A Fusion Mechanism for the Generalized Asymmetric Partition Crossover

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

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

***Abstract*—Partition crossover operators use information about the interaction between decision variables to recombine solutions. The Generalized Asymmetric Partition Crossover (GAPX) was recently proposed for the asymmetric Traveling Salesman Problem (TSP). Unlike former partition crossover operators, GAPX is capable of finding crossover points by splitting vertices of degree 4. GAPX also finds recombining components with more than two crossover points. The first step of GAPX is to define candidate components for recombination by finding connected components in the union graph formed by two parents. Some of the candidate components are infeasible for recombination. However, candidate components can be fused in order to create new recombining components. We introduce a fusion mechanism that allows GAPX to find more recombining components. When k recombining components are found, GAPX generates the best of 2^k offspring at cost $O(n)$. When two local optima are recombined by partition crossover, the offspring is very often a local optimum. Fusion can be used to increase k, allowing GAPX to exploit many more offspring. Experimental results show that GAPX with fusion is capable of improving solutions generated by the LKH heuristic. Very good results are also obtained by a hybrid Genetic Algorithm that uses GAPX with fusion.**

***Index Terms*—Recombination Operators, Traveling Salesman Problem, Combinatorial Optimization, Genetic Algorithms.**

I. Introduction

Several optimization algorithms have been proposed for the *Traveling Salesman Problem* (TSP) since it was formally introduced more than 70 years ago [1]. An instance of the TSP is defined by a weighted graph $G(V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n vertices and the set of edges is given by E. Each edge $e_{i,j} \in E$ between vertices $v_i, v_j \in V$ is associated with a weight $w_{i,j} \in \mathbb{R}^+$. Solutions of the TSP are Hamiltonian cycles in G. The evaluation of a particular solution $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}} \in X$ is given by:

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

The objective of the optimization algorithm is to find $\mathbf{x} \in X$ with the minimum evaluation $f(\mathbf{x})$.

Exact methods such as Concorde [1] are capable of solving instances of the TSP with hundreds of cities in seconds. However, exact methods require prohibitive computational costs for very large TSP instances. Several heuristics have been proposed, some of them with very impressive results. The most successful heuristics, e.g., the *Lin-Kernighan-Heusgaum* (LKH) heuristic [2], use some form of recombination. Crossover is also used in population based metaheuristics applied to the TSP. Thus, the search for optimal recombination operators for the TSP is an important research topic [3]–[5].

One recently proposed operator for the *asymmetric TSP* (ATSP) is *Generalized Asymmetric Partition Crossover* (GAPX) [4]. In the ATSP, the weight matrix is asymmetric. Unlike most recombination operators for the TSP, partition crossover operators are both respectful and transmit genes, i.e.: i) all common edges between the parents are inherited by the offspring; ii) offspring are composed only by edges contained in the parents. Those two properties result in an interesting phenomenon: when two local optima are recombined, the offspring are very often local optima. The analysis of Local Optima Networks for the ATSP in [6] reveals the existence of multiple valleys of local optima and illustrates GAPX's ability to tunnel between local optima in $O(n)$ time.

GAPX presents two enhancements when compared to former partition crossover operators for the symmetric TSP [7]. First, GAPX is capable of finding crossover points that split vertices with degree 4. Second, GAPX finds recombining components with more than two entry points. The first step of GAPX is to create a union graph formed by two parent solutions. Then the common edges are removed and the connected components of the resulting graph are found. All connected components are candidate components for recombination. Finding cutting points that are vertices with degree 4 results in much more candidate components. By considering that some candidate components with more than two entry points are also recombining components, much more recombining components are found. When k recombining components are found, GAPX generates the best of 2^k offspring at cost $O(n)$. This is done by selecting the best paths inside each recombining component: if the path inside the recombining component is shorter for the first parent, then it is chosen; otherwise, the path is chosen from the second parent. For example, if $k = 10$ recombining components are found in one recombination by GAPX, the best of 2^{10} offspring is returned. This is done in $O(n)$ time because GAPX uses a

This research was partially funded by São Paulo Research Foundation (FAPESP) under grant 2015/06462-1.

978-1-5090-6017-7/18/$31.00 ©2018 IEEE

greedy strategy to pick the best partial solution for each of the k recombining components.

Nevertheless, some of the candidate components found by GAPX are infeasible for recombination. However, two (or more) infeasible candidate components can sometimes be fused in order to create a new recombining component. We introduce in this paper a fusion mechanism that allows GAPX to find more recombining components, i.e., to increase k. An exponentially larger number of offspring is exploited when k linearly increases. The computational cost of GAPX is not altered by adding the fusion mechanism proposed here: it still executes in $O(n)$ time.

The GAPX and the fusion mechanism are respectively presented in sections II and III. GAPX can be applied as a recombining operator when other efficient heuristic methods are used to solve the ATSP. Section IV demonstrates the efficiency of the GAPX when: i) GAPX is used to recombine solutions inside a hybrid Genetic Algorithm; and ii) GAPX is applied to improve solutions generated by multi-trial LKH. The experimental results indicate that GAPX can be employed to produce very competitive algorithms. The paper is concluded in Section V with a discussion of future work.

II. Generalized Asymmetric Partition Crossover

Partition crossover operators recombine partial solutions that are not shared in common between two parent solutions [4], [7], [8]. Given two parents represented by graphs G_1 and G_2, GAPX creates a union graph $G_u = G_1 \cup G_2$. Some of the edges of the G_u are common for both parents, i.e., they are linked to the same vertices and have the same flow direction. The next step of GAPX is to remove common edges from G_u. Connected components are then identified using Breadth First Search or Depth First Search[1].

All the connected components are *candidate components* for recombination. Some candidate components can be recombined, i.e., when the partial solutions inside the component are linked to partial solutions inside other components, they always generate Hamiltonian cycles. Connected components that are feasible for recombination are called *recombining components*. GAPX deterministically finds the best recombinations by selecting the best partial solutions inside the recombining components. This is possible because the evaluations of the partial solutions inside a recombining component are independently computed. When k recombining components are identified, the best of 2^k offspring is found at computational cost $O(n)$.

Candidate components are always recombining components if they can be separated from union graph G_u by removing exactly 2 common edges. In this case, both tours enters the candidate component in one vertex v_{in} and exits the component in another vertex v_{out}. For the ATSP, entries and exits are exactly defined because all of the (common) edges in the ATSP are directional. It is easy to prove that all offspring generated by recombining connected components with 1 entry and 1 exit are Hamiltonian cycles [7].

GAPX includes two enhancements to previous forms of partition crossover. One enhancement is the identification of recombining components with multiple entries and exits. These recombining components are separated from the rest of the graph by removing more than 2 common edges. However, the number of common edges must be even, with an equal number of entries and exits. These additional recombining components are identified by creating *simplified graphs*. A simplified graph is created for each candidate component by removing all of the vertices except the entry and exit vertices. We then check if the simplified paths inside the candidate component are identical for both parents. If the simplified paths are identical, then inheriting a path inside a candidate component from one or another parent always yields a Hamiltonian cycle [4]. Thus if the simplified paths are identical for a candidate component, then that candidate component is also a recombining component. Figure 1 shows an example with two recombining components.

A second enhancement is the discovery of new crossover opportunities that are found by splitting vertices of degree 4 in G_u. A "ghost" vertex is created for every vertex of degree 4. The creation of ghost vertices is very easy: a new vertex is inserted immediately after each vertex of degree 4 in both parents. The notion of "after" is well defined because all edges are directional in the ATSP. Thus, each original and ghost vertex are linked by a common edge with weight equal to zero. The new common edges are new candidate sites for crossover.

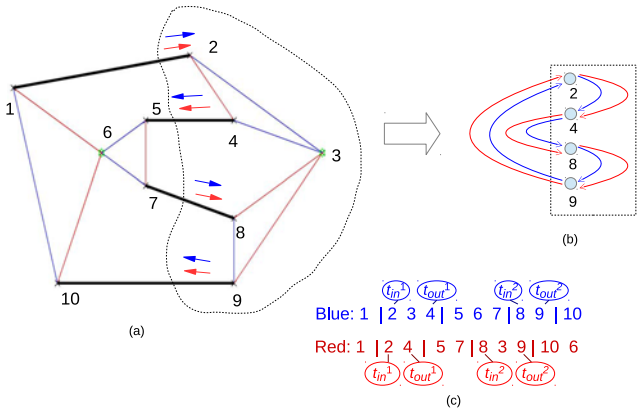

Fig. 1. Example of recombination by GAPX. The edges of the first and second parents are respectively represented by blue and red solid lines. The common edges are represented by dark solid lines. After removing the common edges from the union graph (a), two candidate components are found. The simplified graphs for the candidate component indicated by the dashed line are equal for both parents (b). In order to create the simplified graphs, the indices of the entries and exits in the component are identified (c).

III. Fusion of Infeasible Candidate Components

When one candidate component is not a recombining component, it can be fused to another infeasible component to

[1]Both are $O(n)$ when applied to solutions of the TSP.

generate a recombining component. In the procedure presented here, fusion occurs between two infeasible candidate components that are neighbors. Two candidate component are neighbors if they are linked by a path composed only of common edges. When a candidate component a is fused to a candidate component b, it means that the paths inside a and b are inherited from the same parent. Here, after fusing two neighbor candidate components, the new candidate component is tested to check if it is a recombining component. Testing a new candidate component is done by checking if the simplified graphs inside the component are equal for both parents.

There are many ways to fuse neighbor candidate components. Trying all combinations of infeasible partitions can be very time consuming. A heuristic procedure is adopted here to maintain $O(n)$ runtime costs. Only one fusion is made for each infeasible component. The fusions occurs between neighboring infeasible candidate components that share the most common paths. However, fusions are made only between candidate components that are neighbors of at most two other infeasible candidate components. Successful fusions, i.e., fusions that result in recombination components, are more likely if the fusion is done between components that do not have many neighbors. Counting the number of common paths between infeasible partitions is done at $O(n)$ cost if the number of neighbors is limited to a small constant. The heuristic fusion procedure is given by:

Fusion Procedure:

i) *Identify the infeasible candidate components that are neighbors of at most two other infeasible candidate components*;

ii *Perform a cycle of fusions between pairs of infeasible candidate components identified in step i) with more common paths between them*;

iii *Test if the resulting candidate components are recombining components. This is done by checking if the simplified graphs inside the component are equal for both parents.*

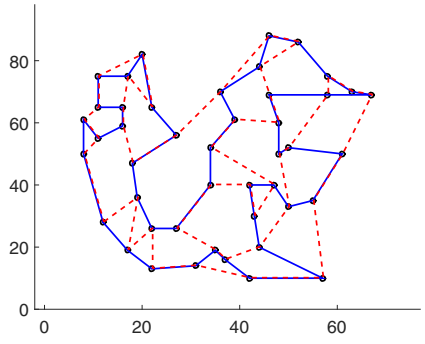

Fig. 2. Union graph G_u formed by two solutions (blue and red tours).

Figures 2 and 3 show an example of fusion. Figure 2 shows the graph G_u obtained by the union of parents G_1 and G_2 (blue and red tours). After removing the common edges, 3 connected components can be identified (Figure 3). Candidate component 1 is a recombining component. Candidate components 2 and 3 are not feasible for recombination because

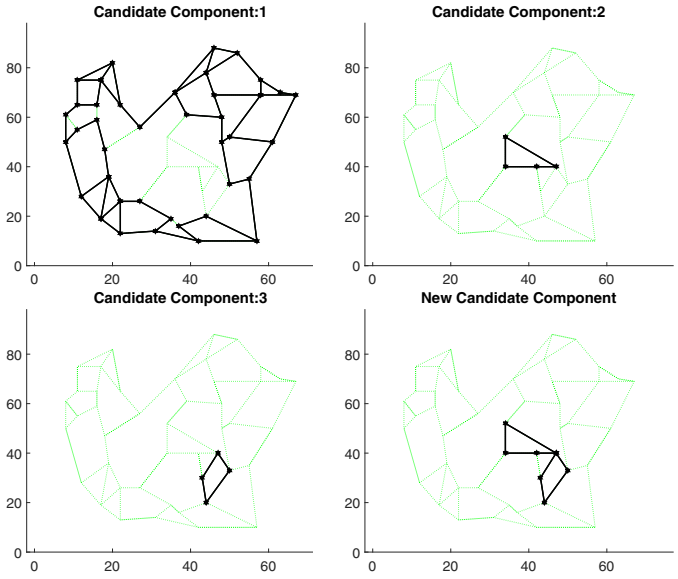

Fig. 3. Three candidate components are found after removing the common edges of G_u. The candidate component 1 is a recombining component while candidate components 2 and 3 are not. However, the new candidate candidate formed by the fusion of candidates 2 and 3 is a recombining component.

the corresponding entries and exits in each component are different for the parent tours, i.e, the simplified graphs for the parents are different. When fusion is applied, candidate components 2 and 3 are merged into a new candidate component. The simplified graphs for the new candidate component are equal, i.e, the new candidate component is a recombining component. Before fusion, the number of recombining components is $k = 2$ (candidate component 1 and the rest of the graph); therefore the best of $2^2 = 4$ reachable offspring is found. After fusion, the number of recombining components is $k = 3$ (candidate component 1, the new candidate component, and the rest of the graph); therefore the best of $2^3 = 8$ reachable offspring is found.

Fusions occur between two candidate components. A candidate component infeasible for recombination can participate in only one fusion, but if the new candidate component is still infeasible for recombination, it can be merged to another candidate component. Larger components can be obtained by repeating the fusion procedure. For example, one fusion can occur first between components a and b; if the resulting new component is not a true recombining component, it can be fused with component c. The number of fusions performed is limited to n_f, that is less or equal to the number of infeasible components identified before. Algorithm 1 shows the pseudocode of GAPX with fusion. This results in the following theorems.

Theorem 1. *Recombining components generated by fusion yields Hamiltonian cycles.*

Proof. The proof is based on that one presented in [4]. After removing the common edges from the union graph, the connected components are found. Thus, entry and exit

vertices of a candidate component are entries or exits for both Hamiltonian cycles (parents). However, the order in which the entry/exit pairs are visited can be different for the paths given by each parent. When candidate components are fused, some of the entries and exits disappear and the tour inside the new component is inherited from only one parent.

We can split the Hamiltonian tour for each parent in two sets of paths: one for paths inside and other for paths outside the new component. Let $G_{in}^{p_1} = (V_s, E_{in}^{p_1})$ and $G_{in}^{p_2} = (V_s, E_{in}^{p_2})$ denote simplified graphs respectively for parents p_1 and p_2. The set V_s contains the entry/exit vertices of the new component and $E_{in}^{p_i}$ are edges representing the paths inside the component for parent p_i. Let $G_{out}^{p_1} = (V_s, E_{out}^{p_1})$ and $G_{out}^{p_2} = (V_s, E_{out}^{p_2})$ denote the simplified graphs for the parents with the edges representing paths **outside** the component. The set $E_{out}^{p_i}$ contains edges representing the path outside the component for parent p_i.

We consider that recombination is applied and two offspring are generated each time combining each new candidate component with the rest of the graph. The combined simplified graphs in the two offspring o_i are $G_s^{o_1} = (V_s, E_{in}^{p_1} \cup E_{out}^{p_2})$ and $G_s^{o_2} = (V_s, E_{in}^{p_2} \cup E_{out}^{p_1})$. GAPX considers that a candidate component is a recombining component only if the simplified graphs inside the component are equal for both parents. In other words, $E_{in}^{p_1} = E_{in}^{p_2}$. Then the combined graphs of the offspring are equal to the simplified combined graphs for the parents, i.e., $G_s^{p_1} = G_{in}^{p_1} \cup G_{out}^{p_1}$ and $G_s^{p_2} = G_{in}^{p_2} \cup G_{out}^{p_2}$. Since the simplified combined graphs for both parents are Hamiltonian cycles, the offspring generated in this way are also Hamiltonian cycles. □

Theorem 2. *The time complexity of GAPX with fusion is* $O(n)$.

Proof. Finding candidate components, identifying recombining components, and generating offspring in GAPX is $O(n)$ [4]. Fusion is limited to infeasible candidate components with at most two neighbors. Therefore, it is necessary to count only the number of common paths between each component and at most two neighbors. This can be done visiting each vertex of the graph, resulting in a procedure with cost $O(n)$. Verifying if the new candidate components are recombining components is also $O(n)$ [4]. Repeating fusion n_f times, where n_f is a small constant, does not alter the complexity of the algorithm. Fusion does not impact in the time complexity of the other procedures of GAPX. Thus, GAPX with fusion is $O(n)$. □

IV. Experiments

Two sets of experiments are presented. First, GAPX is used with a hybrid Genetic Algorithm. GAPX is compared to another efficient recombination operator for the TSP in these experiments. Then, GAPX is used to recombine local optima generated by multi-trial LKH. Multi-trial LKH uses a crossover operator, called *Iterative Partial Transcription* (IPT), to recombine solutions generated in different trials of the LK heuristic. Thus, an eventual improvement in the solutions generated by LKH means that GAPX is capable of finding recombination opportunities missed by IPT. In the experiments, $n_f = 5$ fusions.

Algorithm 1 GAPX with fusion

Step 1. Insert a ghost vertex immediately after each vertex with degree 4 in both parents. The graphs G_1 and G_2 (parents) with the ghost vertices are denoted G_1' and G_2'.
Step 2. Create a union graph $G_u' = G_1' \cup G_2'$.
Step 3. Remove all common edges of G_u'.
Step 4. Find the connected components of G_u'. The connected components with more than one node are candidate components for recombination.
Step 5. For each candidate components, create 2 simplified directed graphs, one for each parent. The simplified graphs contain as vertices only the entry and exit vertices of the candidate component. Replace the path between each entry and respective exit vertex by a single edge for each component. If the two simplified graphs are equal, then the candidate component is a recombining component.
for $i = 1$ **to** n_f **do**
  Step 6.a. Identify the infeasible candidate components that are neighbors of at most two other infeasible candidate components.
  Step 6.b. Perform the fusion between the pairs of candidate components with more common paths between them.
  Step 6.c. Apply Step 5 to verify if the new candidate components are recombining components.
end for
Step 7. The remaining graph is also considered a recombining component.
Step 8. Apply crossover by selecting the shortest path inside each recombining component.

A. Hybrid Genetic Algorithm

Here, we apply GAPX with fusion as a recombining operator inside a hybrid *Genetic Algorithm* (GA) for the ATSP. The hybrid GA uses local search based on the operator 3-opt (Algorithm 2).

In Algorithm 2, local search is applied to random solutions in order to generate the initial population. Elitism is used to preserve the current best solution and tournament selection is used to select parents for crossover. Mutation is applied when crossover does not improve the best solution. When mutation is applied, the parent is transformed by: a 2-opt move, a double bridge move, or local search. The three types of mutation have the same probability of occurrence. In order to reduce the computational time of local search, 3-opt moves is limited to a fixed number of neighbor vertices (limited neighborhood size). Local search still uses "don't look bits" and "first found solution" strategies [9]. If the best solution is not improved in the last 20 generations, the neighborhood size is increased in local search. The neighborhood size starts in 10 cities and increases up to 150 cities or n (if it is smaller than 150). All solutions, except for the best solution, are replaced by random solutions optimized by local search (immigration). The population size is 300 individuals, the tournament selection parameter is equal to 0.8, and the number of runs is equal to 25. Each run stops if the optimal solution is found or if a maximum number of 1500 generations is reached.

Algorithm 2 Hybrid GA

```
P=popInit();
while termination condition is not satisfied do
  Q(1)=bestSolution(P);
  for i=2 to maxpop do
    (p1,p2)=selection(P);
    Q(i)=crossover(p1,p2);
    if crossover did not improve the solutions then
      Q(i)=mutation(Q(i));
    end if
  end for
  if best sol. did not improve in last 20 gen. then
    increase the neighborhood size used in local search based on
    3-opt;
    Q=immigration();
  end if
  P=Q;
end while
```

The GA with GAPX is here compared to 2 other approaches. All algorithms are equal, with the exception of using different crossover operators. The crossover operators are:

i) *same-site-copy-first crossover* (SSCFX) [10], that is similar to the partially matched crossover (but keeping the cities that occupy the same site in both parent solutions) and was used in the evolutionary algorithm in [11].
ii) *Generalized partition crossover* (GPX) adapted to the ATSP. In this partition crossover, only recombining components with 2 crossover points are identified. Vertices with degree 4 are not considered as possible crossover points and the fusion procedure is not employed. However, unlike the original GPX [7], the common edges are considered according to the flow direction in order to not invert the edges in the ATSP.
iii) GAPX with fusion (Section III).

Table I presents the obtained results for the instances of the TSPLIB. In Table I, $succ$ indicates the percentage of runs where the global optimum was successfully found, μ-err is the average percentage excess with respect to the global optimum evaluation, and μ-T is the average computation time for each run in seconds[2]. Some observations can be made from the results.

First, when compared to the GA with SSCFX, the GA with GAPX with fusion shows much better results. GAPX with fusion is capable of producing new optima from two optimal solutions, allowing the algorithm to reach the global optima in the experiments performed here. When we analyze the final evaluation produced by the two algorithms, it is possible to observe that the GA with SSCFX was often not able to escape from some local optima.

Second, we can observe that GAPX with fusion produced better results than the GPX adapted to the ATSP. Despite the similarities between the operation of the algorithms, GAPX with fusion is capable of exploring many more solutions than the adapted GPX. This occurs because GAPX with fusion exploits more recombining opportunities than the adapted GPX. As a consequence, many more possible offspring is exploited in each recombination. When compared to GAPX without fusion [4], the version of GAPX with fusion is able to find the global optima faster more often. This occurs because GAPX with fusion finds more recombining components (see experiments in next section) and thus more improvements.

[2]The experiments were performed on a computer with processor Core 2 Quad 2.83 GHz

TABLE I
RESULTS FOR THE HYBRID GA WITH: SAME-SITE-COPY-FIRST CROSSOVER (SSCFX) [10]; ADAPTED GPX; GAPX WITH FUSION.

Instance	SSCFX [10]			adapted GPX			GAPX with fusion		
	$succ$	μ-err	μ-T	$succ$	μ-err	μ-T	$succ$	μ-err	μ-T
br17	100	0.000	0.02	100	0.000	0.03	100	0.000	0.03
ftv33	100	0.000	0.36	100	0.000	0.34	100	0.000	0.22
ftv35	88	0.016	24.55	92	0.011	23.00	100	0.000	0.51
ftv38	52	0.063	57.87	92	0.010	45.05	100	0.000	1.14
p43	100	0.000	16.31	100	0.000	16.05	100	0.000	7.13
ftv44	100	0.000	4.34	100	0.000	1.42	100	0.000	0.67
ftv47	76	0.081	76.32	100	0.000	28.67	100	0.000	2.89
ry48p	52	0.265	114.08	100	0.000	29.79	100	0.000	2.58
ft53	100	0.000	33.15	100	0.000	27.83	100	0.000	3.72
ftv55	100	0.000	21.06	100	0.000	11.70	100	0.000	1.21
ftv64	92	0.065	70.72	100	0.000	57.66	100	0.000	2.37
ft70	64	0.032	242.90	92	0.004	169.61	100	0.000	30.30
ftv70	72	0.068	249.84	64	0.074	313.30	100	0.000	8.00
kro124p	56	0.124	954.02	92	0.002	606.14	100	0.000	13.39
ftv170	44	0.501	4600.13	40	0.672	5270.10	100	0.000	624.73
rbg323	100	0.000	32.05	100	0.000	39.13	100	0.000	45.66
rbg358	100	0.000	390.60	100	0.000	432.25	100	0.000	467.59
rbg403	100	0.000	74.23	100	0.000	80.05	100	0.000	108.19
rbg443	100	0.000	31.83	100	0.000	26.97	100	0.000	37.52

B. Improving solutions generated by multi-trial LKH

This section demonstrates the ability of GAPX with fusion to improve solutions generated by multi-trial LKH. The multi-trial LKH has improved solutions of several large TSPs with unknown optima, including symmetric TSPs with almost 2 million cities. Despite being an approximate algorithm, LKH was able to find the optimal solutions for all instances with known optima tested by Helsgaun [12]. LKH deals with the ATSP by transforming it into a symmetric TSP by doubling the size of the original problem [2].

In the experiments presented in this section, GAPX with fusion is used to improve solutions generated by multi-trial LKH with default parameters [12]. Multi-trial LKH is a form of iterated local search that uses a "kick" operator to escape local optima. Multi-trial LKH uses the recombination operator IPT in two different ways. First, IPT is used to recombine the local optimum found after the kick (trial) with the current best solution found in the run. Then, it is used to recombine solutions produced in different runs.

In this last case, the best solution found in a run is recombined with the best solution found until the previous run. In the experiments presented here, we save the best solution found in each run of multi-trial LKH after it was recombined by IPT with the best solution so far if this improves the current best solution. Otherwise, we save the best solution of the run before the recombination. We then apply GAPX with fusion to recombine the solutions produced in different runs by multi-trial LKH. First, GAPX with fusion is applied between the

solutions found in runs 1 and 2 of LKH. The offspring is then recombined with the best local optimum found in run 3 of LKH, and so on. Results of experiments of multi-trial LKH with 20 runs and different numbers of trials are shown here. Therefore, GAPX with fusion is applied 19 times in each experiment.

GAPX with fusion is applied to the best solutions obtained after applying IPT in LKH. Also, the solutions found by GAPX with fusion are not reinserted in LKH. Therefore, improved results are due to the effectiveness of GAPX with fusion when compared to IPT.

We also tested an approach where GAPX with fusion is applied to recombine exhaustively all solutions generated by multi-trial LKH. First, the solution found in the first run of multi-trial LKH is recombined to the other 19 solutions, like in the previous approach. Then, the second solution is recombined to the other 19 solutions. This process is repeated for each remaining solution $\mathbf{x}_i$, recombining the best solution so far to the best solution found by recombining solution $\mathbf{x}_i$ to the other 19 solutions. The best solutions found are then recombined in a sequence (first with second, then with third, and so on). Thus, GAPX with fusion is applied $20^2 - 1$ times. We denote this as *all-to-all approach*, while *incremental approach* denotes the case where crossover is applied only 19 times.

The results of LKH and LKH+GAPX with fusion in instances of 7 classes of ATSPs produced by the benchmark generator proposed in [13] are presented. Twelve randomly generated classes of problems can be produced by the benchmark generator. Instances of seven of them have structure that might be similar to those found in real-world applications. They are: tilted drilling machine instances with additive norm (rtilt), tilted drilling machine instances with sup norm (stilt), random Euclidean stacker crane instances (crane), disk drive instances (disk), pay phone coin collection instances (coin), no-wait flowshop instances (shop), and approximate shortest common superstring instances (super).

We next present results of LKH and LKH+GAPX with fusion applied to three instances with 1000 cities and one with 3162 cities [14] for each one of the seven classes. Table IV shows the results for LKH and GAPX with fusion applied to recombine the LKH solutions (LKH+GAPX with fusion). The results of the percentage excess over the HK bounds are shown. When available, the optimal results are also reported [14]. Table II summarizes the results for all instances. Results for instances shop1000.21, super1000.10, super1000.20, super1000.30, and super3162.30 are not shown in Table IV. However, they are summarized in Table II.

The mean number of recombining components and the maximum component found by GAPX with fusion in the incremental approach are presented in Table III. The table shows results for GAPX for one and five (n_f) fusions. It also shows the mean number of recombining components with 1 entry (and 1 exit) and before fusion is applied. Fusion increases the number of recombining components, resulting in finding the best of a higher number of solutions. Table III shows that the number of recombining components increases when fusion is applied, except for instances of shop1000. Increasing the number of fusions generally results in an increase in the number of recombining components; however, results not shown here indicate that the increase in the number of recombining components becomes each time smaller when the number of fusions increases.

As a consequence of increasing the number of recombining components, GAPX with fusion improved the performance in all instances of the classes rtilt, stilt, crane and coin, with exception of crane1000.22 with 1000 trials (Table IV). The standard deviation is zero for the solutions of the LKH in crane1000.22 with 1000 trials. As all solutions are equal, GAPX with fusion can only generate the offspring equal to the parents.

The low diversity of the solutions also helps to explain the performance of GAPX with fusion in the instances of classes disk, shop and super. The standard deviation is close to zero for the solutions of the LKH in those instances. The mean number of partitions found by GAPX is small for the instances in those classes as well.

TABLE II
SUMMARY OF THE RESULTS. THE NUMBER OF TIMES THAT LKH+GAPX WITH FUSION IMPROVED THE MEAN AND BEST RESULTS OF LKH IS PRESENTED. FOR THE BEST RESULTS, ONLY THE TIMES THAT LKH DID NOT REACHED KNOWN OPTIMUM ARE COUNTED.

Problem	incremental approach		all-to-all approach	
Type	improved mean	improved best	improved mean	improved best
rtilt	12 out of 12	9 out of 12	12 out of 12	10 out of 12
stilt	12 out of 12	9 out of 12	12 out of 12	9 out of 12
crane	11 out of 12	8 out of 12	11 out of 12	8 out of 12
disk	1 out of 12	0 out of 8	4 out of 12	0 out of 10
coin	12 out of 12	9 out of 12	12 out of 12	10 out of 12
shop	4 out of 12	1 out of 11	5 out of 12	2 out of 11
super	0 out of 12	0 out of 9	0 out of 12	0 out of 9

The all-to-all approach resulted in a better performance than the incremental approach. The incremental approach improved the best solution in 35 out of 48 instances of classes rtilt, stilt, crane and coin. The all-to-all approach improved the best in 37 out of 48 instances.

V. CONCLUSIONS

The fusion mechanism increases the average number of recombining components (k) found by GAPX. For example, the mean number of recombining components found by GAPX before fusion was 0.1 in the experiment with LKH for instance crane3612.30 with 1000 trials. After fusion, the mean number increased to 1.5. Except for the experiments where the set of parents has low diversity, fusion resulted in an increase in the number of recombining components found by GAPX. In one experiment with LKH, GAPX was able to find $k = 42$ recombining components in one application of recombination (Section IV-B); this allowed GAPX to find the best of 2^{42} solutions at cost $O(n)$. As a consequence of increasing the number of recombining components, better solutions were found by GAPX with fusion. Experimental results (Section IV-B) indicated that GAPX with fusion can be

applied to improve solutions generated by LKH and other good heuristics. GAPX with fusion can also be successfully applied as a recombination operator inside population metaheuristics. Experimental results (Section IV-A) indicated that GAPX with fusion generated very good results when applied as a recombination operator in a hybrid GA.

The fusion mechanism presented here can be applied to partition crossover operators for the symmetric TSP. Testing the proposed fusion mechanism in partition crossover for the symmetric TSP is an interesting future work. GAPX with fusion was applied to recombine solutions generated by LKH in an offline way, i.e., first LKH generates a set of solutions and then GAPX recombines these solutions to look for additional improvements. In future, we will investigate an approach where GAPX replaces the recombination operator IPT that is used inside LKH. Another future work is the investigation of the use of GAPX to other vehicle routing problems. Finally, other ways of fusion are possible. Particularly, we should investigate fusion mechanisms that can be applied to "nested" candidate components.

REFERENCES

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

[2] K. Helsgaun, "An effective implementation of the Lin-Kernighan traveling salesman heuristic," *European Journal of Oper. Res.*, vol. 126, no. 1, pp. 106–130, 2000.

[3] A. Möbius, B. Freisleben, P. Merz, and M. Schreiber, "Combinatorial optimization by iterative partial transcription," *Phsycal Review E*, vol. 59, no. 4, pp. 4667–4674, 1999.

[4] R. Tinós, D. Whitley, and G. Ochoa, "Generalized asymmetric partition crossover (GAPX) for the asymmetric TSP," in *Proc. of GECCO'2014*, 2014, pp. 501–508.

[5] A. V. Eremeev and Y. V. Kovalenko, "Genetic algorithm with optimal recombination for the asymmetric travelling salesman problem," in *Int. Conf. on Large-Scale Scientific Comp.*, 2017, pp. 341–349.

[6] N. Veerapen, G. Ochoa, R. Tinós, and W. D., "Tunnelling crossover networks for the asymmetric TSP," in *Parallel Problem Solving from Nature XIV. LNCC, vol. 9921*, 2016, pp. 994–1003.

[7] D. Hains, D. Whitley, and A. Howe, "Revisiting the big valley search space structure in the TSP," *Journal of the Oper. Res. Soc.*, vol. 62, no. 2, pp. 305–312, 2011.

[8] R. Tinós, D. Whitley, and F. Chicano, "Partition crossover for pseudo-boolean optimization," in *Proc. of the 2015 ACM Conf. on Foundations of Genetic Alg. XIII*, 2015, pp. 137–149.

[9] Y. Nagata and D. Soler, "A new genetic algorithm for the asymmetric TSP," *Expert Syst. with App.*, vol. 39, no. 10, pp. 8947–8953, 2012.

[10] H.-F. Wang and K.-Y. Wu, "Hybrid genetic algorithm for optimization problems with permutation property," *Computers & Operations Research*, vol. 31, no. 14, pp. 2453–2471, 2004.

[11] L.-N. Xing, Y.-W. Chen, K.-W. Yang, F. Hou, X.-S. Shen, and H.-P. Cai, "A hybrid approach combining an improved genetic algorithm and optimization strategies for the asymmetric traveling salesman problem," *Eng. App. of Art. Int.*, vol. 21, no. 8, pp. 1370–1380, 2008.

[12] K. Helsgaun, "LKH," 2018, http://www.akira.ruc.dk/∼keld/research/LKH/.

[13] J. Cirasella, D. S. Johnson, L. A. McGeoch, and W. Zhang, "The asymmetric traveling salesman problem: algorithms, instance generators, and tests," in *Algorithm Engineering and Experimentation*. Springer, 2001, pp. 32–59.

[14] D. Johnson, L. McGeoch, F. Glover, and C. Rego, "8th DIMACS implementation challenge: The traveling salesman problem," 2013, http://dimacs.rutgers.edu/Challenges/TSP/.

TABLE III
AVERAGE NUMBER OF RECOMBINING COMPONENTS FOUND IN EACH APPLICATION OF GAPX (INCREMENTAL APPROACH). THE MAXIMUM NUMBER OF RECOMBINING COMPONENTS FOUND IN ONE APPLICATION OF GAPX IS SHOWN IN THE LAST COLUMN.

Instance	trials	1 entry	before fusion	1 fusion	5 fusions	max.
rtilt1000.20	1	0.9 ±0.3	1.2 ±0.3	1.4 ±0.4	1.8 ±0.4	6
	20	1.0 ±0.3	1.3 ±0.2	1.5 ±0.3	2.0 ±0.3	6
	1000	0.9 ±0.4	1.0 ±0.4	1.2 ±0.4	1.6 ±0.5	6
rtilt1000.21	1	1.1 ±0.4	1.3 ±0.5	1.6 ±0.5	1.9 ±0.6	9
	20	1.1 ±0.3	1.4 ±0.3	1.6 ±0.4	2.0 ±0.4	7
	1000	1.0 ±0.3	1.2 ±0.3	1.4 ±0.3	1.8 ±0.4	7
rtilt1000.22	1	0.7 ±0.3	1.0 ±0.4	1.2 ±0.5	1.6 ±0.5	7
	20	1.2 ±0.3	1.5 ±0.3	1.8 ±0.3	2.2 ±0.2	7
	1000	1.0 ±0.3	1.1 ±0.3	1.4 ±0.4	1.8 ±0.4	6
rtilt3162.30	1	3.6 ±0.8	4.2 ±0.9	4.3 ±1.0	4.4 ±1.0	13
	20	4.5 ±0.7	5.0 ±0.8	5.2 ±0.8	5.3 ±0.8	16
	1000	4.3 ±0.6	4.6 ±0.6	4.7 ±0.6	4.9 ±0.6	13
stilt1000.20	1	1.2 ±0.4	1.6 ±0.5	1.8 ±0.5	2.1 ±0.6	8
	20	1.2 ±0.4	1.7 ±0.3	1.9 ±0.4	2.3 ±0.6	7
	1000	1.0 ±0.2	1.1 ±0.3	1.2 ±0.3	1.5 ±0.3	5
stilt1000.21	1	0.7 ±0.2	1.0 ±0.4	1.2 ±0.4	1.5 ±0.5	7
	20	0.8 ±0.3	1.3 ±0.5	1.6 ±0.6	1.9 ±0.6	8
	1000	0.9 ±0.2	1.0 ±0.2	1.4 ±0.3	1.8 ±0.4	5
stilt1000.22	1	1.9 ±0.6	2.3 ±0.7	2.6 ±0.8	2.9 ±0.7	10
	20	2.3 ±0.6	2.7 ±0.7	3.2 ±0.8	3.6 ±0.8	9
	1000	2.4 ±0.6	2.7 ±0.7	2.8 ±0.6	3.5 ±0.7	9
stilt3162.30	1	3.4 ±0.8	4.2 ±1.0	4.7 ±1.1	4.9 ±1.2	13
	20	4.1 ±0.5	5.0 ±0.5	5.5 ±0.5	5.6 ±0.5	14
	1000	4.0 ±0.4	4.6 ±0.4	4.7 ±0.4	4.9 ±0.4	13
crane1000.20	1	0.3 ±0.1	0.4 ±0.2	0.9 ±0.3	1.4 ±0.4	5
	20	0.1 ±0.1	0.1 ±0.1	0.7 ±0.2	1.3 ±0.3	4
	1000	0.0 ±0.0	0.0 ±0.0	0.3 ±0.2	0.5 ±0.3	3
crane1000.21	1	0.3 ±0.1	0.6 ±0.3	1.2 ±0.3	1.7 ±0.3	7
	20	0.4 ±0.1	0.7 ±0.3	2.1 ±0.7	2.6 ±0.7	7
	1000	0.0 ±0.0	0.0 ±0.0	0.2 ±0.1	0.9 ±0.2	3
crane1000.22	1	0.2 ±0.1	0.4 ±0.1	0.9 ±0.1	1.5 ±0.3	6
	20	0.1 ±0.1	0.3 ±0.2	0.5 ±0.2	1.2 ±0.5	5
	1000	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.1 ±0.1	2
crane3162.30	1	0.9 ±0.5	1.6 ±0.9	2.7 ±0.9	3.5 ±0.9	10
	20	0.8 ±0.3	1.6 ±0.3	2.8 ±0.5	3.7 ±0.5	11
	1000	0.1 ±0.0	0.1 ±0.0	0.9 ±0.2	1.5 ±0.2	5
disk1000.20	1	0.6 ±0.2	0.9 ±0.2	1.3 ±0.3	1.4 ±0.3	5
	20	0.4 ±0.2	0.9 ±0.2	1.2 ±0.2	1.3 ±0.3	4
	1000	0.4 ±0.2	0.9 ±0.3	1.4 ±0.4	1.4 ±0.4	4
disk1000.21	1	0.4 ±0.2	1.3 ±0.4	1.9 ±0.5	1.9 ±0.5	6
	20	0.6 ±0.1	1.7 ±0.3	2.3 ±0.2	2.3 ±0.2	6
	1000	0.6 ±0.2	1.7 ±0.2	2.8 ±0.3	2.8 ±0.3	7
disk1000.22	1	0.0 ±0.1	0.1 ±0.1	0.5 ±0.1	0.5 ±0.1	3
	20	0.0 ±0.0	0.1 ±0.1	0.5 ±0.2	0.5 ±0.2	3
	1000	0.0 ±0.0	0.1 ±0.1	0.7 ±0.2	0.7 ±0.2	4
disk3162.30	1	0.4 ±0.1	0.9 ±0.3	1.5 ±0.3	1.5 ±0.3	5
	20	0.5 ±0.3	1.1 ±0.5	1.8 ±0.6	1.8 ±0.6	7
	1000	0.5 ±0.2	1.1 ±0.3	1.6 ±0.5	1.6 ±0.5	6
coin1000.20	1	5.4 ±0.6	5.8 ±0.7	6.0 ±0.8	6.2 ±0.8	18
	20	9.0 ±1.2	9.8 ±1.3	10.2 ±1.3	10.3 ±1.3	19
	1000	10.9 ±1.4	11.5 ±1.4	11.8 ±1.6	12.0 ±1.7	26
coin1000.21	1	5.9 ±0.5	6.3 ±0.7	6.5 ±0.7	6.6 ±0.8	17
	20	8.4 ±0.7	9.4 ±0.8	9.6 ±0.8	9.6 ±0.8	18
	1000	9.7 ±0.8	10.6 ±0.8	10.8 ±0.8	10.8 ±0.8	21
coin1000.22	1	5.0 ±0.9	5.8 ±1.0	6.1 ±1.0	6.2 ±1.0	13
	20	6.9 ±0.8	8.1 ±1.0	8.6 ±0.9	8.8 ±0.9	18
	1000	9.9 ±1.1	10.6 ±1.0	10.9 ±1.0	11.0 ±1.0	21
coin3162.30	1	14.2 ±2.5	15.5 ±2.7	16.0 ±2.9	16.1 ±2.9	35
	20	18.8 ±1.6	20.7 ±1.6	21.4 ±1.6	21.5 ±1.6	38
	1000	23.0 ±1.5	24.7 ±1.6	25.1 ±1.7	25.2 ±1.7	42
shop1000.20	1	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	2
	20	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	1
	1000	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.1	2
shop1000.22	1	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.1 ±0.1	2
	20	0.0 ±0.0	0.1 ±0.1	0.1 ±0.1	0.1 ±0.1	2
	1000	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	1
shop3162.30	1	0.0 ±0.1	0.0 ±0.1	0.0 ±0.1	0.1 ±0.2	2
	20	0.0 ±0.1	0.0 ±0.1	0.0 ±0.1	0.1 ±0.1	2
	1000	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	1

TABLE IV

PERCENTAGE EXCESS OVER THE HK BOUNDS FOR LKH AND LKH+GAPX WITH FUSION. RESULTS FOR RUNNING LKH WITH 3 DIFFERENT NUMBERS OF MULTI-TRIALS (DENOTE BY TRIALS = 1, 20, OR 1000) ARE SHOWN. THE SYMBOLS '=' AND '+' RESPECTIVELY INDICATES THAT THE MEAN FOR LKH+GAPX WITH FUSION IS EQUAL OR BETTER THAN THE MEAN FOR LKH. THE LETTER *s* INDICATES THAT THE RESULTS ARE STATISTICALLY SIGNIFICANT. FOR THE STATISTICAL COMPARISONS, THE NON-PARAMETRIC WILCOXON SIGNED-RANK TEST WITH SIGNIFICANCE LEVEL 0.05 WAS USED.

Instance	Trials	LKH		incremental approach		all-to-all approach		optimum
		mean±std	best	mean±std	best	mean±std	best	
rtilt1000.20	1	2.5474 ±0.2228	2.0884	2.4553 ±0.2714 (s+)	1.9104	2.3658 ±0.2514 (s+)	1.9104	
n=1000	20	1.6157 ±0.1186	1.3809	1.5449 ±0.1016 (s+)	1.3472	1.4612 ±0.1300 (s+)	1.2434	
	1000	0.8807 ±0.0545	0.7928	0.8695 ±0.0554 (+)	0.7928	0.8338 ±0.0460 (s+)	0.7608	0.7279
rtilt1000.21	1	2.3521 ±0.2194	2.0720	2.2202 ±0.2406 (s+)	1.8444	2.0875 ±0.1972 (s+)	1.7711	
n=1000	20	1.5390 ±0.1611	1.1765	1.4523 ±0.2086 (s+)	1.1311	1.4348 ±0.1720 (s+)	1.1752	
	1000	0.7760 ±0.0711	0.6480	0.7714 ±0.0702 (+)	0.6480	0.7471 ±0.0570 (s+)	0.6480	0.6305
rtilt1000.22	1	2.7322 ±0.1958	2.4206	2.5419 ±0.1853 (s+)	2.2221	2.5271 ±0.1918 (s+)	2.1668	
n=1000	20	1.6287 ±0.1774	1.2892	1.4730 ±0.1926 (s+)	1.2357	1.3026 ±0.2117 (s+)	0.9613	
	1000	0.8095 ±0.0392	0.7399	0.8081 ±0.0367 (+)	0.7399	0.7950 ±0.0296 (+)	0.7399	0.6947
rtilt3162.30	1	3.0119 ±0.1187	2.7584	2.9687 ±0.1244 (s+)	2.7511	2.9133 ±0.1298 (s+)	2.7391	
n=3162	20	2.3904 ±0.1289	2.1716	2.3384 ±0.1208 (s+)	2.1389	2.2922 ±0.1332 (s+)	2.0401	
	1000	1.5395 ±0.1215	1.3670	1.5022 ±0.1334 (s+)	1.3266	1.4874 ±0.1274 (s+)	1.3266	-
stilt1000.20	1	3.7366 ±0.2727	3.2841	3.5762 ±0.2803 (s+)	3.2017	3.5294 ±0.2808 (s+)	3.1114	
n=1000	20	2.6565 ±0.1233	2.4900	2.5770 ±0.1530 (s+)	2.4225	2.5213 ±0.1422 (s+)	2.3812	
	1000	2.0870 ±0.0198	2.0539	2.0854 ±0.0200 (+)	2.0539	2.0812 ±0.0169 (+)	2.0539	-
stilt1000.21	1	3.7388 ±0.3120	3.3284	3.5092 ±0.2019 (s+)	3.2385	3.3831 ±0.2341 (s+)	3.1072	
n=1000	20	2.6064 ±0.1620	2.2597	2.4198 ±0.1411 (s+)	2.1433	2.3637 ±0.1358 (s+)	2.1433	
	1000	1.8229 ±0.0151	1.8033	1.8104 ±0.0127 (+)	1.8033	1.8104 ±0.0127 (+)	1.8033	-
stilt1000.22	1	3.9330 ±0.2520	3.6078	3.7716 ±0.2682 (s+)	3.4583	3.6707 ±0.3153 (s+)	3.1827	
n=1000	20	2.9457 ±0.1870	2.5670	2.8630 ±0.1677 (s+)	2.5670	2.7687 ±0.1235 (s+)	2.5670	
	1000	2.3678 ±0.0584	2.2756	2.3466 ±0.0609 (s+)	2.2618	2.3261 ±0.0692 (s+)	2.2390	-
stilt3162.30	1	4.2387 ±0.2291	3.7406	4.1670 ±0.2214 (s+)	3.6423	4.0775 ±0.2511 (s+)	3.4880	
n=3162	20	3.3138 ±0.1312	3.0390	3.2258 ±0.1227 (s+)	3.0191	3.1847 ±0.1200 (s+)	2.9896	
	1000	2.4119 ±0.0762	2.3129	2.3813 ±0.0958 (s+)	2.2129	2.3590 ±0.0884 (s+)	2.2129	-
crane1000.20	1	1.2684 ±0.0624	1.2097	1.2374 ±0.0431 (+)	1.1758	1.2118 ±0.0311 (s+)	1.1405	
n=1000	20	1.1626 ±0.0164	1.1427	1.1519 ±0.0204 (+)	1.1289	1.1451 ±0.0233 (s+)	1.1289	
	1000	1.0883 ±0.0025	1.0838	1.0846 ±0.0018 (s+)	1.0838	1.0846 ±0.0018 (s+)	1.0838	-
crane1000.21	1	1.2631 ±0.0973	1.1410	1.2054 ±0.1009 (s+)	1.0592	1.1731 ±0.0801 (s+)	1.0727	
n=1000	20	1.0547 ±0.0241	1.0257	1.0496 ±0.0246 (s+)	1.0225	1.0408 ±0.0070 (s+)	1.0225	
	1000	1.0140 ±0.0012	1.0132	1.0132 ±0.0000 (+)	1.0132	1.0132 ±0.0000 (+)	1.0132	-
crane1000.22	1	1.0660 ±0.0664	0.9861	1.0236 ±0.0370 (s+)	0.9801	1.0035 ±0.0319 (s+)	0.9613	
n=1000	20	0.9555 ±0.0175	0.9258	0.9425 ±0.0184 (s+)	0.9220	0.9332 ±0.0093 (s+)	0.9220	
	1000	0.9220 ±0.0000	0.9220	0.9220 ±0.0000 (=)	0.9220	0.9220 ±0.0000 (=)	0.9220	-
crane3162.30	1	1.3599 ±0.1060	1.1793	1.2868 ±0.1102 (s+)	1.1334	1.2407 ±0.0925 (s+)	1.0997	
n=3162	20	1.0513 ±0.0386	0.9831	1.0054 ±0.0383 (s+)	0.9558	0.9928 ±0.0327 (s+)	0.9347	
	1000	0.8915 ±0.0029	0.8888	0.8890 ±0.0014 (s+)	0.8868	0.8888 ±0.0000 (s+)	0.8888	-
disk1000.20	1	0.0721 ±0.0289	0.0282	0.0721 ±0.0289 (=)	0.0282	0.0696 ±0.0278 (+)	0.0282	
n=1000	20	0.0139 ±0.0043	0.0093	0.0139 ±0.0043 (=)	0.0093	0.0135 ±0.0042 (+)	0.0093	
	1000	0.0024 ±0.0000	0.0024	0.0024 ±0.0000 (=)	0.0024	0.0024 ±0.0000 (=)	0.0024	0.0024
disk1000.21	1	0.0868 ±0.0215	0.0590	0.0866 ±0.0216 (+)	0.0590	0.0863 ±0.0220 (+)	0.0590	
n=1000	20	0.0369 ±0.0033	0.0332	0.0369 ±0.0033 (=)	0.0332	0.0369 ±0.0033 (=)	0.0332	
	1000	0.0319 ±0.0021	0.0291	0.0319 ±0.0021 (=)	0.0291	0.0319 ±0.0021 (=)	0.0291	0.0291
disk1000.22	1	0.0425 ±0.0207	0.0259	0.0425 ±0.0207 (=)	0.0259	0.0425 ±0.0207 (=)	0.0259	
n=1000	20	0.0026 ±0.0024	0.0018	0.0026 ±0.0024 (=)	0.0018	0.0026 ±0.0024 (=)	0.0018	
	1000	0.0018 ±0.0000	0.0018	0.0018 ±0.0000 (=)	0.0018	0.0018 ±0.0000 (=)	0.0018	0.0018
disk3162.30	1	0.0909 ±0.0133	0.0727	0.0909 ±0.0133 (=)	0.0727	0.0906 ±0.0130 (+)	0.0727	
n=3162	20	0.0194 ±0.0076	0.0081	0.0194 ±0.0076 (=)	0.0081	0.0194 ±0.0076 (=)	0.0081	
	1000	0.0052 ±0.0000	0.0052	0.0052 ±0.0000 (=)	0.0052	0.0052 ±0.0000 (=)	0.0052	-
coin1000.20	1	2.8630 ±0.1134	2.7372	2.8091 ±0.0854 (s+)	2.6373	2.7652 ±0.0838 (s+)	2.6174	
n=1000	20	2.0701 ±0.1103	1.9583	2.0222 ±0.1241 (s+)	1.8784	1.9683 ±0.0654 (s+)	1.8584	
	1000	1.5189 ±0.0489	1.4590	1.5009 ±0.0370 (+)	1.4590	1.4909 ±0.0286 (s+)	1.4590	-
coin1000.21	1	2.8179 ±0.1911	2.5301	2.7147 ±0.1736 (s+)	2.4904	2.6611 ±0.2211 (s+)	2.2522	
n=1000	20	1.8432 ±0.1541	1.5772	1.7182 ±0.1136 (s+)	1.5574	1.6328 ±0.1176 (s+)	1.4978	
	1000	1.3827 ±0.0182	1.3589	1.3768 ±0.0272 (+)	1.3390	1.3668 ±0.0251 (s+)	1.3390	-
coin1000.22	1	2.7682 ±0.2919	2.2497	2.6589 ±0.2807 (s+)	2.2497	2.6192 ±0.2867 (s+)	2.0908	
n=1000	20	1.7212 ±0.1274	1.5146	1.5127 ±0.1057 (s+)	1.2961	1.4848 ±0.0818 (s+)	1.2762	
	1000	1.1153 ±0.0288	1.0776	1.1014 ±0.0226 (+)	1.0776	1.0955 ±0.0147 (+)	1.0776	-
coin3162.30	1	3.3497 ±0.1046	3.2044	3.2399 ±0.1386 (s+)	3.0521	3.2076 ±0.1190 (s+)	3.0204	
n=3162	20	2.5820 ±0.0722	2.4812	2.4812 ±0.0882 (s+)	2.3733	2.4285 ±0.0723 (s+)	2.3289	
	1000	1.7110 ±0.0629	1.6247	1.6723 ±0.0567 (s+)	1.5867	1.6641 ±0.0507 (s+)	1.5993	-
shop1000.20	1	0.0288 ±0.0066	0.0208	0.0281 ±0.0058 (+)	0.0208	0.0281 ±0.0058 (+)	0.0208	
n=1000	20	0.0162 ±0.0025	0.0122	0.0152 ±0.0025 (+)	0.0108	0.0150 ±0.0023 (+)	0.0108	
	1000	0.0097 ±0.0003	0.0092	0.0097 ±0.0003 (+)	0.0092	0.0097 ±0.0003 (+)	0.0092	0.0091
shop1000.22	1	0.0196 ±0.0054	0.0114	0.0196 ±0.0054 (=)	0.0114	0.0187 ±0.0044 (+)	0.0114	
n=1000	20	0.0098 ±0.0013	0.0082	0.0097 ±0.0014 (+)	0.0082	0.0097 ±0.0014 (+)	0.0081	
	1000	0.0052 ±0.0007	0.0048	0.0052 ±0.0007 (=)	0.0048	0.0052 ±0.0007 (=)	0.0048	0.0048
shop3162.30	1	0.0640 ±0.0046	0.0543	0.0640 ±0.0046 (=)	0.0543	0.0640 ±0.0046 (=)	0.0543	
n=3162	20	0.0506 ±0.0017	0.0476	0.0506 ±0.0017 (=)	0.0476	0.0506 ±0.0017 (=)	0.0476	
	1000	0.0411 ±0.0014	0.0387	0.0411 ±0.0014 (=)	0.0387	0.0411 ±0.0014 (=)	0.0387	-