crossover [15]. Variations of Partition Crossover have also been developed for pseudo-Boolean optimization problems [14] and clustering [13].

The first step of Partition Crossover for the TSP is to create the union graph $G_u = G_1 \cup G_2$ from graphs G_1 and G_2 representing the parent solutions. Examples of union graphs are presented in figures 1 and 2. Common edges, i.e., edges shared by both parents, are then removed from G_u to create G'_u and connected components are identified by Breadth First Search.

We formally define a **recombining component** as a subgraph of G'_u where: 1) a recombining component contains z vertices, with $2x$ vertices that connect to a different recombining component; the remaining $z - 2x$ vertices connect to vertices inside the recombining

Algorithm 1 Recombination by Partition Crossover

- 1: Create a union graph $G_u = G_1 \cup G_2$ from parents G_1 and G_2 .
 - 2: Split nodes of degree 4, and set of the weight of the common connecting edge to zero.
 - 3: Remove all common edges of G_u to create G'_u .
 - 4: Use Breadth First Search to find the connected components of G'_u .
 - 5: Determine if each connected component is a recombining component (it will have the same entry and exit points for both parents).
 - 6: Compute, for both parents, the length of the tour inside each recombining partition.
 - 7: To create the offspring, select the tour (from one or another parent) with the minimum length for each recombining partition.
-

component; 2) a recombining component has x vertices with external connections that are entry points, and x vertices with external connections that are exit points; 3) the two parent solutions enter the recombining component and exit the recombining component in the same way.

In the simplest case there is one entry and one exit. But, as shown in Figure 2, there can also be multiple entry and exit points. (We will show that some of these are found by splitting vertices of degree 4.) If a graph is broken into k subgraphs, and $k - 1$ of the subgraphs are recombining components, then the remaining k^{th} subgraph must also be a recombining component.

The subpaths inside of a recombining component represent partial solutions that can be independently evaluated. In effect the recombining components linearly decompose the evaluation function. Because we are interested in the best offspring, the paths inside the components are recombined in a greedy way by selecting the shortest path inside each recombining component. The cost of Partition Crossover, including the steps for the identification of the recombining components, is $O(n)$. Algorithm 1 shows the pseudo-code of the GPX2 Partition Crossover operator for the TSP.

GPX2 includes three enhancements that allow it to find additional recombining partitions. The first enhancement is exploiting cuts that break nodes of degree 4 in the union graph as possible sites for crossover. This is possible by splitting every vertex v of degree 4 in order to create a “ghost” vertex v' . The edge between v and v' has zero cost and is shared for both parents. Like all common edges, the common edge between v and v' is a candidate site for crossover. Figure 1 illustrates the recombination by GPX2 in an example with vertices of degree 4. Iterative Partial Transcription, the recombination operator included in the LKH algorithm [3], can also find recombination opportunities at vertices of degree 4, but in the worst case IPT has $O(n^2)$ complexity.

The second enhancement of GPX2 is identifying recombining components with more than one entry and exit connecting the recombining component to the rest of the graph. To aid in the discovery of entries and exits, it is helpful to assume that the parents (i.e., the Hamiltonian circuits) are directed and therefore can be traversed in a particular direction. It then becomes simple to identify which vertex is an entry and which vertex is an exit. For example in Figure 2 there is a recombining component with an

Figure 1: Illustration of recombination by GPX. The bold black line separates the recombining components. GPX finds the best of $2^4 = 16$ offspring, or 14 offspring excluding the parents. All of the edges are inherited from only one parent in each recombining component.

Figure 2: When GPX2 is applied in this example, the union graph is partitioned into 4 recombining components. Instead of just deleting common edges, GPX also “splits” vertices of degree 4 to create a crossover point. The bold black line separates the recombining components. Two of the recombining components have 2 entry points (an entry and an exit). However, two of the recombining components have 4 entry points which are identified by GPX2.

entry point at vertex 11 and an exit point at vertex 16, followed by another recombination component with an entry point at vertex 16 and an exit point at vertex 1. (What is an “entry” or an “exit” changes if the Hamiltonian circuit is traversed in the opposite direction.) Checking if the respective entries and exits for each parent are equal makes it possible to identify recombining components with any even number of connecting points. Figure 2 also presents examples of recombining components with 4 connecting points. For example one of the recombining components has entries/exits at vertices 3, 4, 8 and 10.

A third enhancement of GPX2 is a procedure that merges two unfeasible connected subgraphs, i.e., two connected subgraphs that cannot be used as recombining components. When two unfeasible subgraphs are neighbors, they can be merged in order to create a potential recombining component. Unfeasible subgraphs are neighbors if they are connected by at least two common edges. If the

fusion of two unfeasible subgraphs results in a recombining component, the offspring inherits the path inside the merged partitions from only one parent. The fusion procedure can be recursively applied for a constant number of times, generating each time larger candidate subgraphs. All the three enhancements proposed for GPX2 have $O(n)$ cost.

3 EDGE ASSEMBLY CROSSOVER: EAX

The EAX crossover operator combines two parents into a graph, which we again denote by G_u . It then finds what are known as *AB-Cycles* in the graph G_u . An AB-Cycle is found by taking one edge from parent A, then one edge from parent B, then one edge from parent A again, etc. This continues until a loop results composed of alternating edges: a-b-a-b-a-b ... a-b where “a” denotes any edge from Parent A and “b” denotes any edge from Parent B. The loop of alternating edges is an AB-Cycle. Every graph G_u can be completely decomposed into AB-Cycles; however, some of the AB-Cycles can be degenerate in the sense that an AB-cycle might only contain two vertices. A number of AB-Cycles are generated and together are referred to as the *E-Set*. The AB-Cycles in the E-Set can then be used to cut one of the parents into subcircuits.

The basic steps used by the EAX recombination operator are illustrated in Figure 3. Parent 1 is cut using the AB-Cycles shown. This cutting process means that edges found both in Parent 1 and the AB-Cycles are removed. Then edges from Parent 2 found in the AB-Cycles are added to Parent 1. This process yields the subcircuits. The subcircuits are then merged in a greedy fashion to create a offspring. This greedy process which merges the subcircuits into a Hamiltonian circuit also introduces relatively short edges that are not found in either parent. Thus, EAX has a way to introduce new (and highly useful) diversity into the population [10, 17].

Another novel aspect of the EAX operator is the use of “brood selection.” When using brood selection two parents generate many offspring, and then return the best of all offspring that are generated. This is possible when using the EAX operator because there are many ways to generate different AB-cycles. Brood selection is used by EAX as a mechanism to ensure that EAX returns an improving move as often as possible, and this is critical to the performance of EAX [17]. The EAX implementation is described in Algorithm 2 (see [7, 8, 10, 11] for more details).

The EAX genetic algorithm uses 2-opt local search to determine the initial population, as well as a specific tabu search procedure for generating offspring from very high-quality parent solutions. Furthermore, an entropy-based mechanism is used to preserve diversity in the population of candidate solutions. A rather complex combination of termination criteria is used to determine when a run of EAX is ended, at which point the best tour encountered during the run is returned.

4 PROPOSED ALGORITHMS

We will use GPX2 to recombine solutions generated by EAX. The Edge Assembly Crossover (EAX) operator and the GPX2 crossover operator are complementary in many ways. EAX is highly explorative, and GPX2 is highly exploitive. When GPX2 can decompose 2 parents into many recombining components, it is capable of rapid improvement. The disadvantage of Partition Crossover operators

Figure 3: An illustration of the EAX recombination operator. The top 3 graphs represent the two parents and the union of the two parents into one graph. AB-Cycles are extracted from the graph which is the Union of the Parents. The AB-Cycles are then used to cut Parent 1 into subcircuits. These subcircuits are then reconnected in a greedy fashion to create an offspring. Note that the offspring is composed of edges from Parent 1, edges from Parent 2, and completely new edges not found in either parent.

is that they never introduce new edges into the population. Thus, a genetic algorithm must use some other mechanism to introduce new and useful edges into the population. Introducing new and useful edges is something that EAX does very well.

Preliminary experiments indicated that using Partition Crossover too frequently cause very rapid improvement in the quality of the TSP solutions found by the GA using both EAX and GPX2, but the combination also quickly reduced diversity in the population. In effect, EAX could not generate new edges fast enough to overcome the exploitive nature of GPX2. Therefore, it was necessary to limit the use of GPX2 in order to maintain the longer term search potential of the genetic algorithm using EAX.

In addition, to speed up the early stages of search, we also use both Chained Lin-Kernighan (CLK) and 2-opt search to improve the initial population of solutions passed to genetic algorithm using EAX. This must also be done in a way that does not destroy the

diversity in the population. While the EAX recombination operator is able to introduce new edges into the population, the genetic algorithm (and EAX) is still very sensitive to population diversity. Because we are introducing several changes to the genetic algorithm using EAX, we have made these changes to the EAX genetic algorithm incrementally, and we test performance after making each incremental change. Three proposed algorithms are compared to the standard EAX. The standard EAX optimizes solutions initially generated by 2-opt search using the recombination operator EAX.

The pseudo-code for the standard genetic algorithm using EAX is presented in Algorithm 3. We refer to this as the Standard EAX. One detail not discussed in the pseudocode is selection. In effect, the EAX GA does not use traditional selection mechanisms. Instead, the order of the population is first randomized. After randomization of the order of the population, the solution (tour) at location i in the population (Parent 1) is recombined with the solution (tour) at

Algorithm 2 Recombination by EAX

- 1: Denote a pair of parents as tour-A and tour-B, and define G_u as a graph constructed by merging tour-A and tour-B.
- 2: Divide the edges on G_u into *AB-cycles*, where an *AB-cycle* is defined as a closed loop on G_u that can be generated by alternately tracing the edges of tour-A and tour-B.
- 3: Construct an *E-set* by selecting *AB-cycles* according to a given rule.
- 4: Generate an intermediate solution by applying the *E-set* to tour-A, i.e., by removing tour-A's edges in the *E-set* from tour-A and adding tour-B's edges in the *E-set* to it.
- 5: Modify the intermediate solution to generate a valid tour by connecting its sub-tours. Two sub-tours are connected by deleting one edge from each sub-tour and adding two edges to connect them. Which sub-tours are connected and which edges are deleted are determined heuristically.

Algorithm 3 Standard EAX

- 1: Apply 2-opt local search to $popSize$ random solutions to generate the initial population.
- 2: **while** (stagnation $\neq$ stopCondition) **do**
- 3: Generate the new population applying EAX to the current population.
- 4: **end while**

location $i+1$ (Parent 2). The offspring, produced by EAX using brood selection, replaces the solution at location i (Parent 1). Selective pressure comes from the use of brood selection. (See Nagata's papers [7, 8, 10] for more details.)

The following parameter settings are used in all of the experiments and algorithms. The number of children generated by EAX is denoted by N_{ch} where $N_{ch} = 50$. The stopping condition associated with stagnation is 100 generations without an improvement. Three population sizes are used: $popSize = 200$, $popSize = 400$ and $popSize = 800$.

4.1 EAX+CLK

In the first modification, called *EAX+CLK*, the initial population is generated in the same way as in the standard EAX algorithm, using 2-opt to improve the population. Before applying EAX, CLK is executed to optimize $N_{clk} = 200$ random solutions. The best $M = 20$ solutions generated by CLK are stored in the initial population. The initial population is then completed by $popSize - M$ randomly generated solutions optimized by 2-opt. The pseudo-code of *EAX+CLK* is described in Algorithm 4.

4.2 EAX+CLK+GPX

In *EAX+CLK+GPX*, the initial population is again generated by random solutions optimized by CLK and 2-opt search. The initial population is first seeded using M random solutions optimized by CLK. The M solutions generated by CLK are selected from N_{clk} trials of CLK.

GPX2 is applied in two steps: First, the GPX2 is used to optimize the best M solutions generated by CLK. This means that all M solutions are recombined using GPX2, and the best offspring over

Algorithm 4 EAX+CLK

- 1: Apply CLK to N_{clk} random solutions and store the best M solutions in the initial population.
- 2: Apply 2-opt local search in $popSize - M$ random solutions to complete the initial population.
- 3: **while** (stagnation $\neq$ stopCondition) **do**
- 4: Generate the new population applying EAX to the current population.
- 5: **end while**

Algorithm 5 EAX+CLK+GPX

- 1: Apply CLK in $popSize$ random solutions and store the best $M < popSize$ solutions in the initial population $[x_1, \dots, x_M]$.
- 2: Apply GPX2 in the best M CLK solutions and store the new solution in x_{M+1} .
- 3: Apply 2-opt local search in $popSize - M$ random solutions to complete the initial population $([x_{M+2}, \dots, x_{popSize}])$.
- 4: **while** (stagnation $\neq$ stopCondition) **do**
- 5: **if** (generation $\% k$ or $j == 5$) **then**
- 6: $r(\cdot)$ = a random permutation of $1, \dots, popSize$.
- 7: $i = 1$.
- 8: **while** ($i \leq popSize$) **do**
- 9: $p_A = x_{r(i)}, p_B = x_{r(i+1)}$.
- 10: $[c_1, \dots, c_{N_{ch}}] = \text{EAX}(p_A, p_B)$.
- 11: $x_{r(i)} = \text{GPX2}([c_1, \dots, c_{N_{ch}}])$.
- 12: **end while**
- 13: **end if**
- 14: **end while**

all of the recombination events is saved. This new solution found using GPX2 is stored in the initial population along with the M best solutions generated by CLK. The remainder of the population is then generated randomly and improved using 2-opt.

Second, the GPX2 operator is used to recombine the offspring solutions generated by EAX. We do not apply GPX2 directly to parents in the population. Instead we further improve the offspring found by EAX by recombining the $N_{ch} = 50$ offspring generated by EAX. This greatly increases the changes of finding an improved solution.

Even using GPX2 in this limited way, GPX2 is very aggressive at finding improvements and it still tends to reduce population diversity more quickly than EAX alone. Therefore, in this version of the algorithm, we only applied GPX2 in this way for the first $k = 5$ generations and then, after this, we used GPX2 to improve the offspring of EAX only after EAX alone has not found an improving move for $j = 5$ generations. This works because GPX2 is often able to find new improvements after EAX becomes stuck. Note that the problem of becoming "stuck" and unable to find a new improving move is also related to population size. We explore this trade-off in the next variation of the algorithm. The pseudo-code of *EAX+CLK+GPX* is described in Algorithm 5.

4.3 EAX+CLK+GPXall

The third approach, *EAX+CLK+GPXall*, has exactly the same steps as the *EAX+CLK+GPX* algorithm. The difference is that GPX2

is applied in all generations every time that EAX is called (see Algorithm 6). The idea behind this variation is that while using GPX2 does tend to reduce diversity, diversity can also be increased by using a larger population (increasing the popSize). This also seems to help more on the larger more difficult problems.

Algorithm 6 EAX+CLK+GPXall

```

1: Apply CLK to  $N_{clk}$  random solutions and store the best  $M$ 
   solutions in the initial population  $[x_1, \dots, x_M]$ .
2: Apply GPX2 to the best  $M$  CLK solutions solution and store
   the new solution in  $x_{M+1}$ .
3: Apply 2-opt local search in  $N - M$  random solutions to complete
   the initial population  $([x_{M+2}, \dots, x_{popSize}])$ .
4: while (stagnation != stopCondition) do
5:    $r(.)$  = a random permutation of  $1, \dots, popSize$ .
6:    $i = 1$ .
7:   while ( $i \leq popSize$ ) do
8:      $p_A = x_{r(i)}, p_B = x_{r(i+1)}$ .
9:      $[c_1, \dots, c_{N_{ch}}] = \text{EAX}(p_A, p_B)$ .
10:     $x_{r(i)} = \text{GPX2}([c_1, \dots, c_{N_{ch}}])$ .
11:   end while
12: end while
  
```

5 RESULTS

In this section, we investigate the performance of the proposed modification to the genetic algorithm that uses EAX in combination with GPX2. We compare results to the standard Nagata and Kobayashi implementation of the EAX genetic algorithm.

For all experiments in this paper we have used six datasets from well-known instances taken from TSPLIB [12] and one dataset (ja9847) from National instances [2]. Also, the algorithms were run using eight identical HP workstation xw6600, each 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.

Tables 1, 2, 3, 4 summarize the results from all experimental trials. According to the complexity of the dataset used, we have ran thirty experiments for those datasets considered easy to solve, namely rl5934, rl5915 and pla7397 and ten experiments for more complex datasets, namely ja9847, rl11849, usa13509 and d15112.

We have also analyzed performance in terms of fitness evaluation and running time. Figures 4 and 5 summarize the performance of the algorithms considering one trial (same seed) for pla7397 dataset with 200 and 800 population size, respectively. In both cases, EAX+CLK+GPX and EAX+CLK+GPXall exhibit better performance than standard EAX and EAX+CLK. The diversity of the initial population has an important role in the efficiency of EAX, but often it is difficult for EAX alone to find the optimal solution. On the other hand, when the GPX2 operator is included, the genetic algorithm is more often able to find the optimal solution (see Figure 5 EAX+CLK+GPXall).

Figure 4: Performance of the proposed approaches and standard EAX considering population size 200.

Figure 5: Performance of the proposed approaches and standard EAX considering population size 800.

5.1 Discussion

While GPX2 yields rapid improvement, EAX is a slow and methodical search operator. Being slow and methodical also has advantages. Using brood selection, EAX tries many different ways of breaking

Table 1: Standard EAX

Dataset	popSize	Fitness		Generations		Running Time (sec.)		Number Opt. Sol.
		Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev	
rl5934	200	556090.8	50.27	463.70	19.86	1433.5	34.54	12/30
rl5915	200	565537.57	29.25	412.57	17.75	1221.37	30.06	23/30
rl11849	200	923297.7	8.07	894.90	14.38	8400.9	130.38	1/10
d15112	400	1573093.6	6.90	1282.40	4.06	47749.5	233.44	0/10
ja9847	400	491936.5	5.17	985.30	7.44	17728.2	233.52	0/10
pla7397	400	23262246.16	1044.31	460.90	7.54	6330	95.05	0/30
usa13509	400	19983321.5	491.30	1319.17	3.82	37999.5	237.11	0/10
ja9847	800	491930.1	2.23	1006.70	29.96	37906.8	618.61	0/10
pla7397	800	23261065.6	552.58	454.50	17.60	12627.2	344.49	2/10
usa13509	800	19983194.5	411.99	1396.30	3.53	81689.7	1355.13	0/10

Table 2: EAX+CLK

Dataset	popSize	Fitness		Generations		Running Time (sec.)		Number Opt. Sol.
		Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev	
rl5934	200	556109.47	294.00	372.90	39.70	691.10	49.14	22/30
rl5915	200	565547.80	45.03	380.90	34.97	616.20	26.69	21/30
rl11849	200	923297.2	12.76	828.90	21.11	5444.3	64.93	4/10
d15112	400	1573099.4	9.58	1191.10	4.38	35258.7	248.88	0/10
ja9847	400	491937	4.65	906.57	5.86	11313.28	123.18	0/10
pla7397	400	23262816.86	1331.37	397.53	24.16	3976.76	491.78	1/30
usa13509	400	19983298.33	218.13	1188.67	4.16	25107.66	178.89	0/10
ja9847	800	491928.3	2.41	904.90	10.41	22959.8	303.74	1/10
pla7397	800	23261650.7	907.53	431.60	16.20	8537.4	186.45	1/10
usa13509	800	19983057.375	88.02	1195.25	4.71	50916.125	279.41	0/10

Table 3: EAX+CLK+GPX

Dataset	popSize	Fitness		Generations		Running Time (sec.)		Number Opt. Sol.
		Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev	
rl5934	200	556062.83	35.27	341.33	35.42	1113.53	103.10	18/30
rl5915	200	565534.83	11.61	342.10	31.46	1018.60	89.24	23/30
rl11849	200	923295.7	11.38	772.60	33.35	6952.3	275.72	4/10
d15112	400	1573091.2	7.44	1301.50	30.58	46036.3	539.20	3/10
ja9847	400	491932.7	3.09	926.40	16.64	15784.9	195.39	0/10
pla7397	400	23261471.86	930.56	436.87	25.65	6069.56	323.21	2/30
usa13509	400	19983062.4	230.61	1250.00	21.09	33641	363.76	1/10
ja9847	800	491927.6	3.44	922.50	19.19	32256.3	416.45	3/10
pla7397	800	23260844.5	221.48	418.00	34.69	11566.3	837.75	4/10
usa13509	800	19982979.1	145.39	1237.86	13.30	67498.9	1009.19	1/10

up parents into little pieces, and then reassembling the pieces to construct new offspring. It is the methodical tearing apart and greedy reassembling of these solution fragments that gives EAX its exploratory power. But it is a relatively slow process.

Partition Crossover operators also have another property that needs more theoretical investigation. If GPX2 generated 30 recombining components, it returns the best of 1 billion offspring (2^{30}). It

also creates an *ordering* over the offspring since they are all constructed from the same sample of edges. The best possible offspring in effect dominates all of the other offspring. If one of the parents is better than all of the offspring, the other parent must be worse than all of the offspring.

Currently, a genetic algorithm using both EAX and GPX2 is still very sensitive to premature convergence. If the population loses enough diversity, even EAX fails to do its job. When recombining

Table 4: EAX+CLK+GPXall

Dataset	popSize	Fitness		Generations		Running Time (sec.)		Number Opt. Sol.
		Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev	
rl5934	200	556058.63	33.66	350.57	37.47	1562.73	248.60	21/30
rl5915	200	565537.77	21.70	357.93	38.60	1022.13	73.00	19/30
rl11849	200	923294.8	8.38	832.70	19.72	7484.2	105.49	4/10
d15112	400	1573090.6	4.09	1379.80	30.11	46767.5	363.55	0/10
ja9847	400	491930.1	4.15	902.80	21.08	15466.2	295.69	1/10
pla7397	400	23262037.66	1184.15	452.93	46.48	6409.73	512.79	2/30
usa13509	400	19983064.6	185.41	1262.80	16.75	33208.2	250.47	0/10
ja9847	800	491926.33	2.50	890.78	18.77	30881.55	263.14	4/10
pla7397	800	23260855	222.36	414.00	51.05	11647.3	1235.05	4/10
usa13509	800	19982987.6	173.88	1230.29	23.65	66849.14	818.98	2/10

two parents that are very similar, EAX is also not able to break the parents apart in such a way to explore new offspring in an effective manner. This is also the reason why the genetic algorithm using EAX requires the use of diversity preserving mechanisms.

Thus, one can see in the results presented in Table 3 and Table 4 that increasing the frequency with which the GPX2 recombination operator is applied can sometimes help (particular if the population is larger and can support more diversity), but it can also hurt by generating improvements too rapidly and thus reducing diversity. Therefore, as has been seen many times with genetic algorithms, the trade-off between aggressive improvement and preserving diversity is important. Achieving the right balance between exploitation and preserving diversity is not easy.

6 CONCLUSIONS

We proposed a new way of combining Edge Assembly Crossover and the GPX2 Partition Crossover operator to generate solutions to large and difficult TSP instances. Partition Crossover is a powerful deterministic recombination operator that is highly exploitive. When Partition Crossover decomposes two parents into q recombining components, partition crossover returns the best of 2^q reachable offspring. By contrast, the EAX operator is highly explorative; it not only inherits edges from parents, it also introduces new edges into the population of a genetic algorithm. In order to minimize negative effects on population diversity, GPX2 is only used to improve the set of set of offspring that are generated by EAX using brood selection. This greatly increases the probability that an offspring will be generated that represents an improvement over the parents. Our empirical results demonstrates that using both EAX and GPX2 together can improve performance and results in a better overall heuristic search algorithm for the TSP.

7 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] W. Cook. *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton Univ. Press, 2011.
- [2] W. Cook. TSP test data, 2013. <http://www.math.uwaterloo.ca/tsp/data/index.html>. Last Updated: September 2013.
- [3] D. Hains, D. Whitley, and A. Howe. Improving Lin-Kernighan-Helsgaun with crossover on clustered instances of the TSP. In *Proc. of PPSN XII*, pages 388–397. Springer, 2012.
- [4] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Oper. Res.*, 126(1):106–130, 2000.
- [5] A. Möbius, B. Freisleben, P. Merz, and M. Schreiber. Combinatorial optimization by iterative partial transcription. *Physica E*, 59(4):4667–4674, 1999.
- [6] Z. Mu, H. Hoos, and T. Strötzel. The impact of automated algorithm configuration on the scaling behaviour of state-of-the-art inexact tsp solvers. In *Learning and Intelligent Optimization, LION 10*, pages 157–172. Springer, 2016.
- [7] Y. Nagata. The EAX algorithm considering diversity loss. *Parallel Problem Solving from Nature - PPSN*, pages 332–341, 2004. Springer.
- [8] Y. Nagata. Fast eax algorithm considering population diversity for traveling salesman problems. *Evolutionary Computation in Combinatorial Optimization: 6th European Conference, EvoCOP*, pages 171–182, 2006. Springer.
- [9] Y. Nagata. New EAX crossover for large tsp instances. *Parallel Problem Solving from Nature - PPSN IX*, pages 372–381, 2006. Springer.
- [10] Y. Nagata and S. Kobayashi. Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA'97)*. San Francisco, CA, 1997. Morgan Kaufmann.
- [11] Y. Nagata and S. Kobayashi. A powerful genetic algorithms using edge assemble crossover the traveling salesman problem. *INFORMS Journal on Computing*, 25(2):346–363, 2013.
- [12] G. Reinelt. *TSPLIB 95*, accessed October 24, 2016. <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/>.
- [13] R. Tinós, Z. Liang, F. Chicano, and D. Whitley. A new evaluation function for clustering: The NK internal validation criterion. In *Proceedings of the 2016 Genetic and Evolutionary Computation Conference, GECCO '16*, pages 509–516, 2016.
- [14] R. Tinós, D. Whitley, and F. Chicano. Partition crossover for pseudo-boolean optimization. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII, FOGA '15*, pages 137–149, 2015.
- [15] R. Tinós, D. Whitley, and G. Ochoa. Generalized asymmetric partition crossover for the traveling salesman problem. In *Proceedings of the 2014 Genetic and Evolutionary Computation Conference, GECCO '14*, pages 501–508, 2014.
- [16] R. Tinós, D. Whitley, and G. Ochoa. A new generalized partition crossover for the traveling salesman problem: tunneling between local optima. *Submitted to Evolutionary Computation*, 2017.
- [17] J. Watson, C. Ross, V. Eisele, J. Denton, J. Bins, C. Guerra, D. Whitley, and A. Howe. The traveling salesrep problem, edge assembly crossover, and 2-opt. In *International Conference on Parallel Problem Solving from Nature*, pages 823–832. Springer, 1998.
- [18] D. Whitley, D. Hains, and A. Howe. Tunneling between optima: Partition crossover for the traveling salesman problem. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO '09*, pages 915–922, New York, NY, USA, 2009. ACM.