Machine Learning and Big Data Techniques for Parallel Local Search Metaheuristics

Thé Van Luong and Éric D. Taillard

University of Applied Sciences of Western Switzerland, HEIG-VD, Department of Information and Communication Technologies

Abstract

An unsupervised machine learning method based on association rule is studied for the Quadratic Assignment Problem. Parallel itemsets and local search algorithms are proposed. The extraction of frequent itemsets in the context of local search is shown to produce good results for a few problem instances. Negative results of the proposed learning mechanism are reported for other instances. This result contrasts with other hard optimization problems for which efficient learning processes are known in the context of local search.

Keywords: machine learning; big data; metaheuristics;

1. Introduction

In the past few years, big data has captured the attention of analysts and researchers since there is a strong demand to analyze large data collected from monitoring systems to understand behaviors and identify hidden trends. Science, business, industry, government and society have already undergone a change with the influence of big data. In [1], the authors are exposing opportunities and challenges that represent big data analytics.

On the one hand, with the increase of computational power, machine learning has emerged as the leading research field in artificial intelligence for dealing with big data and more generally with data science [2]. Machine learning techniques

[1] the-van.luong@heig-vd.ch, eric.taillard@heig-vd.ch

have given rise to huge societal impacts in a wide range of applications such as computer vision, natural language understanding and health.

On the other hand, metaheuristics such as genetic algorithms or local search are iterative methods in operations research that have been successfully applied to solve hard combinatorial optimization problems in the past. One of their main goals is to support decision-making processes in complex scenarios and provide near-optimal solutions to industrial problems.

The hybridization of metaheuristics with machine learning techniques is a promising research field for the operations research community [3]. The major interest in using machine learning techniques is to extract useful knowledge from the history of the search in order to improve the efficiency and the effectiveness of a metaheuristic [4].

This paper focuses on the association rule learning, which is an unsupervised machine learning method for discovering interesting relations between variables in very large databases [5]. Agrawal et al. [6] proposed frequent itemset mining for discovering similarities between products in a large-scale transaction data for supermarket chain stores. Initially designed for data mining, finding association rules is now widely generalized in many fields including web research, intrusion detection and bioinformatics.

We propose to incorporate the extraction of frequent itemsets in the context of local search metaheuristics. A similar work comes from Ribeiro et Al. in [7] to improve a GRASP metaheuristic where the learning process consists of extracting different patterns (i.e. subsets of frequent itemsets) from an elite set of 10 solutions and takes a few seconds to provide a new generation.

The motivation of our work goes further, and its application is more appropriate to a big data context with gigabytes of data. The goal is to investigate if one can learn anything from the execution of thousands of local search algorithms to generate new sets of improved solutions. Hence, we propose reproducible strategies based on the extraction of millions frequent itemsets, i.e. extending the training phase to last one day and considering thousands of solutions performed in parallel across many generations.

The quadratic assignment problem (QAP) is considered in this study. This problem is hard to solve, even for instances of moderate size (less than 100 elements). This contrasts, for instance, with the travelling salesman (TSP) problem for which fairly large instances can be solved optimally. For the TSP, the set of edges composed by the union of a few locally optimal solutions of moderate quality may contain a very large proportion of the edges of the best solution known [8, 9]. A goal of this paper is to evaluate if learning with locally optimal solutions is as successful for the QAP as it is for the TSP.

The objective values of solutions obtained by machine learning techniques for hard optimization problems are generally far from the values that can be obtained by direct heuristic algorithms. For the QAP, the reader is referred to [10] for a comparison of different methods based on neural graph machine network.

The remaining of this paper is organized as follow. Section 2 describes some technical background to understand the traditional local search algorithm, the quadratic assignment problem used for the experiments and frequent itemsets in associative rule learning. Section 3 introduces the extraction of frequent itemsets and its parallelization for local search algorithms. The experimental results are reported in Section 4. Finally, 5 concludes and proposes future research avenues.

2. Technical Background

2.1. Principles of Local Search Metaheuristics

Metaheuristics are a set of techniques for designing algorithms for producing hopefully high-quality solutions to hard optimization problems in a reasonable computational effort. Most of them are based on the iterative improvement of either a single solution (e.g. local search, simulated annealing or tabu search) or a population of solutions (e.g. genetic algorithms) of a given optimization problem.

Local search algorithms could be viewed as "walks through neighborhoods" meaning search trajectories through the solutions domains of the problems at hand. The walks are performed by moving from one solution to a (slightly) different one (see Algorithm 1).

Algorithm 1 Local search pseudo-code

```
Generate(s_0);
t := 0;
repeat
    m(t) := SelectMove(s(t));
    s(t+1) := ApplyMove(m(t), s(t));
    t := t + 1;
until Termination_criterion(s(t))
```

A local search starts with any solution, for instance a randomly generated one. At each iteration of the algorithm, the current solution is replaced by another one selected from the set of its neighboring candidates, and so on. An evaluation function associates a fitness value to each solution indicating its suitability to the problem. Many strategies related to the local search can be applied in the selection of a move: best improvement, first improvement, random selection, etc.

The computational complexity of a method based on metaheuristics is typically a polynomial function of n, the size of the instance data. Generally, the degree of the polynomial is moderate, but very seldom lower than $O(n^2)$.

A survey of the history and the state-of-the-art of metaheuristics can be found in [11].

2.2. The Quadratic Assignment Problem

To put in practice the different learning mechanisms proposed in this paper, the popular quadratic assignment problem (QAP) [12] has been investigated.

The QAP [13] arises in many applications such as facility location or data analysis. Let $A = (a_{ij})$ and $B = (b_{ij})$ be $n \times n$ matrices of positive integers. In the context of local search, the most convenient solution representation is by a permutation: The objective of the QAP is to find a permutation $\pi = (1, 2, \ldots, n)$

that minimizes the function:

$$z(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)}$$

The evaluation function has a $O(n^2)$ time complexity where n is the instance size. A neighborhood based on a pair-wise exchange ($\frac{n\times(n-1)}{2}$ neighbors) has been considered. Hence, for each iteration of a local search, $\frac{(n-2)\times(n-3)}{2}$ neighbors can be evaluated in $O(1)$ and $2n-3$ can be evaluated in $O(n)$ (Δ evaluations). The requirement is a structure which stores previous Δ evaluations in a quadratic space complexity. Evaluating all the Δ for the first time takes an effort in $O(n^3)$ but an effort only in $O(n^2)$ for each of the next local search iteration [14].

A complete review of the most successful algorithms to solve the QAP is proposed in [15].

2.3. Frequent Itemsets in Associative Rule Learning

In associate rule learning, the existence of very large databases requires to determine groups of items that frequently appear together in transactions, called *itemsets* [16]. From any itemset, one can determine an association rule that predicts how frequently an itemset is likely to occur in a transaction.

For example, a retail organization provides thousands of products and services [6]. The number of possible combinations of these products and services is potentially huge. The enumeration of all possible combinations is impractical, and methods are needed to concentrate efforts on those itemsets that are recognized as important to an organization. The most used measure of an itemset is its *support*, which is calculated as the percentage of all transactions that contain the itemset. Itemsets that meet a minimum support threshold are referred to as frequent itemsets.

An itemset which contains k items is a k-itemset. So, it can be said that an itemset is frequent if the corresponding support count is greater than a minimum support count.

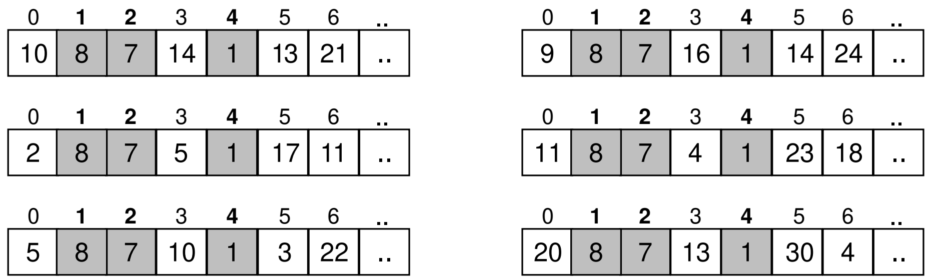

Figure 1: Extraction of one frequent itemset of size 3. In all solutions, elements 8, 7 and 1 appear at positions 1, 2 and 4.

3. Frequent Itemsets for Local Search Algorithms

The motivation of this research work is to investigate if one can learn anything from the solutions found in local search algorithms. One observation is that some elements from local optima may be found at the exact same positions of the global optimum, meaning that elements that frequently appear at particular positions may also be discovered in good solutions.

One tool to achieve this is to extract all the frequent itemsets from a set of solutions. In the context of combinatorial optimization, each itemset can be represented by pairs of one element associated with one position. Figure 1 illustrates an extraction for a 3-itemset.

Once all frequent itemsets are known, a new generation of solutions can be constructed from these itemsets.

3.1. Extraction and Combination of Frequent Itemsets

The global process used in this paper can be divided into two phases: the extraction of frequent itemsets and their combination to generate new solutions. Algorithm 2 gives an insight of how this global process works.

The initial set of solutions is obtained from the execution of multi-start local search algorithms (lines 1 to 4). For each local search, the initial solution is randomly generated and the selection of a better neighbor is done according to the best improvement strategy (steepest descent).

Algorithm 2 Extraction and combination of frequent itemsets

Require: $instance\_data$, $nb\_solutions$, $nb\_generations$, min_sup and $itemsets\_limit$

1: **for** $i \leftarrow 1, \ldots, nb\_solutions$ **do**
2:     $solutions[i] \leftarrow random\_initialization()$
3:     $solutions[i] \leftarrow local\_search(instance\_data, solutions[i])$
4: **end for**
5: **for** $generation \leftarrow 1, \ldots, nb\_generations$ **do**
6:     $all\_itemsets \leftarrow extract\_itemsets(solutions, min\_sup, itemsets\_limit)$
7:     **for** $i \leftarrow 1, \ldots, nb\_solutions$ **do**
8:         $solutions[i] \leftarrow combine\_itemsets(all\_itemsets)$
9:         $solutions[i] \leftarrow local\_search(instance\_data, solutions[i])$
10:     **end for**
11: **end for**

Ensure: $solutions$

In the main loop, the first phase consists in extracting all frequent itemsets from the current set of solutions (line 6) with Apriori algorithm [16]. Since the worst-case time complexity of Apriori algorithm is exponential according to the number of items, min_sup and $itemsets\_limit$ are user-defined parameters to control the number of candidate itemsets to retain in practice. The second phase is a procedure that combines these itemsets to construct new solutions that can be improved afterwards by the same local search algorithm (lines 7 to 10). The process is repeated for a given number of generations.

3.2. Apriori Algorithm for Extracting Itemsets from a Set of Solutions

The Apriori algorithm is used in this paper to extract all frequent itemsets from a set of solutions. It was originally designed to operate on databases containing transactions [16]. Basically, Apriori performs a bottom-up approach where frequent subsets are extended one item at a time (groups of candidates) and tested with the data. The algorithm finishes when no further successful extensions can be discovered.

Even if it is not the fastest method to directly extract the kth-itemset in comparison with other approaches [17, 18], its application seems the most appropriate since all frequent itemsets of any size are required here. More important, Apriori does not make any assumption of the size of the dataset and it perfectly fits in the context of big data algorithms.

Algorithm 3 Apriori algorithm for the extraction of frequent itemsets

Require: *solutions*, *min_sup* and *itemsets_limit*

1: $k := 1$
2: $C_k = generate\_itemsets(solutions, \emptyset)$
3: $L_k = filter\_itemsets(C_k, min\_sup, \emptyset)$
4: $all\_itemsets = L_k$
5: **while** $L_k \neq \emptyset$ **do**
6: $\quad C_{k+1} = generate\_itemsets(solutions, L\_k)$
7: $\quad L_{k+1} = filter\_itemsets(C_{k+1}, min\_sup, itemsets\_limit)$
8: $\quad all\_itemsets = all\_itemsets \cup L_{k+1}$
9: $\quad k := k + 1$
10: **end while**

Ensure: *all_itemsets*

Algorithm 3 describes the major steps of Apriori used in the *extract_itemsets* procedure of Algorithm 2. The first step consists in generating the list of all candidate itemsets of size 1 (lines 1 and 2). In the case of combinatorial optimization, an 1-itemset is exactly a pair of one element associated with one position. The candidate list is then pruned according to the *minimum support* (i.e. minimum number of times that an itemset must appear in all solutions) defined by the user (line 3). From the resulting filtered list of 1-itemsets, all candidate itemsets of size 2 are investigated (line 6) where a 2-itemset represents two pairs of one element associated with one position. The process is repeated with the filtered list of 2-itemsets to produce all 3-itemsets and so on until a candidate list cannot be built.

At each generation, all extracted k-itemsets are conserved in a list (lines 4 and 8) that will be later used to construct new solutions in the *combine_itemsets* procedure of Algorithm 2.

Limiting the number of retained itemsets (e.g. keeping one million itemsets that are among the most frequent ones) is necessary to reduce the computational and space complexities when generating new candidates for further generations.

3.3. Combining Itemsets for Creating a New Set of Solutions

The goal of the combine phase is to create a new set of solutions from all the frequent itemsets extracted during the previous generation.

Each solution is constructed by exploring all frequent itemsets. In this paper, two main strategies are taken into account regarding how itemsets are explored:

1. Random exploration of all frequent itemsets (REFI). In this strategy, every retained itemset has the same probability to be applied during the construction of a new solution.
2. Exploration based on sorted frequent itemsets (ESFI). All itemsets are sorted according to their support in decreasing order. The probability of applying an itemset to a solution (i.e. fixing elements at different positions) depends on the itemset support. For instance, a 2-itemset (e.g. element 5 at position 10 and element 1 at position 7) that appears in 2% of all previous solutions has also a probability of 2% to be in a new solution.

If the current solution cannot be completely constructed from the exploration of all itemsets, all unassigned elements will be randomly added at unassigned positions.

3.4. Parallelization Techniques for Frequent Itemsets

3.4.1. Parallel Execution of Local Search Algorithms

In a multi-start algorithm, the execution of each local search being independent, all algorithms can be parallelized according to a pool of executions (i.e. tasks waiting to be launched). The same stands for the combination of itemsets during the construction of each new solution.

Regarding the parallel thread execution, a dynamic scheduling is carried out since each local search may take a different amount of time.

A buffer is used to store a certain number of solutions that are being executed in parallel (e.g. 1000 solutions). When solutions are completed, they are written to a file and the buffer can be reused for the next solutions. Such a process allows limiting the space complexity in case a lot of solutions are created (e.g. 1,000,000 solutions). The process is repeated until all local search methods have been dealt with.

3.4.2. Parallel Extraction of Frequent Itemsets

Let n be the number of itemsets of size k. A new itemset being a combination of two previous itemsets, the number of candidate itemsets of size $k+1$ to examine is $m = n \times (n-1)/2$.

Since this extraction is independent for each itemset, all m itemsets in Apriori can be performed in parallel.

On the one hand, an itemset is composed of two indexes of previous itemsets. On the other hand, parallelization units such as threads are determined by a unique *id*. Therefore, one mapping has to be considered to transform one index into two ones.

Given *id* the index of a new itemset to generate, the index of the first previous itemset i is equal to $n - 2 - \lfloor \frac{\sqrt{8 \times (m - id - 1) + 1} - 1}{2} \rfloor$ and the index of the second previous itemset j is equal to $id - i \times (n-1) + \frac{i \times (i+1)}{2} + 1$.

This calculated mapping avoids an unnecessary use of mapping tables (containing all indexes) that can rapidly become prohibitive in terms of memory.

In a similar manner, a buffer and a file are also required to reduce the space complexity.

4. Performance Evaluation

The computational results presented in this section were obtained on a PC running on Linux and equipped with an AMD Ryzen Threadripper 1950X 3.4Ghz (16 cores / 32 threads). The algorithms introduced in Section 2 were implemented in C++ using the OpenMP Library for the parallelization. The candidate itemsets for one generation representing up to a dozen of gigabytes of data, they are written in a file and a buffer storing only 10,000 candidate itemsets is reused accordingly.

This parallelization approach results in almost ideal speed-ups (from 12× to 15× according to the number of candidate itemsets). An efficient parallelization of a local search on GPU is not evident and previous works have reported relatively modest speed-ups due to memory access latency [19].

4.1. QAP instances

The QAPLIB repository [20] contains 136 instances and has been enriched by hundreds other ones freely available on `http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap` Since it was not practically possible to conduct our numerical experiments for all instances, only 11 QAP instances were carefully selected. All chosen instances are widely studied in the literature [21] and are considered as the hardest to solve for the QAP.

The selected instances cover a large panel of the flows/distances matrices structures that can be found in the literature. Their size (n between 45 and 64) is large enough so that a solver based on exact methods cannot solve the problem on modern computers.

The first 3 instances are from Skorin-Kapov [22] (sko49, sko56 and sko64). No optimal solution has been proven yet for these instances. The distances are Manhattan on a rectangular grid, and the flows are pseudo-random numbers. These instances are similar to Nugent et al. ones, but larger. Due to symmetries in the distance matrix, multiples of 4 or 8 optimal solutions exists.

Then, 3 asymmetrical instances from Li and Pardalos (lipa50a, lipa60a and lipa50b) were selected. These instances were generated so that the optimal solutions are known [23].

Then, 2 symmetrical instances with flows and distances randomly, uniformly generated have been selected (tai50a and tai60a) [14]. These instances are similar to Roucairol's ones, but larger.

Then, 2 asymmetrical instances non-uniformly generated (tai50b and tai60b) comes from[24]. An instance for generating grey patterns (tai64c) proposed in the same article has also been selected. This instance is not specially hard, but has a very large number of optimal solutions, spread all over the solutions' space.

Finally, a symmetrical and structured instance (tai45e01) proposed in [25] was selected. This instance was generated in such a way that a number of local search based methods have difficulties to find a moderately good solution.

4.2. Parameters for the Experiments

The algorithms of this paper rely on extracting most frequent itemsets from all solutions then combining them to create a new set of solutions.

In Algorithm 2, the number of generations is set to 8 and 10,000 local searches are executed per generation. The default minimum support for the extraction of itemsets is set to 0.1% (i.e. keep itemsets which appear in 10 out of 10,000 solutions). The itemsets limit is set to one million for each k-itemset. Basically, these parameters determined the duration of the training phase and the memory space that is used. All these parameters were selected and tuned in such a way that each generation does not exceed one day of calculation.

Regarding the combining phase, the first set of experiments are based on the random exploration of frequent itemsets (REFI) whereas the second one is on the exploration on sorted frequent itemsets (ESFI). A multi-start with $90,000$ local search algorithms from random solutions is also considered. Even if the execution time differs, it is used as an indicator of comparison where no learning process is implemented. Disregarding the time needed for selecting the itemsets and building starting solutions, all the methods are indeed performing $90,000$ local searches.

4.3. Quality of Solutions

For optimization problems, the main criteria to be evaluated is the quality of solutions. The last is measured relatively to the value of the best solution

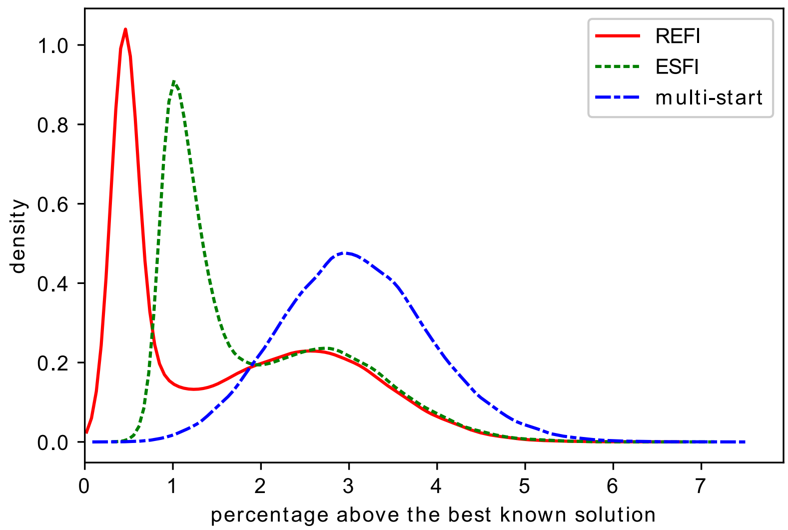

Figure 2: REFI, ESFI and multi-start distribution of solutions quality for sko49 (Manhattan distances on a square grid) using a kernel density estimation plot.

known to date (bvk), which is believed to be optimal. The distribution of the quality of the solutions is visualized with a kernel density estimation plot.

The quality of the solutions for the instances are graphically illustrated in Figures 2, to 13. All the solutions compared to the bvk are represented for the 90, 000 solutions found by the multi-start algorithm (dash-dotted line) and the 8 generations of REFI (plain line) and ESFI (dotted line) learning methods.

Corresponding numerical results including the minimum, the 5th percentile, the median, the mean and the maximum are reported in Tables A.3 to A.14 in Appendix A.

For the instance sko49 (Figure 2), the density reveals that most solutions produced by REFI and ESFI algorithms are, respectively, about 0.5% and 1% above the bvk whereas most of those produced by a random multi-start are around 3%. A similar observation can be made for the instance sko56 (Figure 3). The phenomenon is more pronounced for the instance sko64 (Figure 4) where the REFI algorithm was able to produce solutions very close to the bvk.

The benefits of the learning phase are also prominent for the lipa50a instance

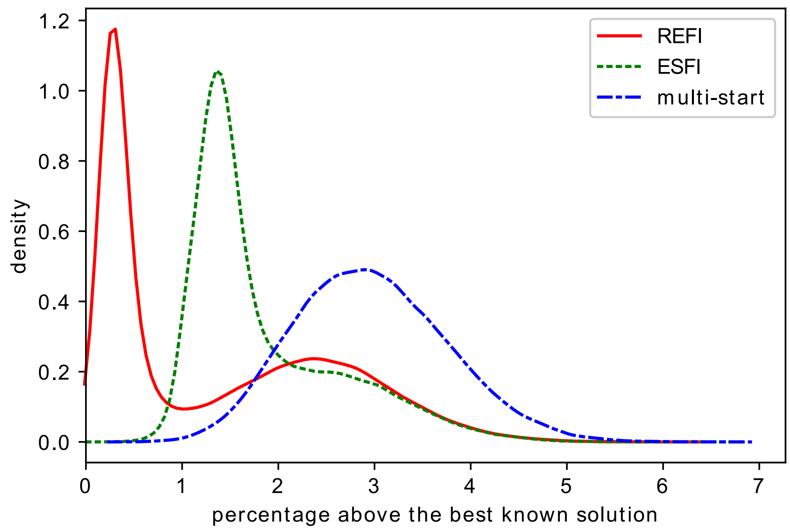

Figure 3: REFI, ESFI and multi-start distribution of solutions quality for sko56 (Manhattan distances on a rectangular grid) using a kernel density estimation plot.

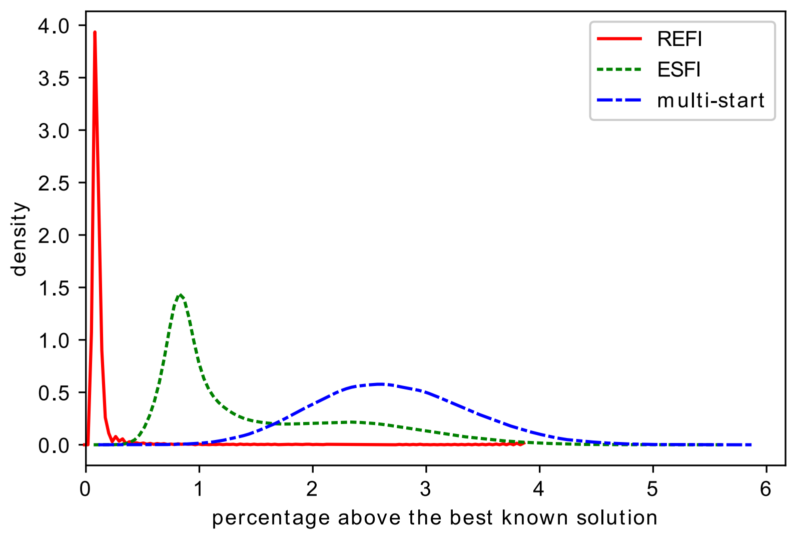

Figure 4: REFI, ESFI and multi-start distribution of solutions quality for sko64 (Manhattan distances on a square grid) using a kernel density estimation plot.

(Figure 5), where the multi-start from random solutions was unable to find the optimum. A similar behavior stands for the lipa60a instance in Figure 6 except

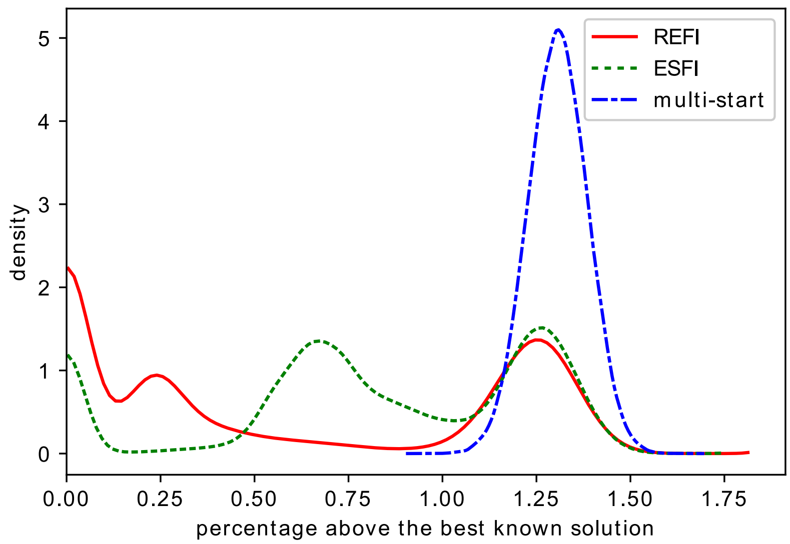

Figure 5: REFI, ESFI and multi-start distribution of solutions quality for lipa50a (asymmetric with known optimal solutions) using a kernel density estimation plot.

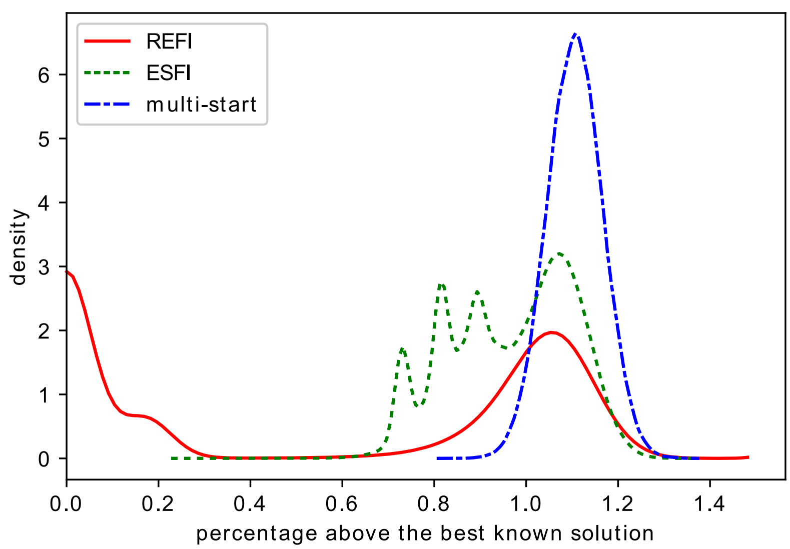

Figure 6: REFI, ESFI and multi-start distribution of solutions quality for lipa60a (asymmetric with known optimal solutions) using a kernel density estimation plot.

that REFI is the only algorithm able to reach the known optimum.

Regarding the lipa50b instance (Figure 7), the difference of quality is very

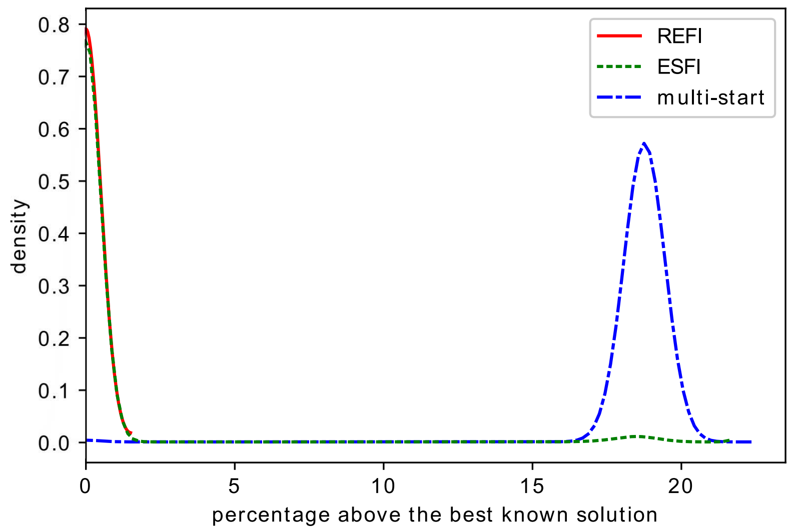

Figure 7: REFI, ESFI and multi-start distribution of solutions quality for lipa50b (asymmetric with known optimal solutions) using a kernel density estimation plot.

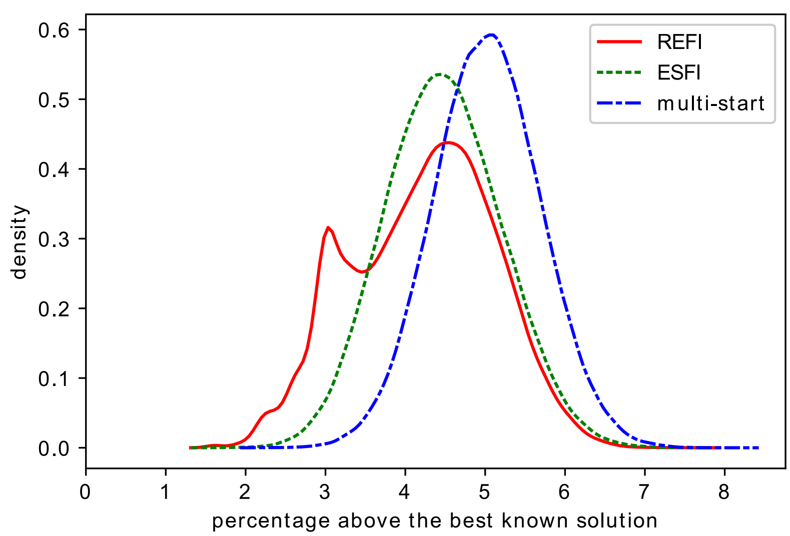

Figure 8: REFI, ESFI and multi-start distribution of solutions quality for tai50a (uniformly generated) using a kernel density estimation plot.

important since most REFI and ESFI solutions are optimal whereas multi-start solutions are between 15 and 20% above the bvk. For the 11 selected instances,

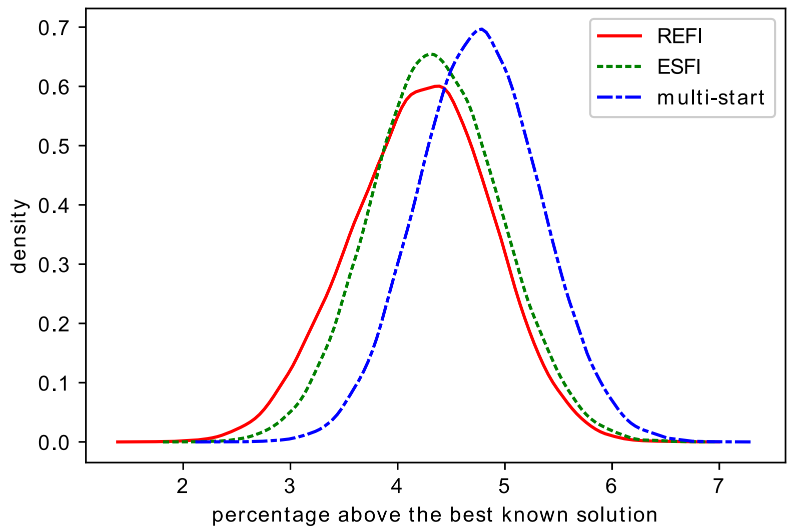

Figure 9: REFI, ESFI and multi-start distribution of solutions quality for tai60a (uniformly generated) using a kernel density estimation plot.

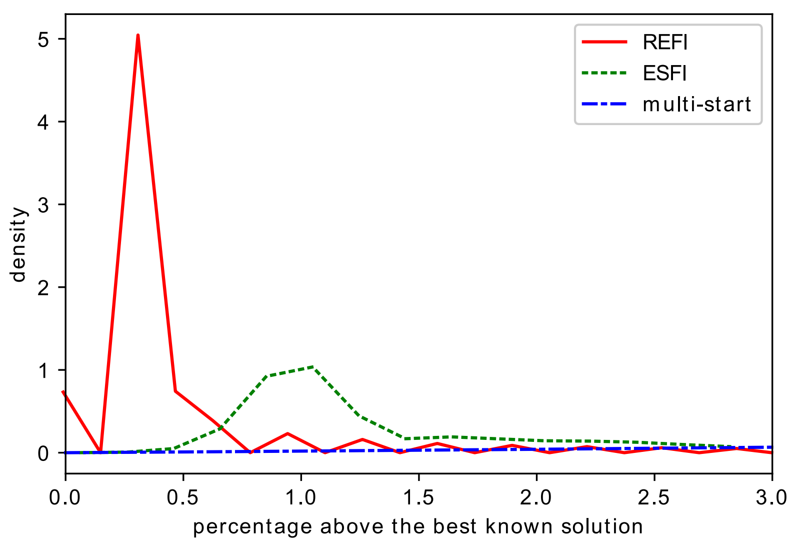

Figure 10: REFI, ESFI and multi-start distribution of solutions quality for tai50b (asymmetric and randomly generated) using a kernel density estimation plot.

this is the only one for which learning with itemsets is highly successful.

For the tai50a instance (Figure 8), there is a moderate trend indicating that

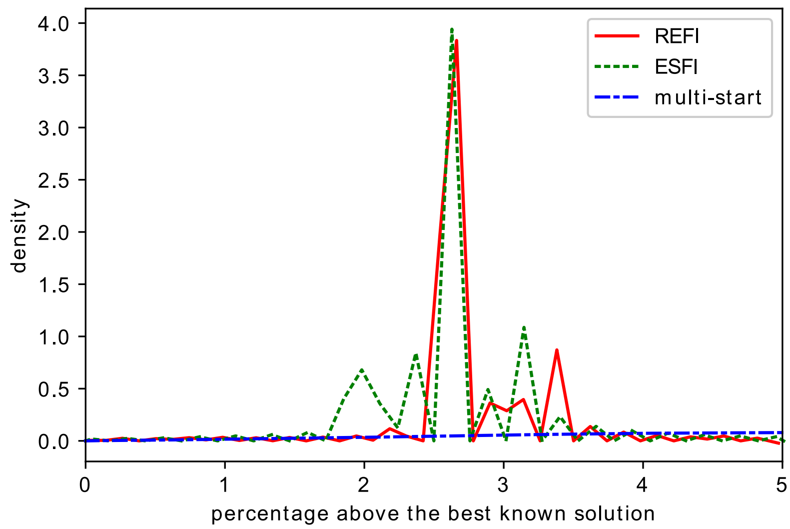

Figure 11: REFI, ESFI and multi-start distribution of solutions quality for tai60b (asymmetric and randomly generated) using a kernel density estimation plot.

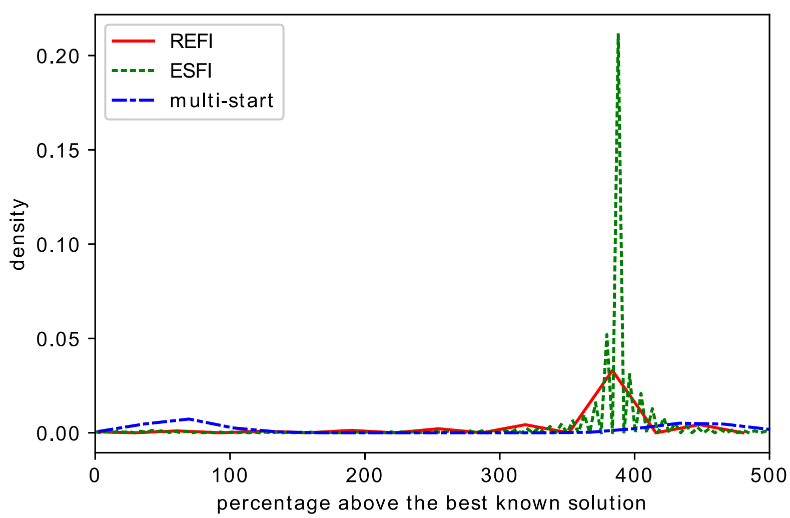

Figure 12: REFI, ESFI and multi-start distribution of solutions quality for tai45e01 (structured) using a kernel density estimation plot.

most of the REFI and ESFI runs are able to learn something. Unsurprisingly, the learning is less pronounced for randomly, uniformly generated instances. A similar observation can be made for the tai60a instance (Figure 9).

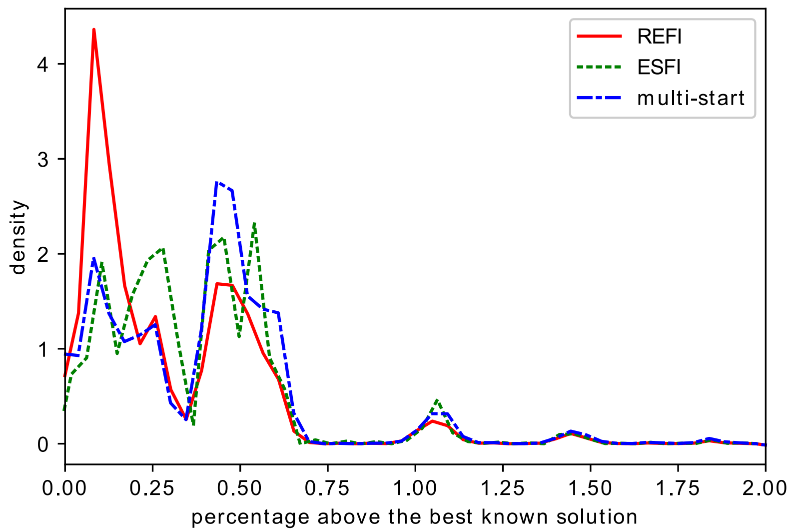

Figure 13: REFI, ESFI and multi-start distribution of solutions quality for tai64c (structured) using a kernel density estimation plot.

Regarding the instance tai50b ((Figure 10), there are peaks showing that most ESFI and REFI solutions are between 0.5 and 1% above the bvk. The multi-start algorithm produces solutions that are spread 7.3% above the bvk with a standard deviation of 3.3%.

A similar observation can be made for the instance tai60b in Figure 11, where different high-density peaks indicate that most ESFI and REFI solutions are between 2 and 4% above the bvk. The multi-start algorithm solutions that are spread 8% above the bvk with a standard deviation of 3.4%. For this type of instances, learning with itemsets is possible, but not as successful as it is for lipab instances.

Regarding the structured instance tai45e01 (Figure 12), the ESFI and REFI algorithms are completely unable to learn something interesting. These algorithms are just focusing on solutions that are very far from the optimal one. The learning techniques based on the frequent itemsets seem to be inefficient for dealing with such structured instances. The population of solutions just converges too early. We were rather surprised by this result, since various metaheuristics combining a local search with a learning mechanism are perfectly able to reach the best solution, for instance GRASP with path relinking, late acceptance local search, genetic hybrids or ant systems [25].

For tai64c (Figure 13), most solutions being below 1% above the bvk, it is not clear that a learning algorithm outperforms a simple multi-start. It might be explained by the fact that the instance has multiple global optima and it is easy to solve it optimally [26].

4.4. Additional Information for the Positions of Solutions

Another criterion to assess is the similarity of the solutions produced by the algorithms with a target solution with bvk. The similarity can be measured by the number of elements in that are at the same position. These results are reported in Table 1 and Table 2. The 3. column provides the mean and the standard deviation of the number of positions identical to the target solution. The next two columns are the percentage of solutions under 5% above the bvk including those ones that share at least 10% of common positions with the target. The next column provides the percentage of different solutions. This proportion gives an indication of the population diversity. Finally, the number of itemsets revealing all the patterns discovered during the exploration phase is reported.

Table 1 shows for the sko49 instance that the number of positions identical to the target is almost non-existent for all algorithms (between 1 and 2 on the average). It is an easy instance since more than 98% of solutions are under 5% above the bvk, including the simple multi-start from random solutions. As shown in Figure 2, the learning mechanism helps improving the last percentages above the bvk.

A similar observation can be made for sko56. The main difference is that among all the solutions that are under 5% above the bvk, there is a significant percentage of solutions (61.88% for REFI and 24.61% for ESFI) that share more than 10% of common positions with the target. The same remark occurs for the instance sko64 but the diversity of REFI solutions is pretty low (3.04%).

Table 1: Additional results for the positions of produced solutions for sko49, sko56, sko64, lipa50a, lipa60a and lipa50b instances : number of positions that are identical (mean and standard deviation) to a target solution, percentage of solutions under 5% above the best value known (bvk) and, for those that share at least 10% of common positions with the target, percentage of different solutions and total number of itemsets discovered.

instance	algorithm	# positions identical to the target	% solutions under 5% above the bvk		% different solutions	# itemsets
			all	pos > 10%		
sko49	REFI	$1.3_{1.5}$	99.80%	3.85%	51.35%	66.8×10^6
	multi-start	$1.8_{1.7}$	98.32%	7.20%	100.00%	-
	ESFI	$1.1_{1.4}$	99.74%	2.69%	69.78%	58.9×10^6
sko56	REFI	$7.6_{4.9}$	99.92%	61.88%	46.13%	83.1×10^6
	multi-start	$2.0_{2.0}$	99.27%	6.01%	100.00%	-
	ESFI	$4.1_{2.2}$	99.93%	24.61%	83.80%	58.9×10^6
sko64	REFI	$7.4_{7.2}$	100.00%	49.74%	3.04%	136.4×10^6
	multi-start	$2.2_{2.0}$	99.94%	3.55%	100.00% -	
	ESFI	$4.6_{2.3}$	99.99%	17.16%	67.41%	84.3×10^6
lipa50a	REFI	$30.3_{21.1}$	100.00%	71.57%	45.46%	79.9×10^6
	multi-start	$1.3_{1.5}$	100.00%	1.87%	100.00%	-
	ESFI	$22.4_{16.9}$	100.00%	70.06%	59.93%	101.4×10^6
lipa60a	REFI	$32.7_{27.0}$	100.00%	66.31%	50.06%	176.7×10^6
	multi-start	$1.2_{1.4}$	100.00%	0.61%	100.00%	-
	ESFI	$10.1_{8.2}$	100.00%	53.88%	77.09%	63.8×10^6
lipa50b	REFI	$50.0_{0.0}$	100.00%	100.00%	0.00%	37.6×10^6
	multi-start	$1.9_{3.7}$	0.41%	0.41%	99.59%	-
	ESFI	$49.1_{6.2}$	98.03%	98.03%	1.97%	23.0×10^6

Table 2: Additional results for the positions of produced solutions for tai50a, tai60a, tai50b, tai60b, tai45e01 and tai64c instances: number of positions that are identical (mean and standard deviation) to a target solution, percentage of solutions under 5% above the best value known (bvk) and, for those that share at least 10% of common positions with the target, the percentage of different solutions and the number of itemsets discovered.

instance	algorithm	# positions identical to the target	% solutions under 5% above the bvk		% different solutions	# itemsets
			all	pos > 10%		
tai50a	REFI	$1.9_{1.4}$	79.93%	0.71%	88.39%	20.5×10^6
	multi-start	$1.1_{1.2}$	48.76%	0.25%	100.00%	-
	ESFI	$1.6_{1.3}$	76.29%	0.64%	100.00%	40.2×10^6
tai60a	REFI	$0.9_{1.0}$	88.77%	0.01%	99.99%	33.0×10^6
	multi-start	$1.0_{1.0}$	66.41%	0.01%	100.00%	-
	ESFI	$0.8_{0.9}$	85.50%	0.01%	100.00%	24.2×10^6
tai50b	REFI	$22.9_{11.4}$	87.58%	78.93%	12.86%	106.0×10^6
	multi-start	$2.1_{2.6}$	26.70%	4.28%	100.00%	-
	ESFI	$1.2_{1.7}$	86.11%	1.56%	38.59%	108.2×10^6
tai60b	REFI	$1.8_{3.5}$	97.67%	3.70%	24.40%	123.4×10^6
	multi-start	$3.1_{3.1}$	21.91%	6.89%	100.00%	-
	ESFI	$2.5_{2.0}$	94.27%	1.88%	25.53%	226.9×10^6
tai45e01	REFI	$0.9_{3.6}$	0.02%	0.02%	10.68%	33.2×10^6
	multi-start	$3.3_{5.0}$	0.01%	0.01%	96.76%	-
	ESFI	$0.5_{3.1}$	0.03%	0.03%	9.48%	109.4×10^6
tai64c*	REFI	$30.0_{15.1}$	99.98%	55.01%	100.00%	71.6×10^6
	multi-start	$10.4_{13.7}$	99.98%	6.14%	100.00%	-
	ESFI	$22.8_{17.3}$	99.98%	33.73%	94.41%	45.1×10^6

Regarding lipa50a and lipa60a instances (asymmetric with known optimal solutions), the number of shared positions of REFI and ESFI with the target is prominent (around 10 and 30). But it is not a difficult instance since 100% of solutions are under 5% above the bvk. As shown in Figure 5 and Figure 6, the learning phase is also determinant for improving the last percentages above the bvk.

The lipa50b case (high values for matrix entries) is interesting since only 0.41% of multi-start solutions are under 5% above the bvk. The benefits of learning mechanisms are meaningful for this instance since most REFI and ESFI solutions converge to the target.

For tai50a and tai60a instances, Table 2 shows that the number of positions identical to the target is also almost non-existent. Indeed, the produced solutions that share 10% of common positions with the target and that are under 5% above the bvk is less than 1%. The number of itemsets (patterns) discovered for both tai50a and tai60a is lower than the other instances.

Things are quite different for the tai50b (asymmetric and randomly generated) where the percentage of different solutions is rather low (less than 40%), meaning that many solutions converge to the same local optima. On the one hand, the solutions produced by REFI share an important number of common positions with the target (22.9 in average). On the other hand, ESFI has very little in common with the target. In both cases, the number of discovered itemsets is rather high (more than 100 millions) and 85% solutions are under 5% above the bvk. It is really significant in comparison with a multi-start where only 26.7% solutions are within the same quality.

A similar observation can be made for the tai60b instance. Even if the number of positions shared with the target is pretty low (less than 2.5), more than 94% of produced solutions by a learning algorithm are under 5% above the bvk, whereas a simple multi-start has only 21.91% under this level. Interestingly this instance has generated the highest number of different patterns.

Regarding the instance tai45e01, the diversity of the population of solutions is also significantly low. It represents less than 11% of different solutions even if the number of itemsets is significant. The number of positions identical to the target is even lower than a multi-start with $90,000$ random solutions. The number of solutions under 5% above the bvk is close to 0%. It seems that the learning mechanisms studied in this article are not really efficient for such a structured instance. The itemsets produced are just focusing on bad quality local optima, very far from the global optimum.

The structured tai64c is easy to solve since $12,715$ different global optima were found during the different runs. Since global optima are spread all over the solutions' space, it is not clear whether something can be learned with itemsets or not. Anyway, since 99.98% solutions are under 5% above the bvk for all algorithms, the benefits of a learning process are not really meaningful in the context of optimization.

5. Conclusions

The main interest in combining the unsupervised association rule learning with metaheuristics is to discover useful knowledge about the history of the search in order to enhance the produced solutions.

In this paper, we proposed to incorporate the extraction of frequent itemsets for parallel local search algorithms in a big data context. The global process can be iterated through two phases: the extraction of millions frequent itemsets and their combination for generating new solutions.

For the QAP, learning mechanisms through association rule learning have shown significant improvements in comparison with a multi-start from random solutions for a number of problem instances of the literature. From this point of view, the REFI and ESFI developed in this paper have been revealed to be competitive but for one problem instance. It has to be mentioned that a uniform selection of itemsets reveals superior to a selection biased with the frequency of appearance.

The drawback of this learning is that they take a full day on a single machine to train one generation of solutions. In comparison, the dedicated robust taboo search [14] or the fast ant systems [27] will find better solutions in just a few minutes.

However, in the context of big data, one day of calculation on a single machine is still reasonable regarding usual machine and deep learning trainings that may take a couple of weeks on a cluster of GPU-based machines [28].

In contrast with metaheuristics dedicated to a specific optimization problem, the advantage of these learning techniques is that they are rather simple to design and do not require a priori knowledge of the problem at hand. However, for the QAP, the quality of the solution produced with these learning techniques is not competitive compared to state-of-the-art metaheuristics.

A research avenue could be a finer tuning of parameters (i.e. minimum support, itemsets limit and number of solutions) to see how they can influence the search process and to control the duration of the execution according to the scenario. For example, a low minimum support allows limiting the training phase to couple minutes, while a higher number of solutions will make it last a week. Another perspective could be to investigate how machine learning can enhance state-of-the-art metaheuristics for the QAP.

The general conclusion of this paper is that there is still a long way till general learning techniques will surpass more direct optimization techniques for the QAP. This contrasts with works on other optimization problems like the travelling salesman. Indeed, for this problem, a few dozen of a very fast randomized local search is able to extract most of the components of target solutions. Since learning techniques can be very efficient for this optimization problem, it would be interesting to study its behavior for in other problems where a permutation is search for, such as the flowshop scheduling problem.

[1] Z. H. Zhou, N. V. Chawla, Y. Jin, G. J. Williams, Big data opportunities and challenges: Discussions from data analytics perspectives [discussion forum], IEEE Computational Intelligence Magazine 9 (4) (2014) 62–74. doi:10.1109/MCI.2014.2350953.

[2] A. L'heureux, K. Grolinger, H. F. Elyamany, M. A. Capretz, Machine learning with big data: Challenges and approaches, IEEE Access 5 (2017) 7776–7797.

[3] M. Birattari, J. Kacprzyk, Tuning metaheuristics: a machine learning perspective, Vol. 197, Springer, 2009.

[4] L. Calvet, J. de Armas, D. Masip, A. A. Juan, Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs, Open Mathematics 15 (1) (2017) 261–280.

[5] E. Alpaydin, Introduction to machine learning, MIT press, 2020.

[6] R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, in: Proceedings of the 1993 ACM SIGMOD international conference on Management of data, 1993, pp. 207–216.

[7] M. Ribeiro, A. Plastino, S. Martins, Hybridization of grasp metaheuristic with data mining techniques, J. Math. Model. Algorithms 5 (2006) 23–41. doi:10.1007/s10852-005-9030-1.

[8] É. D. Taillard, An n log n heuristic for the travelling salesman problem, in: Proceedings of the 13. Metaheuristics International Conference, 2019, pp. 121–124.

[9] É. D. Taillard, K. Heslgaun, POPMUSIC for the travelling salesman problem, EURO Journal of Operational Research 272 (2) (2019) 420–429. doi:10.1016/j.ejor.2018.06.039.
URL http://mistic.heig-vd.ch/taillard/articles.dir/TaillardHelsgaun2018.pdf

[10] R. Wang, J. Yan, X. Yang, Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching (2020). arXiv:1911.11308.

[11] E.-G. Talbi, Metaheuristics: From design to implementation, Wiley, 2009.

[12] T. C. Koopmans, M. Beckmann, Assignment Problems and the Location of Economic Activities, Econometrica 25 (1) (1957) 53–76.
URL http://www.jstor.org/stable/1907742

[13] R. E. Burkard, E. Çela, G. Rote, G. J. Woeginger, The quadratic assignment problem with a monotone anti-monge and a symmetric toeplitz matrix: Easy and hard cases, Math. Program. 82 (1998) 125–158.

[14] É. D. Taillard, Robust taboo search for the quadratic assignment problem, Parallel Computing 17 (4-5) (1991) 443–455.

[15] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, T. Querido, A survey for the quadratic assignment problem, European journal of operational research 176 (2) (2007) 657–690.

[16] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: International Conference on Very Large Databases (VLDB), 1994, pp. 487–499.

[17] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD '00, Association for Computing Machinery, New York, NY, USA, 2000, pp. 1–12. doi:10.1145/342009.335372.
URL https://doi.org/10.1145/342009.335372

[18] J. S. Park, M.-S. Chen, P. S. Yu, An effective hash-based algorithm for mining association rules, Acm sigmod record 24 (2) (1995) 175–186.

[19] T. V. Luong, N. Melab, É. D. Taillard, E.-G. Talbi, Parallelization strategies for hybrid metaheuristics using a single gpu and multi-core resources,

in: 12th International Conference on Parallel Problem Solving From Nature (PPSN) prooceedings, 2012, only preprint technical report available. URL http://mistic.heig-vd.ch/taillard/articles.dir/LuongMTT2012.pdf

[20] R. E. Burkard, S. E. Karisch, F. Rendl, Qaplib–a quadratic assignment problem library, Journal of Global optimization 10 (4) (1997) 391–403. URL https://coral.ise.lehigh.edu/data-sets/qaplib/

[21] E. Loiola, N. Abreu, P. Boaventura-Netto, P. Hahn, T. Querido, A survey of the quadratic assignment problem, European Journal of Operational Research 176 (2007) 657–690. doi:10.1016/j.ejor.2005.09.032.

[22] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem, ORSA Journal on computing 2 (1) (1990) 33–45.

[23] Y. Li, P. M. Pardalos, Generating quadratic assignment test problems with known optimal permutations, Computational Optimization and Applications 1 (2) (1992) 163–184.

[24] É. D. Taillard, Comparison of iterative searches for the quadratic assignment problem, Location Science 3 (2) (1995) 87 – 105. doi:10.1016/0966-8349(95)00008-6. URL http://www.sciencedirect.com/science/article/pii/0966834995000086

[25] Z. Drezner, P. M. Hahn, É. D. Taillard, Recent advances for the quadratic assignment problem with special emphasis on instances that are difficu Annals OR 139 (1) (2005) 65–94. URL http://mistic.heig-vd.ch/taillard/articles.dir/DreznerHT2005.pdf

[26] Z. Drezner, Finding a cluster of points and the grey pattern quadratic assignment problem, OR Spectr. 28 (3) (2006) 417–436. doi:10.1007/s00291-005-0010-7. URL https://doi.org/10.1007/s00291-005-0010-7

[27] É. D. Taillard, Fant: Fast ant system, Tech. rep., Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, iDSIA Technical Report IDSIA-46-98 (1998).

[28] D. Erhan, A. Courville, Y. Bengio, P. Vincent, Why does unsupervised pre-training help deep learning?, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 201–208.

Appendix A. Quality of Solutions: Minimum, 5th percentile, Median, Mean and Maximum

.

Table A.3: Quality of the solutions for sko49. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	23506	23796	24100	$24106.8_{197.4}$	24924
REFI gen 1	23478	23752	24046	$24051.3_{188.3}$	24872
REFI gen 2	23428	23740	24032	$24037.7_{188.4}$	24806
REFI gen 3	23446	23724	24006	$24014.3_{189.5}$	24872
REFI gen 4	23450	23639	23876	$23896.5_{178.4}$	24602
REFI gen 5	23422	23482	23582	$23608.9_{120.0}$	24326
REFI gen 6	23420	23452	23494	$23496.7_{25.9}$	23672
REFI gen 7	23440	23458	23484	$23492.7_{17.5}$	23602
REFI gen 8	23440	23458	23484	$23488.6_{18.3}$	23634
ESFI gen 1	23446	23758	24056	$24063.4_{192.3}$	24988
ESFI gen 2	23522	23755	24050	$24056.1_{190.0}$	24764
ESFI gen 3	23488	23720	24018	$24020.6_{189.0}$	24796
ESFI gen 4	23480	23676	23924	$23932.8_{173.9}$	24646
ESFI gen 5	23504	23608	23702	$23708.5_{70.7}$	24164
ESFI gen 6	23530	23576	23644	$23646.1_{45.9}$	23892
ESFI gen 7	23540	23598	23646	$23643.7_{33.6}$	23768
ESFI gen 8	23550	23588	23604	$23609.4_{18.8}$	23748
multi-start$_{90000}$	23474	23790	24098	$24105.1_{198.3}$	25086

Table A.4: Quality of the solutions for sko56. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	34582	35048	35460	$35475.7_{272.4}$	36518
REFI gen 1	34628	34984	35368	$35383.2_{260.3}$	36518
REFI gen 2	34570	34955	35340	$35354.2_{258.9}$	36402
REFI gen 3	34556	34900	35302	$35314.8_{264.3}$	36408
REFI gen 4	34534	34724	35064	$35099.2_{265.4}$	36434
REFI gen 5	34462	34512	34616	$34619.9_{73.2}$	35094
REFI gen 6	34462	34516	34550	$34560.6_{32.7}$	34748
REFI gen 7	34462	34528	34548	$34552.9_{17.5}$	34742
REFI gen 8	34462	34542	34548	$34549.7_{11.2}$	34708
ESFI gen 1	34620	34992	35392	$35402.3_{260.2}$	36646
ESFI gen 2	34614	34966	35366	$35378.8_{262.9}$	36420
ESFI gen 3	34634	34930	35286	$35299.7_{242.5}$	36732
ESFI gen 4	34572	34816	35000	$35004.5_{116.8}$	35544
ESFI gen 5	34544	34800	34916	$34915.6_{71.3}$	35210
ESFI gen 6	34566	34802	34904	$34904.0_{70.9}$	35180
ESFI gen 7	34566	34804	34934	$34928.6_{79.5}$	35286
ESFI gen 8	34580	34808	34930	$34930.0_{76.8}$	35242
multi-start$_{90000}$	34628	35056	35464	$35477.9_{272.2}$	36750

Table A.5: Quality of the solutions for sko64. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	48498	48498	48514	$48519.1_{18.5}$	48666
REFI gen 1	48498	48498	48518	$48578.2_{214.4}$	50364
REFI gen 2	48498	48508	48518	$48516.6_{17.5}$	49160
REFI gen 3	48498	48518	48518	$48514.7_{6.0}$	48592
REFI gen 4	48498	48504	48518	$48520.4_{17.1}$	48674
REFI gen 5	48498	48504	48518	$48523.1_{22.1}$	48708
REFI gen 6	48498	48506	48522	$48529.8_{23.0}$	48698
REFI gen 7	48498	48508	48508	$48516.7_{13.2}$	48656
REFI gen 8	48498	48506	48526	$48529.2_{21.7}$	48666
ESFI gen 1	48738	49220	49700	$49716.7_{319.6}$	51082
ESFI gen 2	48754	49178	49674	$49686.8_{323.7}$	51006
ESFI gen 3	48692	49058	49508	$49529.7_{308.9}$	50832
ESFI gen 4	48704	48880	49080	$49088.4_{137.1}$	49814
ESFI gen 5	48672	48792	48940	$48945.9_{98.7}$	49312
ESFI gen 6	48680	48799	48902	$48915.7_{71.0}$	49244
ESFI gen 7	48742	48786	48906	$48881.3_{59.6}$	49246
ESFI gen 8	48812	48852	48896	$48892.2_{17.2}$	49010
multi-start$_{90000}$	48664	49288	49788	$49803.3_{327.2}$	51240

Table A.6: Quality of the solutions for lipa50a. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	62693	62823	62904	$62903.5_{48.9}$	63081
REFI gen 1	62683	62808	62888	$62887.6_{48.3}$	63074
REFI gen 2	62579	62789	62874	$62872.7_{50.4}$	63104
REFI gen 3	62357	62718	62837	$62831.9_{65.4}$	63034
REFI gen 4	62093	62093	62329	$62319.3_{174.1}$	62839
REFI gen 5	62093	62093	62093	$62156.2_{96.1}$	62569
REFI gen 6	62093	62093	62093	$62149.3_{83.3}$	62457
REFI gen 7	62093	62093	62093	$62143.4_{75.0}$	62432
REFI gen 8	62093	62093	62093	$62151.8_{81.8}$	62430
ESFI gen 1	62656	62812	62891	$62891.0_{49.1}$	63074
ESFI gen 2	62624	62798	62880	$62879.9_{49.4}$	63086
ESFI gen 3	62396	62746	62849	$62845.6_{58.9}$	63028
ESFI gen 4	62093	62381	62611	$62591.2_{117.5}$	62877
ESFI gen 5	62093	62259	62536	$62518.2_{137.9}$	62856
ESFI gen 6	62093	62093	62440	$62458.4_{142.5}$	62730
ESFI gen 7	62093	62495	62495	$62512.8_{35.7}$	62552
ESFI gen 8	62093	62093	62093	$62086.5_{6.5}$	62093
multi-start$_{90000}$	62672	62824	62904	$62936.7_{58.5}$	63128

Table A.7: Quality of the solutions for lipa60a. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	108196	108298	108403	$108403.6_{64.3}$	108622
REFI gen 1	108110	108281	108387	$108387.2_{64.0}$	108592
REFI gen 2	108056	108259	108366	$108365.4_{64.8}$	108582
REFI gen 3	107941	108220	108336	$108334.7_{69.4}$	108626
REFI gen 4	107218	107978	108218	$108198.0_{128.6}$	108526
REFI gen 5	107218	107218	107218	$107215.1_{29.2}$	108036
REFI gen 6	107218	107218	107218	$107243.2_{69.0}$	107401
REFI gen 7	107218	107218	107218	$107259.3_{80.4}$	107401
REFI gen 8	107218	107218	107218	$107261.0_{81.1}$	107401
ESFI gen 1	108147	108285	108392	$108391.5_{64.6}$	108637
ESFI gen 2	108120	108275	108381	$108380.7_{64.2}$	108609
ESFI gen 3	108119	108261	108368	$108367.6_{65.1}$	108623
ESFI gen 4	108012	108232	108342	$108341.7_{66.0}$	108576
ESFI gen 5	107513	108049	108212	$108205.7_{87.0}$	108573
ESFI gen 6	107668	107974	108130	$108121.8_{84.4}$	108367
ESFI gen 7	107876	108001	108172	$108140.7_{75.1}$	108333
ESFI gen 8	107947	108000	108091	$108072.0_{54.5}$	108185
multi-start$_{90000}$	108106	108297	108404	$108471.8_{94.1}$	108669

Table A.8: Quality of the solutions for lipa50b. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	1210244	1427465	1437272	$1436333.5_{15298.1}$	1459683
REFI gen 1	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
REFI gen 2	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
REFI gen 3	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
REFI gen 4	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
REFI gen 5	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
REFI gen 6	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
REFI gen 7	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
REFI gen 8	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
ESFI gen 1	1210244	1210244	1210244	$1245422.1_{81194.2}$	1453442
ESFI gen 2	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
ESFI gen 3	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
ESFI gen 4	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
ESFI gen 5	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
ESFI gen 6	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
ESFI gen 7	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
ESFI gen 8	1210244	1210244	1210244	$1210345.2_{101.3}$	1210244
multi-start$_{90000}$	1210244	1427468	1437230	$1436295.4_{15700.8}$	1462070

Table A.9: Quality of the solutions for tai50a. 10,000 solutions per generation. The common starting point is a multi-start then 10,000 local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	5063184	5132739	5186884	$5187009.5_{33038.6}$	5310002
REFI gen 1	5063168	5124209	5176272	$5176624.5_{31945.7}$	5316194
REFI gen 2	5053050	5121611	5172418	$5172665.5_{31910.2}$	5295152
REFI gen 3	5045192	5118250	5170511	$5170641.0_{31708.1}$	5308670
REFI gen 4	5036054	5114363	5166370	$5166714.5_{31811.9}$	5275734
REFI gen 5	5033100	5108081	5159545	$5160104.5_{31627.9}$	5304274
REFI gen 6	5021506	5091967	5141618	$5141535.0_{30270.8}$	5261996
REFI gen 7	5018182	5058936	5102462	$5102097.0_{25329.6}$	5200178
REFI gen 8	5018182	5048834	5087568	$5084641.0_{17904.2}$	5153608
ESFI gen 1	5064096	5129209	5181559	$5181889.5_{32846.1}$	5307170
ESFI gen 2	5061370	5124526	5177912	$5178253.5_{32489.0}$	5312104
ESFI gen 3	5065478	5122971	5175219	$5175634.5_{32433.0}$	5304444
ESFI gen 4	5054256	5119025	5170738	$5171047.0_{32013.8}$	5288918
ESFI gen 5	5039096	5106205	5157359	$5157884.5_{31591.2}$	5283254
ESFI gen 6	5027022	5086195	5132262	$5132346.0_{28146.3}$	5232916
ESFI gen 7	5023610	5094308	5143031	$5142891.0_{29622.1}$	5252534
ESFI gen 8	5017024	5091101	5138713	$5138868.0_{28969.9}$	5238334
multi-start$_{90000}$	5045838	5133043	5186781	$5186889.5_{32964.5}$	5343300

Table A.10: Quality of the solutions for tai60a. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	7390306	7480823	7547714	$7548364.5_{41549.6}$	7718816
REFI gen 1	7408304	7469956	7535587	$7535738.0_{39787.3}$	7694328
REFI gen 2	7377916	7464064	7528840	$7529231.0_{39711.7}$	7689762
REFI gen 3	7365166	7463834	7526508	$7526913.5_{38854.9}$	7680518
REFI gen 4	7388740	7461063	7524574	$7524828.0_{39079.6}$	7654806
REFI gen 5	7363014	7455432	7519646	$7519943.0_{39319.9}$	7671852
REFI gen 6	7375208	7446550	7509378	$7509672.0_{38559.7}$	7674214
REFI gen 7	7322694	7423893	7483624	$7483851.5_{36744.4}$	7610188
REFI gen 8	7321632	7395713	7450103	$7449910.5_{32878.1}$	7579504
ESFI gen 1	7368098	7474779	7542200	$7541912.5_{40506.5}$	7689608
ESFI gen 2	7375510	7472263	7536184	$7536732.5_{40402.1}$	7691958
ESFI gen 3	7383938	7468625	7533693	$7534254.5_{40368.1}$	7694198
ESFI gen 4	7364126	7466962	7530445	$7530784.5_{39562.9}$	7688154
ESFI gen 5	7359572	7458935	7523814	$7524240.5_{39973.9}$	7680950
ESFI gen 6	7364390	7441710	7504444	$7504441.5_{38115.1}$	7645068
ESFI gen 7	7351906	7432209	7494194	$7493773.5_{37245.0}$	7661714
ESFI gen 8	7363172	7431633	7493303	$7493433.0_{37475.0}$	7640796
multi-start$_{90000}$	7372634	7481713	7548760	$7548856.5_{41104.5}$	7720164

Table A.11: Quality of the solutions for tai50b. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	460019200	469334422	491514704	$492297088.0_{15079275.0}$	560795776
REFI gen 1	458834944	462755248	481617232	$483789056.0_{15227086.0}$	551268544
REFI gen 2	458821504	459174080	460514400	$462097696.0_{6372497.0}$	534738432
REFI gen 3	458896928	460225536	460409216	$460460800.0_{487539.0}$	501801440
REFI gen 4	459194624	460225536	460396960	$460403808.0_{122913.8}$	461893216
REFI gen 5	460149280	460307424	460396960	$460419712.0_{100541.9}$	461546560
REFI gen 6	460149280	460307424	460440768	$460446560.0_{127234.4}$	461834688
REFI gen 7	460149280	460307424	460440768	$460456832.0_{135945.8}$	461949120
REFI gen 8	460149280	460307424	460442432	$460457760.0_{134727.1}$	461812800
ESFI gen 1	459318272	467149560	488284976	$489368128.0_{15344915.0}$	569918592
ESFI gen 2	460043328	465814897	480042912	$484013248.0_{14892598.0}$	551371136
ESFI gen 3	460713088	462353508	469842144	$470045728.0_{4768029.0}$	514776960
ESFI gen 4	460729024	461875520	466398400	$466454304.0_{2365897.8}$	476795136
ESFI gen 5	461431424	462597248	462597248	$462769344.0_{597814.8}$	467657600
ESFI gen 6	462316672	463986688	463986688	$463985760.0_{80095.0}$	464520928
ESFI gen 7	462803904	462803904	463404096	$463144384.0_{296389.4}$	463404096
ESFI gen 8	463404096	463404096	463404096	$463423936.0_{19842.1}$	463404096
multi-start$_{90000}$	458913472	469258948	491238624	$492124736.0_{15113656.0}$	575916032

Table A.12: Quality of the solutions for tai60b. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	609355200	623452550	656833568	$656662528.0_{20697538.0}$	732287040
REFI gen 1	608216512	611743132	626683424	$629156096.0_{14141397.0}$	699654336
REFI gen 2	608216512	621378368	626683584	$626597632.0_{4616621.0}$	685744192
REFI gen 3	623767680	624075776	626510304	$626395200.0_{2086134.8}$	636349632
REFI gen 4	623818624	624022464	624477760	$625653888.0_{2084760.6}$	636581760
REFI gen 5	623934144	623938816	624217728	$624534272.0_{1326390.0}$	636548160
REFI gen 6	623935552	623938816	624129664	$624122944.0_{301865.6}$	635483648
REFI gen 7	623934144	623935552	623938816	$623947520.0_{81396.4}$	624488128
REFI gen 8	623935552	623935552	623977920	$623978880.0_{104172.8}$	624594048
ESFI gen 1	608387520	616770499	636278272	$640118528.0_{17702224.0}$	706870016
ESFI gen 2	617880256	619691958	622211680	$623065024.0_{3118336.2}$	651764160
ESFI gen 3	619384832	619712384	620080256	$620425152.0_{1429504.0}$	636607808
ESFI gen 4	619601024	626998464	627137472	$626881280.0_{1068640.9}$	632455424
ESFI gen 5	621768960	622274688	623017792	$623327808.0_{1058790.0}$	625134848
ESFI gen 6	624292992	624292992	624292992	$624288448.0_{4543.9}$	624292992
ESFI gen 7	624292992	624292992	624292992	$624288448.0_{4543.9}$	624292992
ESFI gen 8	624292992	624292992	624292992	$624288448.0_{4543.9}$	624292992
multi-start$_{90000}$	608666880	623378672	656834016	$656634432.0_{20708170.0}$	740834112

Table A.13: Quality of the solutions for tai45e01. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	6554	8964	12689	$22200.7_{12553.5}$	268332
REFI gen 1	6412	8381	31901	$24559.1_{11524.5}$	268720
REFI gen 2	30472	30806	30814	$31217.0_{515.7}$	35526
REFI gen 3	30472	30472	30814	$31109.3_{543.0}$	34920
REFI gen 4	30472	30472	30814	$31078.0_{557.4}$	34306
REFI gen 5	30472	30472	30814	$31042.6_{637.5}$	34980
REFI gen 6	30472	30472	30806	$30907.6_{491.9}$	34728
REFI gen 7	30472	30472	30814	$31094.1_{580.0}$	35802
REFI gen 8	30472	30472	30806	$30957.8_{444.6}$	34348
ESFI gen 1	6412	7749	31892	$25615.2_{10635.0}$	40740
ESFI gen 2	30472	30644	31682	$31943.7_{1059.2}$	37860
ESFI gen 3	30644	30644	30710	$31001.6_{494.6}$	35048
ESFI gen 4	31272	31272	31320	$31317.2_{16.4}$	31320
ESFI gen 5	31320	31320	31320	$31324.6_{4.6}$	31320
ESFI gen 6	31320	31320	31320	$31324.6_{4.6}$	31320
ESFI gen 7	31320	31320	31320	$31324.6_{4.6}$	31320
ESFI gen 8	31320	31320	31320	$31324.6_{4.6}$	31320
multi-start$_{90000}$	6444	8940	12705	$22236.8_{12499.1}$	267652

Table A.14: Quality of the solutions for tai64c. 10,000 solutions per generation. The common starting point is a multi-start then $10,000$ local search methods are executed for each generation of algorithm (REFI and ESFI). The minimum, the 5th percentile of the best solutions, the median, the mean and the maximum are reported.

algorithm	min	5%	median	$mean_{std}$	max
multi-start$_{10000}$	1855928	1855928	1863678	$1864225.4_{9427.4}$	1955976
REFI gen 1	1855928	1855928	1863678	$1864032.6_{9131.2}$	1955976
REFI gen 2	1855928	1855928	1863678	$1863845.9_{8690.8}$	1955976
REFI gen 3	1855928	1855928	1863678	$1863434.9_{8147.6}$	1955976
REFI gen 4	1855928	1856396	1860942	$1862543.8_{6655.9}$	1934358
REFI gen 5	1855928	1856396	1860348	$1861328.9_{4973.1}$	1933266
REFI gen 6	1855928	1857646	1858710	$1860166.2_{4137.7}$	1907616
REFI gen 7	1855928	1857646	1857646	$1859543.2_{3665.3}$	1903972
REFI gen 8	1855928	1857646	1857646	$1858774.6_{3146.6}$	1903972
ESFI gen 1	1855928	1855928	1863678	$1864049.5_{9126.9}$	1955976
ESFI gen 2	1855928	1855928	1863678	$1864110.2_{8938.6}$	1955976
ESFI gen 3	1855928	1855928	1863678	$1863979.4_{9177.8}$	1955976
ESFI gen 4	1855928	1855928	1863678	$1863527.0_{8294.2}$	1955976
ESFI gen 5	1855928	1856396	1860942	$1862569.4_{6869.1}$	1955976
ESFI gen 6	1855928	1856396	1860942	$1861213.1_{4676.7}$	1913788
ESFI gen 7	1855928	1856396	1860942	$1861499.1_{4499.0}$	1903972
ESFI gen 8	1855928	1857646	1863678	$1863013.8_{3000.8}$	1926154
multi-start$_{90000}$	1855928	1855928	1863678	$1864672.1_{9265.7}$	1955976