Dramatically Faster Partition Crossover for the Traveling Salesman Problem

Ozéas Quevedo de Carvalho
Darrell Whitley
{ozeasx,darrell.whitley}@gmail.com
Department of Computer Science, Colorado State University
Fort Collins, Colorado, USA

Abstract

The Partition Crossover is a deterministic crossover operator for the Traveling Salesman Problem (TSP). It decomposes the union graph of two TSP solutions, A and B, into connected components known as AB-cycles, from which the lower-cost edges are selected and recombined to produce offspring. The operator finds the best offspring within a search space of 2^k solutions in linear time, where k is the number of recombining components. We introduce Generalized Partition Crossover 3 (GPX3), a new implementation of Partition Crossover. GPX3 features a new algorithm to quickly find AB-cycles in the union graph. It also identifies additional recombining AB-cycles, expanding the reachable search space. We show that GPX3 runs in $O(n)$ time and is more efficient and effective than previous implementations of Partition Crossover for the TSP.

CCS Concepts

• **Computing methodologies** → **Genetic algorithms**; • **Mathematics of computing** → **Combinatorial optimization**; *Matchings and factors*; *Graph algorithms*; *Paths and connectivity problems.*

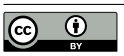
Keywords

Traveling Salesman Problem, Combinatorial Optimization, Genetic Algorithms, Crossover Operators

ACM Reference Format:
Ozéas Quevedo de Carvalho and Darrell Whitley. 2025. Dramatically Faster Partition Crossover for the Traveling Salesman Problem. In *Genetic and Evolutionary Computation Conference (GECCO '25), July 14–18, 2025, Malaga, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3712256.3726465

1 Introduction

The traveling salesman problem (TSP) is one of the most studied problems in combinatorial optimization. With many applications in circuit design, network optimization, logistics, and gene sequencing, it has been extensively studied by the AI, Operations Research, and metaheuristics communities [4]. Its formulation is as simple as it is deceptive: given a set of cities and the distances between them, find the shortest round trip that visits each city only once. Despite its apparent simplicity, the TSP is NP-hard, and its decision version belongs to the class of NP-complete problems [12].



GECCO '25, July 14–18, 2025, Malaga, Spain

ACM ISBN 979-8-4007-1465-8/2025/07
https://doi.org/10.1145/3712256.3726465

We can model the TSP as a connected graph, where the vertices represent the cities, and the edges represent the roads connecting them. Consider a complete graph $T = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is a set of n vertices representing the cities, and $E = \{(i, j) \mid i, j \in V, i \neq j\}$ is a set of edges connecting the cities. Each edge (i, j) is associated with a non-negative weight w_{ij} that represents the distance (or cost) of traveling from city i to city j.

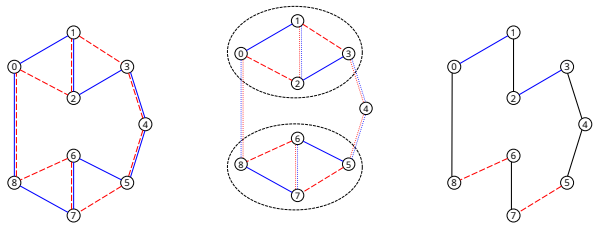
The goal is to find the shortest possible tour that visits each city exactly once and returns to the starting city. This type of tour is known as a Hamiltonian circuit. The tour length is the sum of the weights of the edges in the tour. A solution is represented as a permutation of vertices $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$, where $\pi : V \to V$. The goal is to find a permutation π that minimizes the following cost:

$$f(\pi) = \sum_{i=1}^{n-1} w_{\pi_i \pi_{i+1}} + w_{\pi_n \pi_1}$$

If $w_{ij} = w_{ji}$ for all $i, j \in V$ with $i \neq j$, the TSP is symmetric. Otherwise, the TSP is said to be asymmetric. In this work, we consider the symmetric case.

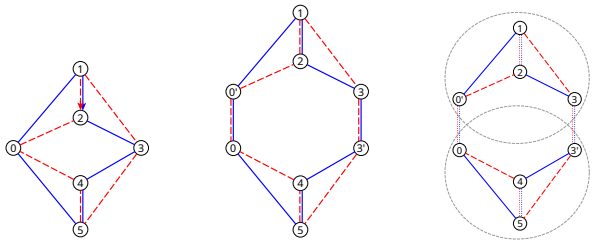
Many exact and inexact solvers have been proposed for the TSP [4]. The Concorde TSP Solver [1], a collection of linear and dynamic programming algorithms, is one of the most effective exact TSP solvers available. It has been used to solve many TSP instances [2]. However, the computational time required for exact solvers increases rapidly with the size of the problem, making them impractical for large TSP instances. Inexact methods have been more successful in attacking large problems, such as Lin-Kernighan-Helsgaun (LKH) [11], the Genetic Algorithm with the Edge-Assembly Crossover (GA-EAX) [14], and the Mixing Genetic Algorithm with the Generalized Partition Crossover 2 (MGA-GPX2) [27, 31].

The recombination of TSP solutions plays a central role in these methods. The problem of producing the best possible offspring by recombining two parent solutions is known as the *Optimal Recombination Problem*. It is an NP-hard problem with significant consequences for the design of recombination operators [10, 9]. In a seminal study about the impact of solution representation on the performance of genetic algorithms, Radcliffe and Surry [18] discuss two critical properties of recombination operators: *respect* and *transmission*. A recombination operator *respects* a representation when the offspring inherits all the information the parents share. A recombination operator *transmits* information when all the offspring information comes from at least one of its parents. Their work connected these properties with the success of some early recombination operators for the TSP, such as the Edge Recombination Operator [18, 34].



(a) $G = A \cup B$ **(b)** G^- **(c)** G^*

Figure 1: Recombination of two TSP tours in the Partition Crossover.



(a) G **(b)** G^+ **(c)** G^-

Figure 2: Splitting vertices of degree four.

The Generalized Partition Crossover 2 (GPX2) is a very effective partition crossover for the TSP [27]. Like other Partition Crossover operators, GPX2 is deterministic and can tunnel between local optima; if the parent solutions are local optima, the offspring will likely be locally optimal. This is possible because these operators preserve (or *respect*) the information shared by the parents and *transmit* only information available in the parents to the offspring.

GPX2 decomposes the union graph of two TSP tours into connected components known as AB-cycles. From these cycles, subsets of edges are selected and recombined to generate new tours. Given a graph partitioning with q AB-cycles, GPX2 finds the best offspring in a search space of size 2^k in $O(n)$ time, where $k \leq q$ is the number of recombining cycles, also known as recombining components. GPX2 distinguishes between *recombining* cycles, which can be independently recombined, and *non-recombining* cycles, which are grouped for recombination. To increase the number of recombining cycles, GPX2 employs several strategies. It introduces additional edges to increase the number of potential cuts and, consequently, the number of cycles. It also performs multiple scans of the union graph to identify more cycles. Finally, GPX2 applies a *fusion* process, which attempts to merge non-recombining cycles into recombining cycles. These strategies typically produce decompositions with more recombining cycles but at the cost of increased computation and memory usage.

We introduce the Generalized Partition Crossover 3 (GPX3) [16], a new implementation of Partition Crossover. We propose an alternative search algorithm that finds an equivalent set of AB-cycles as GPX2 without introducing additional edges. Building upon the ideas introduced in [17], we present a method to recombine AB-cycles regarded as non-recombining by GPX2. We demonstrate that the resulting performance is superior to or, at the very least, equivalent to GPX2 without *fusion*. Moreover, we show that these modifications can improve the operator's running time by approximately one order of magnitude.

2 Partition Crossover for the TSP

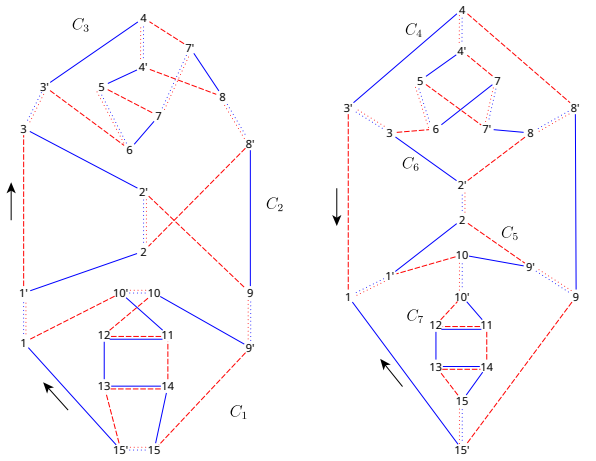
The Generalized Partition Crossover 2 (GPX2) [27] belongs to a family of partition crossover operators, including the original Partition Crossover (PX) introduced by Whitley et al. [33], the Generalized Partition Crossover (GPX) [32], and the Generalized Asymmetric Partition Crossover (GAPX) [28]. Initially developed for the TSP, Partition Crossover has been successfully combined with other methods [20, 21] and adapted to various optimization problems, such as Boolean Satisfiability [3, 8], Pseudo-Boolean Optimization [26, 15], the Vehicle Routing Problem [29], and the Steiner Tree Problem [6].

For a general understanding of how Partition Crossover works, consider the example in Figure 1, where the solid blue lines represent parent A, and the dashed red lines represent parent B (we will use this convention hereon). First, we create the union graph $G = A \cup B$ (Figure 1a). In the union graph, edges are either single edges (when they come from one parent) or shared edges (when they come from both parents). Some shared edges (or paths) are bridges that disconnect the graph if removed, such as the edge $(0, 8)$ and the path $(3, 4, 5)$. We create the graph G^- by removing all shared edges, potentially partitioning it into multiple connected components (Figure 1b).

Each subgraph this process identifies is a connected component known as an AB-cycle. Given that all shared edges are removed, each vertex in the cycle must have degree two and is incident to one edge from parent A and one from B. As a result, the cycles necessarily alternate single edges from A and B in a sequence $[ABAB\dots]$. This is clear in Figure 1b, cycle $(0, 1, 3, 2, 0)$, with the blue edges $\{(0, 1), (3, 2)\}$ and the red edges $\{(1, 3), (2, 0)\}$. AB-cycles always contain an even number of single edges, with at least four [27]. In the context of *fusion*, we also consider AB-cycles that include shared edges. Hence, each AB-cycle consists of two subsets of edges: one subset from A and one from B. We generate the offspring G^* by recombining the shortest subset of edges from each cycle with the shared edges previously removed (Figure 1c).

2.1 Splitting vertices of degree four

To potentially increase the number of cuts, GPX2 creates additional shared edges by splitting vertices of degree four. In the union graph, edges are either single edges or shared edges. As a result, all vertices are necessarily of degree two, three, or four. Often, vertices of degree four are articulation points that would disconnect G if removed. Consider the example in Figure 2. The union graph G is in Figure 2a. The shared edges $(1, 2)$ and $(4, 5)$ are not cuts; their removal does not disconnect the graph. However, the vertices 0 and 3, which have degree four, are articulation points. We create the graph G^+ by splitting degree four vertices in both solutions, creating the shared edges $(0, 0')$ and $(3, 3')$, both of cost zero (Figure 2b). These new shared edges are cuts whose removal disconnects the union graph (Figure 2c). The new vertices $0'$ and $3'$ are called *ghost* vertices or *ghost* nodes [27].



(a) Forward traversal (b) Reversed traversal

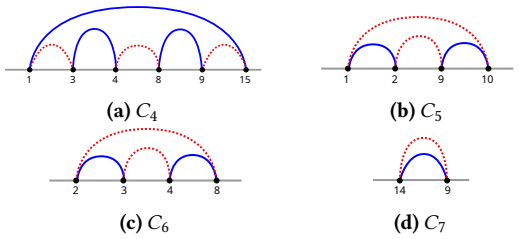
Figure 3: The two traversals used by GPX2 to partition the union graph. Example adapted from Tinós et al. [27]

There are two ways to split a vertex and reconnect the edges. Continuing with the example in Figure 2a, assume the shared edge $(1, 2)$ is directed, which sets a traversal direction for both solutions. Now consider the path $(5, 0, 1)$ in solution A and the path $(2, 0, 4)$ in solution B, where vertex 0 has degree four. We can introduce a ghost vertex after or before the original vertex. In this example, the *ghost* vertex appears after 0 in solution A, forming the path $(5, 0, 0', 1)$, and before 0 in solution B, forming $(2, 0', 0, 4)$ (Figure 2b). The placement determines how the neighboring vertices are reconnected. Because the graph is undirected, we could reach the same result by using the opposite placement and reversing the traversal direction in one of the solutions.

For a union graph with $m \leq n$ vertices of degree four, there are 2^m different ways to split them and, hence, 2^m potential partitionings (assuming independent choices). GPX2 avoids this search problem by fixing how vertices are split and varying only the traversal direction. In the *forward* traversal, both solutions are traversed in the same direction to perform the splits. In the *reversed* traversal, solution B is reversed before splitting. Removing the shared edges after the splits results in the partitions shown in Figures 3a and 3b.

2.2 Disjoint cycle selection with multiple scans

There are three cycles found after the *forward* traversal (Figure 3a, cycles C_1 to C_3) and four cycles found after the *reversed* traversal (Figure 3b, cycles C_4 to C_7). We need a disjoint set of AB-cycles for recombination, but we have two different partitionings of the union graph. For example, C_1 and C_2 share edges with C_5; some cycles are nested within others across traversals, such as $C_7 \subset C_1$. GPX2 addresses these issues by scanning the union graph multiple times while alternating the traversal pattern. Each scan results in a candidate list, from which the smallest recombining AB-cycle is kept, and the scan is restarted using the opposite traversal. This process results in a set of disjoint AB-cycles that might not match one or the other partitioning in Figure 3. For example, the smallest



(a) C_4 (b) C_5 (c) C_6 (d) C_7

Figure 4: Simplified graphs of all cycles in Figure 3b

recombining AB-cycle, C_7, is found during the *reversed* traversal. We keep C_7 and restart the scan with the opposite traversal, the *forward* traversal. Remember that $C_7 \subset C_1$. Now, the *forward* traversal will ignore the edges in C_7 and find the cycle $C_1 - C_7 = (1, 15, 9, 10, 1)$. This cycle cannot be found by one or the other traversal alone. By successively alternating traversals and selecting the smallest recombining AB-cycle at each step, GPX2 discovers all nested cycles, ultimately producing a disjoint set of AB-cycles.

If the number of scans is bounded by a positive constant n_r, the search has $O(n)$ time complexity [27]. In the currently available GPX2 implementation, $n_r = 1000$ [24].

2.3 Simplified graphs

Let q be the number of AB-cycles in G^-. GPX2 finds the best offspring in a search space of 2^k solutions in $O(n)$ time, where $k \leq q$ is the number of recombining AB-cycles. Each AB-cycle consists of two subsets of edges: one from A and one from B. Recombination involves selecting the lower-cost subset from each AB-cycle and reconnecting them using the previously removed shared edges to form a Hamiltonian circuit. However, not all such selections can be independently recombined without breaking the tour. To ensure that the final tour is valid, GPX2 distinguishes between two types of cycles: *recombining* cycles, which can be independently recombined, and *non-recombining* cycles, which must be grouped for recombination.

We introduce some concepts to discuss why some AB-cycles cannot be independently recombined. Consider the partitioning in Figure 3b again. If we traverse a tour in any direction, we can see that it necessarily enters/leaves each cycle an even number of times. For example, assuming the direction $(15', 1)$, tour A enters C_7 by vertex $10'$ and leaves it by vertex 15. These entry and exit points are called *portals*. Although tour B visits the vertices in C_7 in a different order than tour A, the portals for B are the same. We can simplify a tour inside each cycle using edges connecting the portals, which is called a *simplified graph*. Therefore, each cycle has two simplified graphs, one for each parent. A representation of the simplified graphs for each cycle of this example is in Figure 4. Cycle C_4 has six portals, cycles C_5 and C_6 have four portals each, and C_7 has two.

A cycle with two portals, like C_7, is recombining because the tour is not rerouted within the cycle. Selecting parent A or B inside the cycle does not affect the tour outside. As a generalization of this idea, a cycle with more than two portals is also a recombining cycle if the simplified graphs are the same. In the example, only cycle C_7 is a recombining cycle. C_4, C_5, and C_6 are not recombining

cycles because the simplified graphs are different for A and B (See Figure 4).

2.4 Residual subgraph and *fusion* of AB-cycles

Assume we want to perform recombination using the partitioning in Figure 3b, with cycles C_4, C_5, C_6, and C_7. We know that C_7 is a recombining cycle. What should we do with the residual subgraph composed of C_4, C_5, and C_6? The subtours in the residual subgraph come from one parent or another and are independent of the subtours selected in the recombining cycles. Consequently, the residual subgraph must also work as a recombining cycle.

If the residual subgraph is a recombining cycle, would merging subsets of cycles also result in recombining cycles? GPX2 explores this idea in a process called *fusion*. To avoid testing all merging combinations, GPX2 tries to merge cycles using different strategies, such as selecting adjacent cycles. As with the multiple scans of GPX2, *fusion* is also a high-cost bounded search. Because it incurs a high computational cost, it can be turned off if desired in the current GPX2 implementation.

3 Improving Partition Crossover for the TSP

The different versions of the Partition Crossover for the TSP essentially differ in their ability to partition the union graph and find recombining cycles. The original Partition Crossover works only with two cycles [33]. The second version, GPX, can recombine any number of cycles if each has only two portals [32]. GPX2 finds more cycles by splitting vertices of degree four and merging cycles. It recombines any number of cycles with any number of portals [27]. All these mechanisms make GPX2 a very effective recombination operator with a linear running time [25]. However, GPX2 could be redesigned to be faster. We propose a new search algorithm to find AB-cycles without splitting vertices. We also propose additional mechanisms to identify AB-cycles regarded as non-recombining by GPX2.

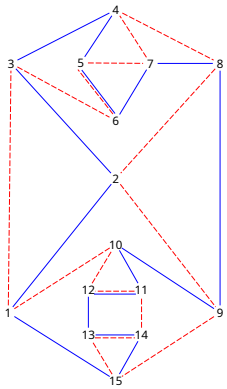
3.1 Finding AB-cycles fast without splitting

We propose a simple search algorithm that avoids vertex splitting and finds partitionings equivalent to those found by GPX2. Consider the example discussed in Section 2.1. We start with the parent solutions:

$$A: \ (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$$
$$B: \ (1\ 3\ 6\ 5\ 7\ 4\ 8\ 2\ 9\ 15\ 13\ 14\ 11\ 12\ 10)$$

The union graph is in Figure 5a. Consider the first two columns of the table in Figure 5b, where solution A is represented using an adjacency structure [23]. It can also be considered as a bidirectional cyclic permutation [22]. Each row i represents the two edges connected to vertex i. The elements $(i, Prev)$ and $(i, Next)$ represent the *previous* and *next* vertices connected to i in the permutation. This representation allows an efficient traversal of the solution in either direction, starting from any vertex. To represent the union graph, we horizontally stack the adjacency structures of A and B, forming what we call the *edge table*, where the first two columns represent A, and the remaining two represent B (Figure 5b).

There are multiple ways to scan the table because there are two solutions and directions in which they could be traversed. In fact,



(a) Union graph

	A		B	
	Prev	Next	Prev	Next
1:	15	2	10	3
2:	1	3	8	9
3:	2	4	1	6
4:	3	5	7	8
5:	4	6	6	7
6:	5	7	3	5
7:	6	8	5	4
8:	7	9	4	2
9:	8	10	2	15
10:	9	11	12	1
11:	10	12	14	12
12:	11	13	11	10
13:	12	14	15	14
14:	13	15	13	11
15:	14	1	9	13

(b) Edge table

Figure 5: Union graph of two TSP solutions and the corresponding edge table. Example adapted from Tinós et al. [27]

any arbitrary alternation of edges from A and B results in an AB-cycle. We can reproduce GPX2 traversals by scanning the table in a particular pattern. We first start by marking all shared edges as visited in both directions. We marked the shared edges as visited using gray in the table in Figure 5b. Then, we traverse the table by alternating between solutions, taking an edge from A, one from B, another from A, and so on. This alternating traversal continues until we return to the initial vertex, forming an AB-cycle.

Assume the traversal pattern *Next-Next* means we select an edge from A and then an edge from B using column "Next". If the edge is visited and its endpoint is not the starting vertex, we invert the traversal pattern to its counterpart *Prev-Prev*, meaning we now select edges using column "Prev". This has the effect of always traversing both solutions in the *same* direction. Starting the search at vertex $i = 1$, *Next-Next* will first visit the edge $(A, 1, 2)$; the index is then moved to vertex $i = 2$, leading to $(B, 2, 9)$; continuing this process, we get $(A, 9, 10)$ and $(B, 10, 1)$; the next edge would be

Algorithm 1 AB-cycle Search Algorithm

Assumptions:
All shared edges are marked as visited in both directions;
Initial edge $(A, i_{start}, Next)$ is unvisited;
Require: Edge table with solutions A and B;
Require: Initial vertice i_{start};
Require: Initial traversal $pattern$ (*Next-Next* or *Next-Prev*);
Ensure: $cycle$
Initialization:
$cycle \leftarrow$ empty list
$graph \leftarrow A$
$i \leftarrow i_{start}$
$pattern \leftarrow$ initial $pattern$
while true **do**
  append $edge$ $(graph, i, pattern)$ to $cycle$
  mark $edge$ as visited in both directions
  $i \leftarrow$ next index from $edge$ endpoint
  if $(i = i_{start})$ **and** $(graph = B)$ **and** $(|cycle| \geq 4$ edges$)$ **then**
    break
  end if
  $graph \leftarrow$ the other parent
  if $edge$ $(graph, i, pattern)$ is visited **then**
    invert $pattern$
  end if
end while
return $cycle$

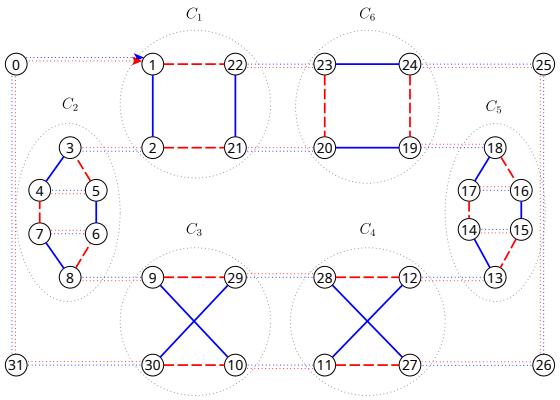


Figure 6: Recombining and non-recombining cycles. Example adapted from Tinós et al. [27]

$(A, 1, 2)$, but the search stops at this point because the endpoint of $(B, 10, 1)$ is the vertex the search started from. This results in the AB-cycle $(1, 2, 9, 10, 1)$, the same set of edges found by GPX2's *reversed* traversal (see Figure 3b).

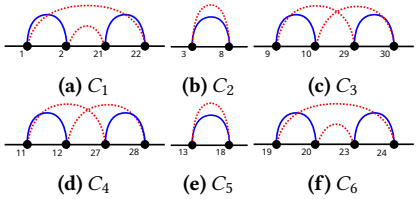
The pattern *Next-Next* yields GPX2' *reversed* traversal instead of the *forward* traversal. That happens because the shared edges created in G^+ force a change of direction during GPX2's traversal. Similarly, we can start the search with the pattern *Next-Prev* to reproduce GPX2's *forward* traversal. Again, if the search reaches a visited edge whose endpoint is not the starting vertex, it inverts the traversal pattern to its counterpart *Prev-Next*. This inversion is necessary to preserve the symmetry of the traversal.

The proposed search is described in Algorithm 1. The algorithm is complete and runs in $O(n)$ time. After removing all shared edges, each vertex in the union graph is of degree two or four. Hence, the graph is Eulerian and contains at least one AB-cycle. Each parent solution contains n edges, so the union graph has at most $2n$ edges. The search traverses each edge at most once. Visited edges are marked and skipped, and the process continues until all unvisited edges are processed. As a result, the total number of steps is proportional to $2n$, and the algorithm's time complexity is $O(n)$.

GPX3 finds the same AB-cycles and simplified graphs as GPX2 without splitting vertices and using less memory. GPX2 requires an $(n+m)\times 8$ edge table, where n is the number of cities and $m \leq n$ is the number of degree four vertices [27]. In contrast, GPX3's space usage is independent of m. It relies on two $n \times 4$ tables: an edge table and a visited edge table. GPX2 and GPX3 use the same space to find AB-cycles and simplified graphs when $m = 0$.

3.2 Finding more recombining AB-cycles

We now revisit the concept of simplified graphs and their relation with recombining cycles discussed in Section 2.3. We will use the example in Figure 6. In this example, there are no vertices of degree four. After removing the shared edges, the union graph is partitioned into six cycles $(C_1, C_2, \ldots, C_6)$. Their simplified graphs are in Figure 7. The cycles C_2 and C_5 are recombining because they have two portals each, and therefore, the same simplified graph. For GPX2, the remaining cycles are non-recombining because they



(a) C_1 **(b)** C_2 **(c)** C_3

(d) C_4 **(e)** C_5 **(f)** C_6

Figure 7: Simplified graphs of all cycles from the example in Figure 6

Table 1: Reduced recombination table from example in Figure 6

Recombination	C_1	C_6	C_3	C_4	Valid tour	GPX2	GPX3
1	A	B	A	A	×		
2	A	B	A	B	×		
3	A	B	B	A	×		
4	A	B	B	B	×		
5	B	A	A	A	×		
6	B	A	B	B	×		
7	B	A	A	B	✓	×	×
8	B	A	B	A	✓	×	×
9	A	A	A	A	✓	✓	✓
10	A	A	A	B	✓	×	✓
11	A	A	B	A	✓	×	✓
12	A	A	B	B	✓	✓	✓
13	B	B	A	A	✓	✓	✓
14	B	B	A	B	✓	×	✓
15	B	B	B	A	✓	×	✓
16	B	B	B	B	✓	✓	✓

each have different simplified graphs. If *fusion* is disabled, GPX2 uses the residual graph as a recombining cycle. So, although the partitioning results in six cycles, GPX2 uses three cycles for recombination, which reduces the space of reachable solutions from $2^{k=6}$ to $2^{k=3}$; otherwise, GPX2 would find two additional recombining cycles by merging C_1 with C_6 and C_3 with C_4.

Consider the recombination space with the four non-recombining cycles, represented in Table 1. Column C_i represents the subtour choice in cycle C_i; the cycles are ordered according to their adjacency in the union graph; the column "Valid tour" indicates whether the offspring is valid; the column "GPX2" indicates whether the offspring is reachable through recombination by GPX2; the column "GPX3" indicates whether the offspring is reachable through recombination using the proposed tests (we will explain that later). Because the two recombining cycles C_2 and C_5 are out of the table, each row represents 2^2 recombinations. There are six rows of invalid tours and ten rows of valid tours. Hence, the reachable space has $6 \times 2^2 = 24$ invalid tours, and $10 \times 2^2 = 40$ valid tours in a total of $2^{k=6} = 64$ recombinations, including the parents. That means $4 \times 2^2 = 16$ valid offspring are reachable using GPX2 with *fusion* enabled.

However, we can observe that, regardless of the choices in the other cycles, the offspring is invalid only when C_1 and C_6 are different (except in recombinations 7 and 8). Conversely, the offspring is always valid when the choices are the same in C_1 and C_6, which means C_3 and C_4 can be independently recombined as long as C_1 and C_6 are grouped.

(a) C_1 **(b)** C_2

Figure 8: Internal and external matching of C_1 and C_2

To better understand why this is the case, we expand on the concept of simplified graphs introduced in Section 2.3. A simplified graph is a perfect matching over the complete graph formed by the portals. A perfect matching is a set of disjoint edges that covers all vertices of a graph. Although not every union of perfect matchings corresponds to a Hamiltonian circuit, every Hamiltonian circuit is the union of two perfect matchings [5], and the union of perfect matchings relates to the problem of finding Hamiltonian circuits in a graph [7].

Recall that a simplified graph, hereon called matching, is a set of undirected edges that simplifies a subtour *inside* a cycle. By extension, we apply this concept to subtours *outside* a cycle [17]. A possible representation of this idea is shown in Figure 8. Given a cycle C_i, let A_i^{in}, A_i^{out}, B_i^{in}, and B_i^{out} be the internal and external matchings of solutions A and B. For example, in cycle C_3, we have:

$$A_3^{\text{in}} = \{(9, 10), (29, 30)\}, \qquad A_3^{\text{out}} = \{(9, 30), (10, 29)\},$$
$$B_3^{\text{in}} = \{(9, 29), (10, 30)\}, \qquad B_3^{\text{out}} = \{(9, 30), (10, 29)\}.$$

As expected, the union of internal and external matchings reconstructs a simplification of the original solutions:

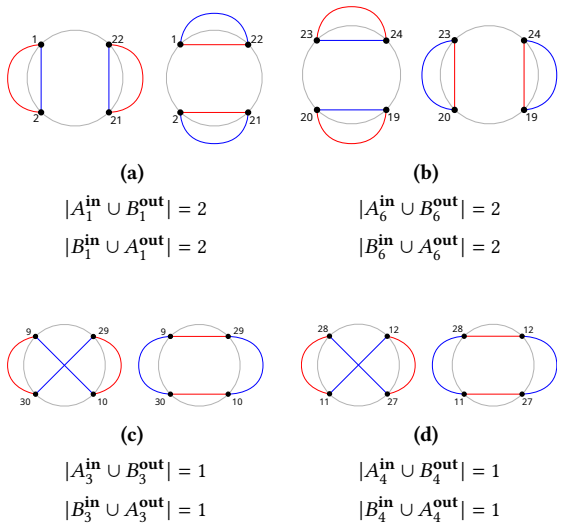
$$A_i^{\text{in}} \cup A_i^{\text{out}} = A, \quad B_i^{\text{in}} \cup B_i^{\text{out}} = B$$

Note that for C_3, $A_3^{\text{out}} = B_3^{\text{out}}$. That means the outside subtour is constant regardless of the subtour choice within C_3, but only if all other cycles are recombining. Consequently, choosing A or B within C_3 does not affect the final tour; hence, although its internal matchings are not equal, C_3 is a recombining cycle. That is also true for C_4.

What about the crossed unions $A_i^{\text{in}} \cup B_i^{\text{out}}$ and $B_i^{\text{in}} \cup A_i^{\text{out}}$? These unions represent a simplification of the offspring within the scope of cycle C_i. For example, $A_1^{\text{in}} \cup B_1^{\text{out}}$ corresponds to the offspring configuration when solution A is selected for cycle C_1 and solution B is selected for all other cycles. For the offspring to be valid, the crossed union must be a single cycle, i.e., $|A_1^{\text{in}} \cup B_1^{\text{out}}| = 1$. This condition is not always sufficient because we need to make different choices across the cycles, and the external matchings can change depending on the set of choices [17].

Additionally, there are cases in which $|A_i^{\text{in}} \cup B_i^{\text{out}}| = 1$ while $|B_i^{\text{in}} \cup A_i^{\text{out}}| > 1$, or vice-versa. We say these cycles are partially recombining for one of the parents. That is useful when $|A_i^{\text{in}} \cup B_i^{\text{out}}| = 1$ and A is shorter in cycle i, or vice versa.

Continuing with the example, consider the matchings in Figure 9, where we have the crossed unions for each cycle. For cycles C_1 and C_6, the crossed unions return more than one cycle. However, that is not true for cycles C_3 and C_4. Therefore, C_3 and C_4 are recombining cycles while C_1 and C_6 are not. This is consistent with our observations about Table 1. Except for recombinations 7 and 8, we cannot independently recombine C_1 and C_6.



(a)
$|A_1^{\text{in}} \cup B_1^{\text{out}}| = 2$
$|B_1^{\text{in}} \cup A_1^{\text{out}}| = 2$

(b)
$|A_6^{\text{in}} \cup B_6^{\text{out}}| = 2$
$|B_6^{\text{in}} \cup A_6^{\text{out}}| = 2$

(c)
$|A_3^{\text{in}} \cup B_3^{\text{out}}| = 1$
$|B_3^{\text{in}} \cup A_3^{\text{out}}| = 1$

(d)
$|A_4^{\text{in}} \cup B_4^{\text{out}}| = 1$
$|B_4^{\text{in}} \cup A_4^{\text{out}}| = 1$

Figure 9: Crossed unions

GPX3 implements additional tests to identify recombining cycles on top of the procedures implemented in GPX2. A cycle C_i is recombining (or partially recombining) if it satisfies any of the following conditions:

$$\begin{cases} |A_i^{\text{in}}| = 1 & \text{(two portals)} \\ A_i^{\text{in}} = B_i^{\text{in}} & \text{(same internal matchings)} \\ A^{\text{out}} = B^{\text{out}} & \text{(same external matchings)} \\ |A_i^{\text{in}} \cup B_i^{\text{out}}| = 1,\ |B_i^{\text{in}} \cup A_i^{\text{out}}| > 1 & \text{(single cycle for } A\text{)} \\ |A_i^{\text{in}} \cup B_i^{\text{out}}| > 1,\ |B_i^{\text{in}} \cup A_i^{\text{out}}| = 1 & \text{(single cycle for } B\text{)} \\ |A_i^{\text{in}} \cup B_i^{\text{out}}| = |B_i^{\text{in}} \cup A_i^{\text{out}}| = 1 & \text{(single cycle for } A \text{ and } B\text{)} \end{cases}$$

For the example in Figure 6, these tests double the space of reachable solutions from 16 valid solutions to 32 valid solutions, as we can see in Table 1, comparing columns "GPX2" and "GPX3".

GPX2 implements the first two tests and partially addresses the third using a procedure that updates the number of portals to identify additional recombining cycles. If two nested cycles share portals and the inner cycle is recombining, the shared portals are removed from the outer cycle. If only two portals remain in the outer cycle, it is classified as recombining. This allows GPX2 to identify cycles where $A^{\text{out}} = B^{\text{out}}$, as GPX3 does, but it does not cover all cases. For instance, the portals would not be updated in the example in Figure 6. First, the cycles are not nested; second, even if we propagate the portals of C_2 and C_5 through the bridges, removing portals 2, 9, 12, and 19 would leave each remaining cycle with three portals, which would be hard to interpret.

3.3 Recombination

As discussed, GPX2 scans the graph multiple times, alternating the traversal pattern to obtain a disjoint set of AB-cycles. GPX3 takes a different approach to avoid scanning the graph multiple times. First, GPX3 performs two scans, the *forward* and the *reversed* traversals, which result in two independent partitionings of the union graph, like the ones in Figure 3. Next, if a cycle is big enough, GPX3

Table 2: Average number and classification of AB-cycles found and used by GPX2-NF and GPX3 in the pairwise experiment. Results include total candidate cycles found, cycles in the partition used for the recombination (q), and their classification as recombining (k) or non-recombining ($q - k$).

	Candidates		Partition (q)		Recombining (k)		Non-Recombining ($q - k$)	
TSP	GPX2-NF	GPX3	GPX2-NF	GPX3	GPX2-NF	GPX3	GPX2-NF	GPX3
pcb442	216.20 ± 15.90	41.26 ± 7.80	20.81 ± 4.00	20.66 ± 3.96	6.92 ± 2.65	7.85 ± 2.81	13.89 ± 3.44	12.81 ± 3.33
d657	321.14 ± 20.94	43.41 ± 7.94	21.80 ± 4.06	21.73 ± 4.01	6.88 ± 2.61	7.82 ± 2.80	14.93 ± 3.44	13.91 ± 3.33
pr1002	453.03 ± 25.46	76.96 ± 10.90	38.51 ± 5.55	38.47 ± 5.49	10.98 ± 3.33	12.75 ± 3.59	27.53 ± 4.87	25.71 ± 4.66
vm1084	707.19 ± 24.40	58.64 ± 9.16	29.45 ± 4.67	29.23 ± 4.59	9.85 ± 3.10	11.42 ± 3.31	19.60 ± 4.29	17.81 ± 4.07
pcb1173	606.40 ± 28.92	72.77 ± 10.89	36.28 ± 5.51	36.40 ± 5.45	13.31 ± 3.66	15.00 ± 3.91	22.97 ± 4.51	21.40 ± 4.35
d1291	907.29 ± 29.31	56.62 ± 9.96	28.56 ± 5.09	28.34 ± 5.05	10.00 ± 3.45	11.16 ± 3.68	18.56 ± 4.19	17.18 ± 3.99
rl1304	973.19 ± 25.89	40.15 ± 8.31	20.19 ± 4.24	20.04 ± 4.16	8.52 ± 2.89	9.63 ± 3.12	11.67 ± 3.32	10.42 ± 3.05
rl1323	990.66 ± 25.67	39.65 ± 7.90	19.72 ± 3.99	19.83 ± 3.97	7.66 ± 2.69	8.85 ± 2.90	12.07 ± 3.19	10.97 ± 3.03
nrw1379	587.14 ± 30.04	100.01 ± 11.84	50.25 ± 6.10	50.08 ± 6.05	13.95 ± 3.65	16.23 ± 3.94	36.30 ± 5.51	33.85 ± 5.37
fl1400	474.32 ± 30.19	179.86 ± 15.73	90.47 ± 8.10	90.09 ± 8.03	27.28 ± 5.10	32.61 ± 5.72	63.19 ± 7.18	57.48 ± 6.82
fl1577	1033.57 ± 37.87	85.10 ± 12.68	42.34 ± 6.34	42.53 ± 6.35	21.52 ± 4.95	24.74 ± 5.32	20.82 ± 4.78	17.79 ± 4.48
d1655	966.79 ± 33.39	93.06 ± 11.82	46.91 ± 6.08	46.51 ± 5.94	12.29 ± 3.61	14.28 ± 3.85	34.62 ± 5.28	32.23 ± 5.05
vm1748	1152.13 ± 32.12	80.73 ± 11.20	40.31 ± 5.69	40.34 ± 5.64	12.95 ± 3.61	14.79 ± 3.88	27.36 ± 4.92	25.55 ± 4.72
u1817	1000.90 ± 33.80	93.50 ± 12.59	46.92 ± 6.43	46.79 ± 6.35	12.77 ± 3.61	14.71 ± 3.85	34.14 ± 5.74	32.09 ± 5.46
rl1889	1343.13 ± 30.27	69.64 ± 10.28	34.76 ± 5.20	34.78 ± 5.16	13.64 ± 3.61	15.68 ± 3.84	21.12 ± 4.41	19.10 ± 4.13
d2103	1405.69 ± 65.86	54.10 ± 9.82	27.21 ± 5.01	27.02 ± 4.94	7.43 ± 2.82	8.74 ± 3.08	19.78 ± 4.32	18.28 ± 4.12
u2319	641.90 ± 28.00	311.62 ± 20.95	160.49 ± 11.28	156.11 ± 11.31	19.79 ± 4.48	25.43 ± 4.98	140.70 ± 10.85	130.68 ± 10.60
pr2392	1240.04 ± 41.62	146.62 ± 15.04	73.45 ± 7.72	73.27 ± 7.57	25.65 ± 4.96	28.60 ± 5.21	47.80 ± 6.75	44.67 ± 6.44
pcb3038	1386.34 ± 45.38	191.96 ± 17.44	96.25 ± 9.00	95.98 ± 8.81	28.73 ± 5.25	32.28 ± 5.60	67.52 ± 7.84	63.70 ± 7.45
fnl4461	1822.34 ± 52.84	303.91 ± 21.45	152.83 ± 11.01	152.20 ± 10.91	42.39 ± 6.53	48.20 ± 6.89	110.44 ± 9.69	104.00 ± 9.33

scans it using the opposite traversal to find embedded cycles. In the example in Figure 3, cycle C_1 is found by the *forward* traversal. Remember that $C_7 \subset C_1$. Scanning C_1 with the *reversed* traversal should find C_7 embedded in it. Considering AB-cycles have at least four edges, it is not necessary to rescan cycles with less than eight edges. That means GPX3 can solve all embeddings by scanning the graph at most four times in the worst scenario. Still, that leaves us with two independent partitionings of the union graph. Each partitioning corresponds to a distinct search subspace. GPX3 evaluates which partitioning reaches the best possible offspring and uses that partitioning for the recombination.

4 Mixing GA

The Mixing Genetic Algorithm (MGA) is a novel generational evolutionary algorithm that, unlike traditional genetic algorithms, does not apply selection or mutation [30]. It organizes the population so that fit solutions are often recombined with fit solutions, while unfit solutions are often recombined with unfit solutions. This results in some interesting properties for evolutionary algorithms. First, the fitness of half the population steadily increases, while the fitness of the other half decreases. Additionally, MGA does not converge prematurely, as all genetic information is preserved across generations. The Partition Crossover couples well with MGA because it retains all information during recombination and can generate the best possible offspring and its complement. Selecting A or B in a cycle can be seen as a binary choice. The complementary offspring is obtained by negating the choices that generate the best offspring. In the search space lattice, the complementary solution occupies a mirrored position relative to the best solution. Previous studies have shown that MGA combined with Partition Crossover is highly effective in solving the TSP [30].

5 Experiments

We conducted 30 experimental trials of the MGA with each operator (GPX2 without *fusion* and GPX3) to evaluate the proposed implementation across 20 problems from the TSPLIB dataset [19]. The MGA terminates the execution if the optimal solution is found or after 10,000 generations. For each trial, we generated a unique population of 1024 individuals optimized using the 2-opt heuristic implemented in GA-EAX [13]. One limitation of the MGA is that the initial population must contain all the edges necessary to assemble the globally optimal solution. The initial populations were tested to guarantee that the edges of the optimal solution were present. The experiments were executed under similar conditions in a distributed computing environment with identical machines. The results are in Tables 3 and 4.

To count the number and types of cycles found by each operator, we also ran a single pairwise experiment on each TSP instance using a population of 4096 random individuals optimized with 2-opt. In this experiment, we paired consecutive solutions for recombination, ensuring that GPX2 and GPX3 recombined identical pairs of parents. The results are in Table 2.

GPX2 was adjusted to generate the best offspring and its complement. Because GPX3 does not implement a *fusion* mechanism, we turned *fusion* off in GPX2 to evaluate the proposed improvements. When relevant, we refer to GPX2 without fusion as "GPX2-NF". GPX2 and GPX3 were compiled with the same optimization flags.

6 Results

Table 2 shows the average count and types of AB-cycles found during recombination in the pairwise experiment, in which we created 2048 distinct solution pairs from a population of 4096 random individuals optimized with 2-opt. Each pair was recombined once by GPX2-NF and once by GPX3, resulting in 2048 distinct recombination events per operator per TSP instance.

The column "Candidates" shows the average number of candidate AB-cycles found by each operator. For GPX2-NF, this number corresponds to the cycles identified during multiple scans of the union graph; for GPX3, it corresponds to the cycles found after the forward and reversed scans. As expected, GPX2-NF's multiple scans find many more candidate cycles than GPX3.

Table 3: Results of Mixing GA with a population size of 1024 on 20 TSP instances.

	Average Gap (%)		Optimal/Trials	
TSP	GPX2-NF	GPX3	GPX2-NF	GPX3
pcb442	0.052	**0.038**	18/30	9/30
d657	0.083	**0.053**	0/30	4/30
pr1002	0.531	**0.324**	0/30	0/30
vm1084	0.123	**0.019**	0/30	0/30
pcb1173	0.710	**0.326**	0/30	0/30
d1291	0.353	**0.165**	0/30	0/30
rl1304	0.097	**Opt.**	25/30	30/30
rl1323	0.045	**0.022**	7/30	7/30
nrw1379	0.380	**0.174**	0/30	0/30
fl1400	0.244	**0.068**	1/30	24/30
fl1577	1.583	**0.595**	0/30	3/30
d1655	0.599	**0.345**	0/30	0/30
vm1748	0.217	**0.092**	0/30	0/30
u1817	1.019	**0.684**	0/30	0/30
rl1889	0.377	**0.121**	0/30	0/30
d2103	0.236	**0.067**	0/30	0/30
u2319	**0.212**	0.247	0/30	0/30
pr2392	0.943	**0.551**	0/30	0/30
pcb3038	1.142	**0.661**	0/30	0/30
fnl4461	1.000	**0.574**	0/30	0/30

The column "Partition (q)" shows the average number of cycles in the partitions used for recombination. On average, GPX3 and GPX2-NF use partitions with a similar number of cycles. Except for *u2319* and *d1655*, all problems have p-values greater than 0.05, indicating no statistically significant difference. This suggests that the proposed AB-cycle search in GPX3 is statistically equivalent to the multiple scans performed by GPX2-NF regarding the number of cycles used for recombination.

The last two columns in Table 2 show the number of recombining (k) and non-recombining ($q - k$) cycles identified by GPX2-NF and GPX3. GPX3 identifies more recombining cycles than GPX2-NF on average. As a consequence, fewer cycles are identified as non-recombining compared to GPX2-NF. For these two columns, $p < 0.05$ for all TSP instances, indicating the difference is statistically significant. In absolute values, GPX3 found 98,718 recombining cycles for the TSP instance *fnl4461*, compared to 86,805 found by GPX2-NF. Now, the question is whether GPX3 translates these figures into improvements of the recombined solutions compared with GPX2-NF.

Table 3 shows the average gap (in %) observed and the number of trials in which the operator found the optimal solution. GPX3 found the optimal solution more frequently than GPX2-NF, except for problems *pcb442* and *rl1323*. When the optimal solution was not found, the MGA with GPX3 generated offspring with a smaller average gap than GPX2-NF for all problems except *u2319*.

Table 4 shows the average execution time and the time ratio *GPX2-NF/GPX3* for problems where the operators did not find the optimal solution, i.e., when the trial was not terminated early. On average, GPX3 is an order of magnitude faster than GPX2-NF. For all TSP instances, $p < 0.05$, indicating the results are statistically significant. This is also clear in Figure 10, where problem sizes are plotted against the execution time on a logarithmic scale.

7 Conclusion

In this work, we presented GPX3, an improved partition crossover for the TSP. We introduced a fast AB-cycle search algorithm that avoids the computational overhead of splitting vertices and scanning the union graph multiple times. This approach achieves an equivalent partitioning of the union graph compared to GPX2 while significantly reducing the number of candidate cycles that need to be evaluated. Additionally, we proposed a new set of classification tests for recombining cycles, which expands the set of cycles that can be independently recombined. This effectively increases the reachable search space compared to GPX2 without *fusion*.

Experimental results demonstrate that GPX3 is approximately one order of magnitude faster than the current GPX2 implementation. The MGA with GPX3 found optimal solutions more frequently than GPX2 without fusion, with smaller gaps when the optimal solution was not found. The statistical significance of the results supports the relevance of the proposed improvements. GPX2-NF and GPX3 are mostly ineffective at finding the optimal solution. Results from similar experiments in the literature demonstrate that the standard GPX2 (with *fusion*) frequently finds optimal solutions for most of the considered TSP instances [30]. However, this does not undermine the present results because *fusion* is an independent process.

GPX3 represents a significant step forward in the design of recombination operators for the TSP. By simplifying cycle search and identification, it offers a more efficient alternative to GPX2. Future work could explore the integration of GPX3 with *fusion* mechanisms. In its current state, GPX3 is not as effective as the standard GPX2 (with *fusion*). Since *fusion* is an independent process, we expect that integrating it with GPX3 would make it as effective as the standard GPX2 while preserving the relative speed-up.

Table 4: Execution time - MGA

TSP	GPX2-NF	GPX3	GPX2-NF/GPX3
pr1002	9526.23 ± 1429.51	**807.60 ± 132.63**	11.80
vm1084	8139.11 ± 2978.51	**603.72 ± 167.76**	13.48
pcb1173	6546.43 ± 81.73	**616.48 ± 40.41**	10.62
d1291	6444.04 ± 1417.13	**467.16 ± 59.43**	13.79
nrw1379	8810.40 ± 45.93	**905.84 ± 208.68**	9.73
d1655	15458.69 ± 2135.37	**1139.31 ± 186.52**	13.57
vm1748	12989.47 ± 5009.05	**965.89 ± 258.59**	13.45
u1817	10204.30 ± 1441.86	**929.33 ± 57.47**	10.98
rl1889	10721.15 ± 2369.80	**670.92 ± 78.38**	15.98
d2103	16518.35 ± 2142.97	**930.89 ± 103.70**	17.74
u2319	26326.57 ± 8237.99	**3248.80 ± 373.79**	8.10
pr2392	18218.08 ± 3751.22	**1365.89 ± 312.96**	13.34
pcb3038	19663.17 ± 690.10	**1908.98 ± 221.44**	10.30
fnl4461	35378.63 ± 1559.81	**3375.27 ± 292.55**	10.48
		Average	12.38

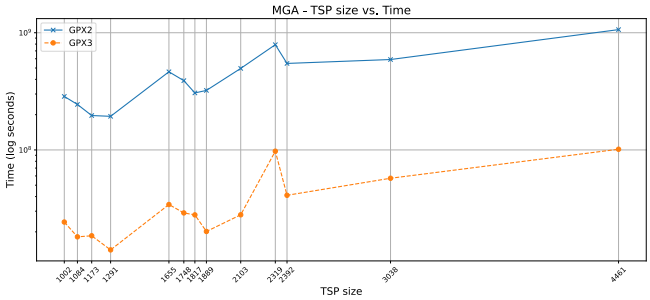


Figure 10: Execution time - MGA

References

[1] David Applegate, Robert Bixby, Václav Chvátal, and William J. Cook. 2003. Concorde tsp solver. C. Accessed: 2025-04-08. (2003). https://www.math.uwaterloo.ca/tsp/concorde/.

[2] David L. Applegate, Robert E. Bixby, Vašek Chvatál, and William J. Cook. 2006. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press. ISBN: 978-0-691-12993-8. https://www.jstor.org/stable/j.ctt7s8xg.

[3] Wenxiang Chen, Darrell Whitley, Renato Tinós, and Francisco Chicano. 2018. Tunneling between plateaus: improving on a state-of-the-art maxsat solver using partition crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO '18). Association for Computing Machinery, New York, NY, USA, (July 2018), 921–928. ISBN: 978-1-4503-5618-3. doi:10.1145/3205455.3205482.

[4] William Cook. 2014. *In pursuit of the traveling salesman: mathematics at the limits of computation*. eng. (third printing, and first paperback printing ed.). Princeton University Press, Princeton. ISBN: 978-0-691-16352-9.

[5] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. 2018. Fast hamiltonicity checking via bases of perfect matchings. en. *Journal of the ACM*, 65, 3, (June 2018), 1–46. doi:10.1145/3148227.

[6] Giliard Almeida de Godoi, Renato Tinós, and Danilo Sipoli Sanches. 2021. A graph-based crossover and soft-repair operators for the steiner tree problem. en. In *Intelligent Systems*. André Britto and Karina Valdivia Delgado, (Eds.) Springer International Publishing, Cham, 111–125. ISBN: 978-3-030-91702-9. doi:10.1007/978-3-030-91702-9_8.

[7] Dwight Duffus, Bill Sands, and Robert Woodrow. 1988. Lexicographic matchings cannot form hamiltonian cycles. en. *Order*, 5, 2, 149–161. doi:10.1007/BF00337620.

[8] Preston Dunton and Darrell Whitley. 2022. Reducing the cost of partition crossover on large maxsat problems: the px-preprocessor. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO '22). Association for Computing Machinery, New York, NY, USA, (July 2022), 694–702. ISBN: 978-1-4503-9237-2. doi:10.1145/3512290.3528748.

[9] Anton Eremeev and Julia Kovalenko. 2014. Optimal recombination in genetic algorithms for combinatorial optimization problems: part ii. en. *Yugoslav Journal of Operations Research*, 24, 2, 165–186. doi:10.2298/YJOR131030041E.

[10] Anton V. Eremeev and Julia V. Kovalenko. 2013. Optimal recombination in genetic algorithms. en. arXiv:1307.5519, (July 2013). arXiv:1307.5519 [cs]. http://arxiv.org/abs/1307.5519.

[11] Keld Helsgaun. 2009. General k-opt submoves for the lin–kernighan tsp heuristic. en. *Mathematical Programming Computation*, 1, 2, (Oct. 2009), 119–163. doi:10.1007/s12532-009-0004-6.

[12] David Johnson and Christos Papadimitriou. 1981. Computational complexity and the traveling salesman problem. en. *LCS Technical Memos*, (Dec. 1981). Accepted: 2023-03-29T14:20:09Z. https://dspace.mit.edu/handle/1721.1/149020.

[13] Yuichi Nagata. 2021. GA-EAX: genetic algorithm with edge assembly crossover for the tsp. C++. Accessed: 2025-04-08. (2021). https://github.com/nagata-yuichi/GA-EAX.

[14] Yuichi Nagata and Shigenobu Kobayashi. 2013. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. en. *INFORMS Journal on Computing*, 25, 2, (May 2013), 346–363. doi:10.1287/ijoc.1120.0506.

[15] Michal W. Przewozniczek, Renato Tinós, Bartosz Frej, and Marcin M. Komarnicki. 2022. On turning black - into dark gray-optimization with the direct empirical linkage discovery and partition crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO '22). Association for Computing Machinery, New York, NY, USA, (July 2022), 269–277. ISBN: 978-1-4503-9237-2. doi:10.1145/3512290.3528734.

[16] Ozéas Quevedo de Carvalho. 2025. GPX3: Generalized Partition Crossover for the TSP. C. Accessed: 2025-04-14. (2025). https://github.com/ozeasx/GPX3.

[17] Ozéas Quevedo de Carvalho, Renato Tinós, Darrell Whitley, and Danilo Sipoli Sanches. 2019. A new method for identification of recombining components in the generalized partition crossover. In *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE, Salvador, Brazil, (Oct. 2019), 36–41. ISBN: 978-1-72814-253-1. doi:10.1109/BRACIS.2019.00016.

[18] 1995. *Fitness variance of formae and performance prediction*. en. *Foundations of Genetic Algorithms*. Vol. 3. Elsevier, 51–72. ISBN: 978-1-55860-356-1. doi:10.1016/B978-1-55860-356-1.50007-8.

[19] Gerhard Reinelt. 1995. TSPLIB95: a traveling salesman problem library. Accessed: 2025-04-08. (1995). http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[20] 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 the Genetic and Evolutionary Computation Conference* (GECCO '17). Association for Computing Machinery, New York, NY, USA, (July 2017), 329–336. ISBN: 978-1-4503-4920-8. doi:10.1145/3071178.3071305.

[21] Danilo Sanches, Darrell Whitley, and Renato Tinós. 2017. Improving an exact solver for the traveling salesman problem using partition crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO '17). Association for Computing Machinery, New York, NY, USA, (July 2017), 337–344. ISBN: 978-1-4503-4920-8. doi:10.1145/3071178.3071304.

[22] Sandra Sattolo. 1986. An algorithm to generate a random cyclic permutation. *Information Processing Letters*, 22, 6, (May 1986), 315–317. doi:10.1016/0020-0190(86)90073-6.

[23] Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1, 2, (June 1972), 146–160. doi:10.1137/0201010.

[24] Renato Tinós. 2021. GPX2: Generalized Partition Crossover for the TSP. C++. Accessed: 2025-04-08. (2021). https://github.com/rtinos/gpx2.

[25] Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and Darrell Whitley, (Eds.) 2018. *Efficient recombination in the lin-kernighan-helsgaun traveling salesman heuristic. Parallel Problem Solving from Nature – PPSN XV*. Vol. 11101. Lecture Notes in Computer Science. Springer International Publishing, Cham, 95–107. ISBN: 978-3-319-99252-5. doi:10.1007/978-3-319-99253-2_8.

[26] Renato Tinós, Darrell Whitley, and Francisco Chicano. 2015. Partition crossover for pseudo-boolean optimization. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII* (FOGA '15). Association for Computing Machinery, New York, NY, USA, (Jan. 2015), 137–149. ISBN: 978-1-4503-3434-1. doi:10.1145/2725494.2725497.

[27] Renato Tinós, Darrell Whitley, and Gabriela Ochoa. 2020. A new generalized partition crossover for the traveling salesman problem: tunneling between local optima. *Evolutionary Computation*, 28, 2, (June 2020), 255–288. doi:10.1162/evco_a_00254.

[28] Renato Tinós, Darrell Whitley, and Gabriela Ochoa. 2014. Generalized asymmetric partition crossover (gapx) for the asymmetric tsp. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (GECCO '14). Association for Computing Machinery, New York, NY, USA, (July 2014), 501–508. ISBN: 978-1-4503-2662-9. doi:10.1145/2576768.2598245.

[29] Takwa Tlili, Francisco Chicano, Saoussen Krichen, and Enrique Alba. 2015. Evolutionary algorithm based on partition crossover (eapx) for the vehicle routing problem. In *2015 International Conference on Intelligent Networking and Collaborative Systems*. (Sept. 2015), 169–175. doi:10.1109/INCoS.2015.89.

[30] Swetha Varadarajan and Darrell Whitley. 2019. The massively parallel mixing genetic algorithm for the traveling salesman problem. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO '19). Association for Computing Machinery, New York, NY, USA, (July 2019), 872–879. ISBN: 978-1-4503-6111-8. doi:10.1145/3321707.3321772.

[31] Swetha Varadarajan, Darrell Whitley, and Gabriela Ochoa. 2020. Why many travelling salesman problem instances are easier than you think. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (GECCO '20). Association for Computing Machinery, New York, NY, USA, (June 2020), 254–262. ISBN: 978-1-4503-7128-5. doi:10.1145/3377930.3390145.

[32] Darrell Whitley, Doug Hains, and Adele Howe. 2010. A hybrid genetic algorithm for the traveling salesman problem using generalized partition crossover. en. In *Parallel Problem Solving from Nature, PPSN XI* (Lecture Notes in Computer Science). Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, (Eds.) Springer, Berlin, Heidelberg, 566–575. ISBN: 978-3-642-15844-5. doi:10.1007/978-3-642-15844-5_57.

[33] Darrell Whitley, Doug Hains, and Adele Howe. 2009. Tunneling between optima: partition crossover for the traveling salesman problem. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (GECCO '09). Association for Computing Machinery, New York, NY, USA, (July 2009), 915–922. ISBN: 978-1-60558-325-9. doi:10.1145/1569901.1570026.

[34] L. Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. 1989. Scheduling problems and traveling salesmen: the genetic edge recombination operator. In *Proceedings of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, (June 1989), 133–140. ISBN: 978-1-55860-066-9.