Unsupervised Machine Learning for the Quadratic Assignment Problem

Thé Van Luong[1] and Éric D. Taillard[2]

[1] University of Lausanne, Service de la recherche, Bâtiment Amphipôle, CH-1015 Lausanne, Suisse
The-Van.Luong@unil.ch
[2] University of Applied Sciences of Western Switzerland, HEIG-VD, Department of Information and Communication Technologies, Route de Cheseaux 1, CH-1401 Yverdon-les-Bains, Suisse
eric.taillard@heig-vd.ch

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, quadratic assignment

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 [27], 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 [13]. Machine learning techniques 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 [4]. 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 [7].

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 [3]. Agrawal et al. [1] 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 [19] 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 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 problem (TSP) 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 [24, 25]. 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 [26] for a comparison of different methods based on neural graph machine network.

The remaining of this paper is organized as follows. 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 Sect. 4. Finally, Sect. 5 concludes and proposes future research avenues.

2 Technical Background

2.1 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 [5] 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 π of the set $\{1, 2, \ldots, n\}$ 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 exchanging 2 elements ($\frac{n \times (n-1)}{2}$ 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 [21].

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

2.2 Frequent Itemsets in Associative Rule Learning

In associate rule learning, the existence of very large databases requires determining groups of items that frequently appear together in transactions, called *itemsets* [2]. 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 [1]. 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.

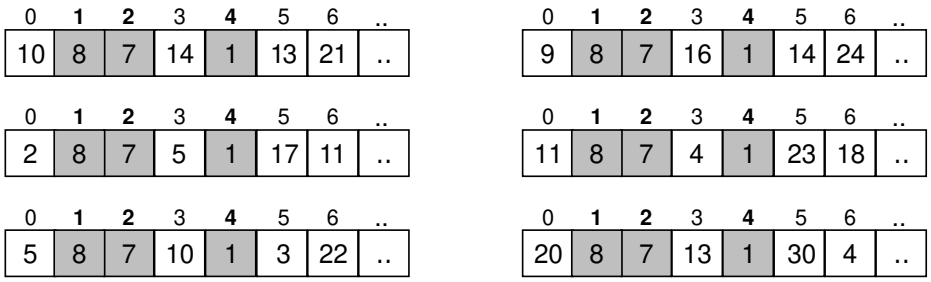

Fig. 1. Extraction of one frequent itemset of size 3. In all solutions, elements 8, 7 and 1 appear at positions 1, 2 and 4.

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 1 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).

In the main loop, the first phase consists in extracting all frequent itemsets from the current set of solutions (line 6) with Apriori algorithm [2]. 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.

Algorithm 1 Extraction and combination of frequent itemsets

Require: *instance_data*, *nb_solutions*, *nb_generations*, *min_sup* and *itemsets_limit*
for $i \leftarrow 1, \ldots, nb\_solutions$ **do**
  $solutions[i] \leftarrow random\_initialization()$
  $solutions[i] \leftarrow local\_search(instance\_data, solutions[i])$
end for
for $generation \leftarrow 1, \ldots, nb\_generations$ **do**
  $all\_itemsets \leftarrow extract\_itemsets(solutions, min\_sup, itemsets\_limit)$
  for $i \leftarrow 1, \ldots, nb\_solutions$ **do**
    $solutions[i] \leftarrow combine\_itemsets(all\_itemsets)$
    $solutions[i] \leftarrow local\_search(instance\_data, solutions[i])$
  end for
end for
Ensure: *solutions*

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 [2]. 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 k-itemsets in comparison with other approaches [11, 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 2 Apriori algorithm for the extraction of frequent itemsets

Require: *solutions*, *min_sup* and *itemsets_limit*
$k := 1$
$C_k = generate\_itemsets(solutions, \emptyset)$
$L_k = filter\_itemsets(C_k, min\_sup, \emptyset)$
$all\_itemsets = L_k$
while $L_k \neq \emptyset$ **do**
  $C_{k+1} = generate\_itemsets(solutions, L\_k)$
  $L_{k+1} = filter\_itemsets(C_{k+1}, min\_sup, itemsets\_limit)$
  $all\_itemsets = all\_itemsets \cup L_{k+1}$
  $k := k + 1$
end while
Ensure: *all_itemsets*

Algorithm 2 describes the major steps of Apriori used in the *extract_itemsets* procedure of Algorithm 1. 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, a 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 1.

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

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 a batch of solutions has been 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.

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.

Similarly, a buffer and a file are also required to reduce the space complexity.

4 Performance Evaluation

The computational results presented in this section have been 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 have been 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 10,000 candidate itemsets is reused accordingly.

This parallelization approach results in very good 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 [16].

4.1 QAP instances

The QAPLIB repository [6] 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.html`. Since it was not practically possible to

conduct our numerical experiments for all instances, only 11 QAP instances have been carefully selected. All selected instances are widely studied in the literature [15].

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 an exact method cannot solve most of them on modern computers.

The first 3 instances are from Skorin-Kapov [20] (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. [17] 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 [14].

Then, 2 symmetrical instances with flows and distances randomly, uniformly generated have been selected (tai50a and tai60a) [21]. These instances are similar to Roucairol's ones, but larger.

Then, 2 asymmetrical instances non-uniformly generated (tai50b and tai60b) comes from[22]. An instance for generating gray 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 [9] 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 1 the number of generations is set to 8 and 10,000 local searches are executed per generation.

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.

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. These parameters have been tuned in such a way that each generation does not exceed one day of calculation.

4.3 Quality of Solutions

For optimization problems, the main criteria to be evaluated is the quality of the solutions. The last is measured relative to the value of the best solution known to date (bvk), which is optimal for a few instances (lipa, tai64c and tai45e01) or believed to be optimal for the other ones. The distribution of the quality of the solutions is visualized with the proportion of runs having reached a solution below a given percentage above best known.

The quality of the solutions for the instances are graphically illustrated in Figure 2. 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.

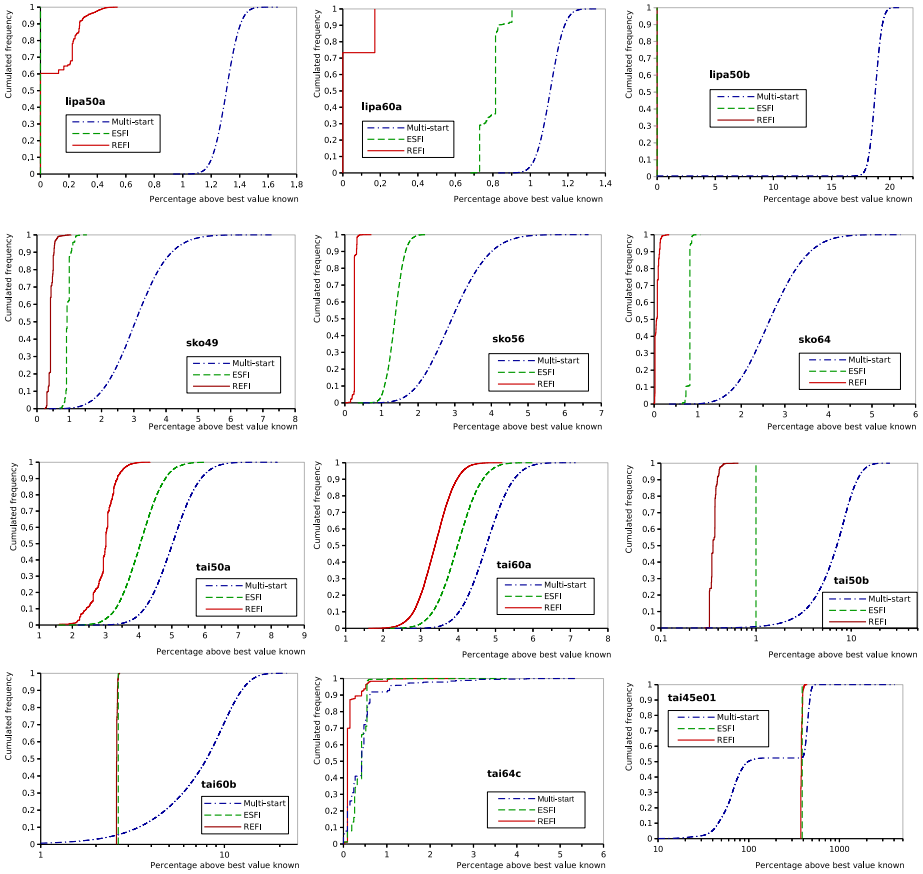

Fig. 2. Multi-start, ESFI and REFI distribution of solutions quality for lipa50a, lipa60a, lipa50b (asymmetric with known optimal solutions), sko49, sko56, sko64 (Manhattan distances on a square grid), tai50a, tai60a (uniformly generated), tai50b, tai60b (asymmetric and randomly generated), tai64c, tai45e01 (structured).

For the instance sko49, the distribution reveals that most solutions produced by REFI and ESFI algorithms are, respectively, about 0.5% and 1% above the bvk whereas multi-start produces solutions with a normal spread around 3% above the bvk. A similar observation can be made for the instance sko56. The phenomenon is more pronounced for the instance sko64 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, where the multi-start from random solutions was unable to find the optimum. A similar behavior stands for the lipa60a instance, except that REFI is the only algorithm able to reach the known optimum.

Regarding the lipa50b instance, the difference of quality is very 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, this is the only one for which learning with itemsets is highly successful.

For the tai50a instance, there is a moderate trend indicating that 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.

Regarding the instance tai50b, 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. The multi-start algorithm solutions 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 lipa..b instances.

Regarding the structured instance tai45e01, 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 optimum, for instance GRASP with path relinking, late acceptance local search, genetic hybrids or ant systems [9].

For tai64c, 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 [8].

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. The third 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 sharing 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 to improve 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%).

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. 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 1 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.

Table 1. Additional results for the positions of produced solutions for sko49, sko56, sko64, lipa50a, lipa60a, lipa50b 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, 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
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	$1.8_{1.4}$	88.77%	0.49%	99.99%	33.0×10^6
	multi-start	$1.2_{1.2}$	66.41%	0.09%	100.00%	-
	ESFI	$1.6_{1.3}$	85.50%	0.26%	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 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. It can be noticed that both REFI and SEFI got the optimal solution at the first generation, but this solution was lost for the remaining generations.

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 [21] or the fast ant systems [23] 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 [10].

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.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on Management of data. pp. 207–216 (1993)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: International Conference on Very Large Databases (VLDB). pp. 487–499 (1994)
3. Alpaydin, E.: Introduction to machine learning. MIT press (2020)
4. Birattari, M., Kacprzyk, J.: Tuning metaheuristics: a machine learning perspective, vol. 197. Springer (2009)
5. Burkard, R.E., Çela, E., Rote, G., Woeginger, G.J.: The quadratic assignment problem with a monotone anti-monge and a symmetric toeplitz matrix: Easy and hard cases. Math. Program. 82, 125–158 (1998)
6. Burkard, R.E., Karisch, S.E., Rendl, F.: Qaplib–a quadratic assignment problem library. Journal of Global optimization 10(4), 391–403 (1997), https://coral.ise.lehigh.edu/data-sets/qaplib/
7. Calvet, L., de Armas, J., Masip, D., Juan, A.A.: Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. Open Mathematics 15(1), 261–280 (2017)
8. Drezner, Z.: Finding a cluster of points and the grey pattern quadratic assignment problem. OR Spectr. 28(3), 417–436 (2006), https://doi.org/10.1007/s00291-005-0010-7
9. Drezner, Z., Hahn, P.M., Taillard, É.D.: Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. Annals OR 139(1), 65–94 (2005), http://mistic.heig-vd.ch/taillard/articles.dir/DreznerHT2005.pdf
10. Erhan, D., Courville, A., Bengio, Y., Vincent, P.: Why does unsupervised pre-training help deep learning? In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. pp. 201–208 (2010)
11. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. pp. 1–12. SIGMOD '00, Association for Computing Machinery, New York, NY, USA (2000), https://doi.org/10.1145/342009.335372

12. Koopmans, T.C., Beckmann, M.: Assignment Problems and the Location of Economic Activities. Econometrica 25(1), 53–76 (1957), http://www.jstor.org/stable/1907742
13. L'heureux, A., Grolinger, K., Elyamany, H.F., Capretz, M.A.: Machine learning with big data: Challenges and approaches. IEEE Access 5, 7776–7797 (2017)
14. Li, Y., Pardalos, P.M.: Generating quadratic assignment test problems with known optimal permutations. Computational Optimization and Applications 1(2), 163–184 (1992)
15. Loiola, E.M., de Abreu, N.M.M., Boaventura-Netto, P.O., Hahn, P., Querido, T.: A survey for the quadratic assignment problem. European journal of operational research 176(2), 657–690 (2007)
16. Luong, T.V., Melab, N., Taillard, É.D., Talbi, E.G.: 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), http://mistic.heig-vd.ch/taillard/articles.dir/LuongMTT2012.pdf, only preprint technical report available
17. Nugent, C.E., Vollmann, T.E., Ruml, J.: An experimental comparison of techniques for the assignment of facilities to locations. Operations Research 16(1), 150–173 (1968), https://doi.org/10.1287/opre.16.1.150
18. Park, J.S., Chen, M.S., Yu, P.S.: An effective hash-based algorithm for mining association rules. Acm sigmod record 24(2), 175–186 (1995)
19. Ribeiro, M., Plastino, A., Martins, S.: Hybridization of grasp metaheuristic with data mining techniques. J. Math. Model. Algorithms 5, 23–41 (04 2006)
20. Skorin-Kapov, J.: Tabu search applied to the quadratic assignment problem. ORSA Journal on computing 2(1), 33–45 (1990)
21. Taillard, É.D.: Robust taboo search for the quadratic assignment problem. Parallel Computing 17(4-5), 443–455 (1991)
22. Taillard, É.D.: Comparison of iterative searches for the quadratic assignment problem. Location Science 3(2), 87 – 105 (1995), http://www.sciencedirect.com/science/article/pii/0966834995000086
23. Taillard, É.D.: Fant: Fast ant system. Tech. rep., Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale (1998), IDSIA Technical Report IDSIA-46-98
24. Taillard, É.D.: A Linearithmic Heuristic for the Travelling Salesman Problem. EURO Journal of Operational Research 297(2), 442–450 (2022), http://mistic.heig-vd.ch/taillard/articles.dir/Taillard2021.pdf, available online June 2021
25. Taillard, É.D., Heslgaun, K.: POPMUSIC for the travelling salesman problem. EURO Journal of Operational Research 272(2), 420–429 (2019), http://mistic.heig-vd.ch/taillard/articles.dir/TaillardHelsgaun2018.pdf
26. Wang, R., Yan, J., Yang, X.: Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching (2020)
27. Zhou, Z.H., Chawla, N.V., Jin, Y., Williams, G.J.: Big data opportunities and challenges: Discussions from data analytics perspectives [discussion forum]. IEEE Computational Intelligence Magazine 9(4), 62–74 (Nov 2014)