A Parallel Tabu Search for the Large-scale Quadratic Assignment Problem

Omar Abdelkafi, Bilel Derbel, Arnaud Liefooghe

A Parallel Tabu Search for the Large-scale Quadratic Assignment Problem

Omar Abdelkafi
Univ. Lille, Computer science
CNRS, UMR 9189 - CRIStAL
Inria Lille-Nord Europe
F-59650 Villeneuve d'Ascq, France
omar.abdelkafi@univ-lille.fr

Bilel Derbel
Univ. Lille, Computer science
CNRS, UMR 9189 - CRIStAL
Inria Lille-Nord Europe
F-59650 Villeneuve d'Ascq, France
bilel.derbel@univ-lille.fr

Arnaud Liefooghe
Univ. Lille, Computer science
CNRS, UMR 9189 - CRIStAL
Inria Lille-Nord Europe
F-59650 Villeneuve d'Ascq, France
arnaud.liefooghe@univ-lille.fr

***Abstract*—Parallelization is an important paradigm for solving massive optimization problems. Understanding how to fully benefit form the aggregated computing power and what makes a parallel strategy successful is a difficult issue. In this study, we propose a simple parallel iterative tabu search (PITS) and study its effectiveness with respect to different experimental settings. Using the quadratic assignment problem (QAP) as a case study, we first consider different small- and medium-size instances from the literature and then tackle a large-size instance that was rarely considered due the its inherent solving difficulty. In particular, we show that a balance between the number of function evaluations each parallel process is allowed to perform before resuming the search is a critical issue to obtain an improved quality.**

***Index Terms*—Big Optimization, Iterative tabu search, Quadratic assignment problem.**

I. Introduction

Optimization problems underlying different application domains are becoming more and more difficult to solve, with the size of the underlying data increasing continuously. In this context, there is evidence that the optimization community is more and more considering the solving of large-scale optimization problems [1], [2]. Despite their high efficiency, state-of-art evolutionary algorithms and metaheuristics [3] are still facing a number of challenges to find a good solution in a reasonable amount of time.

One of the natural way to tackle large-scale problem instances is to benefit from parallel computing facilities and to explore strategies to set up parallel solving procedures [4]. However, understanding how to fully take advantage form the aggregated computing power and what makes a parallel strategy successful is a difficult issue, in particular when tackling large-size problem instances.

In this paper, we consider the Quadratic Assignment Problem (QAP) as a case study for setting and analyzing parallel solving techniques. The QAP was first introduced by Koopmans and Beckmann [5] to model a facility location problem. It can be described as the problem of assigning a set of facilities to a set of locations with given distance and flow between locations and facilities, respectively. The objective is to assign the facilities to locations in such a way that the sum of the products between flows and distances is minimized. More formally, the problem can be stated as follows:

$$\min_{p \in P} z(p) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)} \quad (1)$$

where f and d are the flow and distance matrices respectively, $p \in P$ represents a solution where p_i is the location assigned to facility i and P is the set of all n vector permutations. The objective is to minimize $z(p)$, which is the total cost assignment for the permutation p.

It is well-known that the size of the considered QAP instances significantly increases the difficulty of the solving procedure, even when considering heuristic and stochastic search procedures such as metaheuristics and evolutionary algorithms. Therefore, the QAP is a good candidate for studying the effectiveness of using a parallel search approach. In this work, we investigate the parallelization of an efficient sequential metaheuristic to solve QAP, called Ro-Ts [6]. Our primary goal is to study the behavior of this metaheuristic in a parallel context as a first step toward the setting of highly effective algorithms for solving large-scale instances.

In general, we are tempted to say that increasing the number of processes, keeping the same number of evaluations for each process, is enough to improve the performance of an algorithm. This is not always obvious and it depends on the structure of the instances as well as the behavior of the algorithm that can change in the parallel context. For the case of large-scale instances, we wish to study in particular the impact of two parameters. The first parameter is the number of processes run in parallel, and the second one is the sequential load to be performed by each process.

The rest of the paper is organized as follows. In Section II, we review some of the best-known parallel and sequential approaches to solve the QAP. In Section III, we describe the design of a parallel metaheuristics based on a master-worker architecture. In Section IV, we conduct an experimental analysis using small and medium instances, as well as one large problem instance from the literature. Finally, in section V, we conclude the paper and we propose some perspectives.

II. Background

The QAP can be considered as one of the hardest combinatorial problems in theory and practice.

This problem was introduced in [5] and later studied in an extensive manner. In particular, different tabu search approaches were proposed, e.g., [6]–[8].

One of the first efficient approach to solve the QAP is the robust tabu search (Ro-Ts) proposed by Taillard [6], where it is shown how to reduce the complexity of computing a fitness value of neighboring solutions using an incremental evaluation procedure. Many other works are based on Ro-Ts to solve the QAP, e.g., [7]–[9]. James et al. [7] proposed two categories of Tabu Search (TS) extending Ro-Ts. The first one is a continuous diversification process coupling a perturbation mechanism with TS. The second one is a discontinuous diversification TS process where a new starting solution is used to replace the current working solution in TS trajectory. In [9], a cooperative parallel tabu search algorithm for the QAP is introduced which is based on exchanging information throughout the execution of the algorithm. Later, Benlic et al. [8] proposed an Iterated Local Search (ILS) with a breakout strategy based on the history of the search. According to the evolution of the search, the ILS selects a perturbation degree, among a range of available ones, in order to better escape local optima.

One of the most important parallel work on QAP was proposed in [10]. In fact, a Parallel Multi-start Tabu Search is proposed with respect to the CUDA platform. Good quality solutions are obtained, however only the instances of Skorin-Kapov [12] are considered, and the biggest instance size tackled there-in is $n = 100$, which is still a medium scale. In [13], a parallel hybrid algorithm is composed of three steps. The first step is the generation. It consists in using a parallel Genetic Algorithm (GA) based on the island model. Each process represents an island and all the nodes execute a GA in parallel. The second step is a diversification. This diversification method is applied to all solutions. Finally, the global best solution obtained with the first two steps is used as a starting solution for the Ro-Ts. In [14], a cooperative variant of Iterative Tabu Search (ITS) is proposed. Each process performs an ITS in which a Ro-Ts is executed at each generation. After each iteration, each process sends its current solution, obtained by the Ro-Ts, to a neighboring process. Then, a distance is computed between the current solution and the solution received from the neighbor process. This distance is based on the similarity between the two solutions (the same facilities assigned to the same locations). According to this distance, the algorithm takes the decision to apply a known diversification strategy, to perturb randomly the solution or to generate a new random solution.

III. Parallel Iterative Tabu Search

For this work, we propose a simple Parallel Iterative Tabu Search (PITS) based on the state-of-the-art sequential Ro-Ts [6]. A defined number of processes (**NbrPro**) is launched. Each process executes an ITS. This number is important because it fixes the parallel deployment of the algorithm. This number can not be increased infinitely since it depends closely on the physical limit of the architecture on which it is executed. For this reason, it is important to effectively exploit the potential of each process.

Each ITS is a succession of Ro-Ts. The number of Ro-Ts calls represents the global iterations (**Giterations**). One of our motivations for using this algorithm is its efficiency on the QAP. Moreover, like all TS, the Ro-Ts is able to escape from local optima. For a good management of the parallelization, we want to control the sequential load to be executed by each process. For this reason, having a parameter such as the number of iterations performed by the Ro-Ts (**TSiterations**) is essential. We also would like to vary this parameter in order to find the right balance between the processes.

At each global iteration, a Ro-Ts is performed from different starting solutions. It is done for all the processes. A cooperation mechanism to exchange information between the different processes is used. The objective is to find a good starting solution to improve the global solution. In this approach, after each Ro-Ts, the current process sends the best solution to the master process. The master process select the global best solution. It sends the global best solution to all the processes. Each process receives the information and executes a random perturbation on the received global best solution. Only a part of the solution is changed. This allows the algorithm to keep a good structure for the next Ro-Ts. Many works from the literature, such as [7], [9], show that using the structure of the best solution found by the search can lead to a promising region of the search space. The random perturbation is considered as a diversification after each Ro-Ts to escape from stagnation and to explore new regions of the search space.

As you can note, in our algorithm design, two parameters are important. The first one is, how many processes the algorithm is going to use. It can be handled through *NbrPro*. The second one is how long time the execution will take for each process. It can be handled through the *TSiterations* parameter. In this work, we have based our experiment on these two parameters to see the effect of the parallel design setting.

The choice to have multiple parallel ITS is motivated by many distributed architectures using parallel CPU (Central Processing Unit) that we can have in several computing grids. CPUs are powerful for doing sequential calculations. This is well adapted to the multiple iterations that are executed by the ITS. Obviously, if our intention was to use another architectures such as the Graphic Processing Unit (GPU), we would have made a different choice like parallelizing the evaluations in order to take advantage of the GPUs power in the calculation.

Algorithm 1 is the pseudo-code of the PITS. Figure 1 illustrates the general flow of the algorithm. More information about the Ro-Ts can be found in [6].

Algorithm 1 The general PITS algorithm

Input: **perturb**: % perturbation; **cost**: cost of the current solution; **Fcost**: best cost found; **solution**: current solution; **LBsolution**: local best solution found; **GBsolution**: Global best solution found between all the processes; **NbrPro**: Number of parallel processes executed; **TSiterations**: Number of the Ro-Ts iterations; **Giterations**: Number of global PITS iterations.
for all P in *NbrPro* **do**
  Initialization of the random solution
  for i = 0 to *Giterations* **do**
    /* Sequential Ro-Ts framework*/
    for i = 0 to *TSiterations* **do**
      Ro-Ts iteration /* see algorithm 2 */
    end for
    /* End of the Ro-Ts framework*/
    if cost $<$ Fcost **then**
      /* improvement */
      Fcost = *cost*;
      Update the *LBsolution* with *solution*;
    end if
    if P is the master process **then**
      Receives all the *LBsolution* from the other processes
      Select the *GBsolution* between all the *LBsolution* processes;
      Sends the *GBsolution* to all the processes;
    else
      Sends the *LBsolution* to the master process;
    end if
    Receives the *GBsolution* from the master process
    Random **Perturbation** of *GBsolution*;
    Update a new solution to start a new global iteration;
  end for
end for

Algorithm 2 One TSiteration of the Ro-Ts

Input: **cost**: cost of the current solution; **solution**: current solution; **Fsolution**: best solution found; **Tcost**: best cost found inside the TS; **Tsolution**: best solution found inside the TS;
if movement is tabu but meets all aspiration criteria or is not tabu and is a new *Tcost* **then**
  Store the best permutation indexes that meet all conditions;
end if
Update tabu list;
Update *solution*;
if the *cost* is better than the *Tcost* **then**
  Update the *Tsolution* with *solution*;
end if
Update the delta matrix of the move costs;

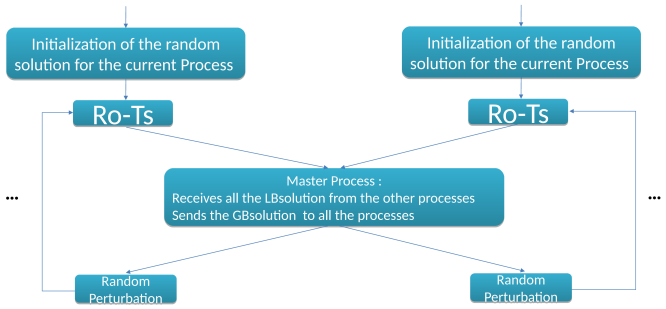

Fig. 1. PITS algorithm

IV. Experimental Analysis

A. Experimental setup

In our experimentation, the algorithm is written in $C/C++$. It is compiled on Grid'5000 [15] with a parallel Intel Xeon E5-2630 v4, 10 cores/CPU, 256GB RAM, 2x279GB HDD, 10Gbps ethernet.

The proposed algorithm is experimented on benchmark instances from the QAPLIB (http://www.seas.upenn.edu/qaplib/inst.html) [16] and instances from (http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html) [17]. The size of the considered instances ranges between 40 and 343. All the results are expressed as a percentage deviation from the Best Known Solution (BKS).

$$deviation = \frac{(solution - BKS) \times 100}{BKS} \quad (2)$$

The QAPLIB archive contains 136 instances that can be classified into four types:

- Real life instances (Type 1);
- Unstructured randomly generated instances based on a uniform distribution (Type 2);
- Randomly generated instances similar to real life instances (Type 3);
- Instances in which distances are based on the Manhattan distance on a grid (Type 4);

For each instance, 20 independent executions are performed and the average performance is considered. We shall focus on the percentage deviation in our comparison, although the execution time is also provided for completeness.

B. Results on small- and medium-size instances

The aim of this experiment is to confirm the efficiency of PITS algorithm against works from the literature. We start with a comparison on well-known small- and medium-size QAP instances. Two competitive algorithms from the literature have been chosen for this comparison.

- Cooperative parallel tabu search (CPTS) [9];
- Population-based iterated local search (PILS) [11].

TABLE I
PARAMETERS OF THE FIRST EXPERIMENTS

Parameter	Value
NbrPro	20
TSiterations	$100 \times n$
Giterations	100
percentage of perturbation	20%

TABLE II
RESULTS

Instances	BKS	PITS		CPTS		PILS	
		deviation	CPUt(min)	deviation	CPUt(min)	deviation	CPUt(min)
tai40a	3139370	0.216(0)	0.53	**0.148(1)**	3.5	0.280(0)	12.0
tai50a	4938796	0.486(0)	1.04	**0.440(0)**	10.3	0.663(0)	11.2
tai60a	7205962	0.532(0)	1.81	**0.476(0)**	26.4	0.820(0)	7.4
tai80a	13515450	**0.662(0)**	4.36	0.691(0)	94.8	0.927(0)	12.7
tai100a	21052466	0.597(0)	8.67	**0.589(0)**	261.2	1.027(0)	9.8
Average		0.499(0)	3.28	**0.469(1)**	79.24	0.743(0)	10.62

CPTS is a parallel algorithm using the number of calls to the objective function as a stopping criteria. This number is fixed to $10000 \times n \times n$ calls. For PILS, which is a sequential algorithm, the stopping criteria is fixed by the execution time (1 200 seconds for instance size less or equal to 100). Both proposals stop the algorithm as soon as the BKS is reached.

The focus of this comparison is solution quality. We use only 5 well-known benchmark instances from the QAPLIB (Type 2), which are the most difficult to solve. The other instances form the QAPLIB are easy to solve for our algorithm and the algorithms from the literature. The instance size ranges between 40 and 100.

This experiments use the parameters reported in Table I. Let us remind that *NbrPro* is the number of processes, *TSiterations* is the number of iterations executed by each Ro-Ts, *Giterations* is the number of Ro-Ts calls and the percentage of perturbation is the percentage of the solution that changes randomly with respect to the global best solution. For example, for an instance of size 100, 20 assignments of the global best solution will be modified randomly.

For this first experiments, we did our best to be as fair as possible when comparing PITS with the CPTS algorithm. The number of objective function calls is set to $200\,000 \times n$ ($20\,processes \times 100\,Giterations \times (100 \times n)\,TSiterations$). For CPTS, it is fixed to $10000 \times n \times n$, which is always more than the number of calls that we use for our algorithm. Indeed, for $n = 20$, we use the same number of calls as CPTS. It means that from a size of 20, our algorithm uses less budget. Indeed, the smallest instance size considered in this paper is 40.

Table II reports the obtained results from all algorithms. The CPU time (CPUt) is computed in minutes. It represents the average CPU time from 20 independent runs. The number in brackets with the deviation is the number of runs where the algorithm reached the BKS.

For the experiments of Table II, the best global average is obtained by the CPTS [9] algorithm with 0.469%. It is followed by PITS with a very close average results of 0.499%. Notice that the proposed PITS obtains better results than the PILS algorithm [11]. Of course, the execution time depends on the architecture used, but it is important to note that our algorithm has a time average of 3.28 minutes, against 79.24 minutes for CPTS.

These preliminary experiments show that our proposed PITS algorithm is able to obtain almost the same global performances (CPTS outperforms PITS with only 0.03% for the average of the 5 instances) in less budget in terms of CPU time and number of objective function calls.

The experimental results show the efficiency of our algorithm. Indeed, PITS obtains a good performance (less than 0.5% deviation from the BKS) for this reference benchmark.

C. Case study: large-size instance tai343e01

In the rest of the paper, we analyze our proposal using one large instance called tai343e01, from (http://mistic.heig-vd.ch/taillardproblemes.dir/qap.dir/qap.html). In order to give an idea about its complexity, let us mention that the number of constraints for this instance is 61 202. To our knowledge, only [18] and [19] attempted to solve this instance in the past. In these two works, few results about *tai343e01* are actually reported. It is considered as very hard to solve, even for metaheuristics.

The first result is reported in [18], where the deviation from the BKS is 18%. In [19], several algorithms are compared and the best result from 20 independents trials is given. The algorithms used in the comparison are: Ro-Ts [6], ILS-ES [11], as well as two variants of the Hierarchical Iterated Local Search (HILS(2) and HILS(3)). The best results are obtained by HILS(3) with 0% deviation from the BKS (for the best trial between the 20 independents run). The second variant HILS(2) show a deviation of 59%, the ILS-ES 4% and the Ro-Ts 34%.

The parameters from Table III are used in the following experiment. The major contribution of our work is the setting of the algorithm in a parallel context. As already explained in section III, we focus our analysis on two parameters. The variation of the processes number (NbrPro) and the variation of the iteration number executed by Ro-Ts (TSiterations). To better compare our results, we consider the work from [18] where the sequential budget (in terms of number of function evaluations) used for the Ro-Ts is 1 619 120. We use a parallel design to get competitive results while remaining close from the budget used in the literature. We decided to choose our settings so that we could have a budget close to the literature as well as larger and smaller budgets, in order to analyze the impact.

Figure 2 shows the average deviation of form 20 independent runs using 20 parallel processes. At each global iteration, the deviation of the best global solution is reported. A number of 100 Ro-Ts are iterated with 1 500 iterations for each one of them.

Our algorithm gets an average deviation of 14.21% from the BKS solution (as shown in Figure 2) and a best trial of 11.61% (best solution between 20 independents executions).

TABLE III
PARAMETERS OF THE LARGE SCALE EXPERIMENTS

Parameter	Value
NbrPro	{10, 20, 40}
TSterations	{350, 750, 1500}
Giterations	100
percentage of perturbation	20%

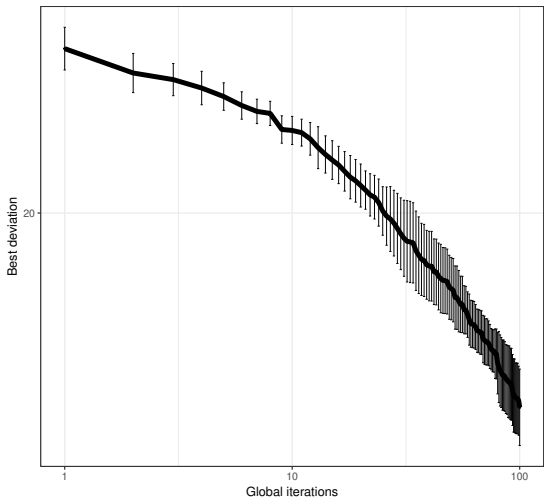

Fig. 2. Average deviation of 20 trials (20 processes and 1500 TSiterations)

The average execution time for the 20 runs is 24.17 minutes in this case. The curve also shows the ability of the algorithm to converge towards better solutions without notable stagnation. The total budget used in this experimentation is 3 000 000 evaluations (1 500 TSiterations $\times$ 100 Ro-Ts $\times$ 20 processes).

It is important to notice that we have experimented the same algorithm with the same parameters but without the cooperation between the processes through the exchange of the best global solution. The results obtained are much lower than the results obtained by the version that uses the exchange. Indeed, without the exchange we manage to obtain an average deviation of 22.73% from the BKS. It gets a deviation of 21.28% for the best trial. It confirms that the cooperation with the best global solution has a positive effect on the behavior of our algorithm.

For the work of [18], using 1 619 120 sequential evaluations, a deviation of 18% for the best trial is reported. For our algorithm, we have a sequential budget of 150 000 evaluations (1 500 TSiterations $\times$ 100 Ro-Ts). We can observe that with a very little sequential budget (less than 10% of what it is used in the literature) but with a multistart iterative Ro-Ts and an effective exchange of the best solution, we get a better results than the 18% found in [18]. This suggest that the strength of a tabu search approach for this instance lies in the perturbation step rather than in the iterative search. This also suggests that for this large instance, we need much less iterations in each TS in order to get good performance, which is an ideal framework for parallelization.

TABLE IV
RESULTS SHOWING THE VARIATION OF ITERATIONS OR PROCESSES

NbrPro/TSiterations	Budget	Deviation	CPUt(min)
10/350	350 000	17.22%	6.28
10/750	750 000	17.59%	12.48
10/1500	1 500 000	17.96%	24.11
20/350	700 000	15.11%	6.47
20/750	1 500 000	15.01%	12.52
20/1500	3 000 000	14.21%	24.17
40/350	1 400 000	**12.19%**	**13.01**
40/750	3 000 000	12.32%	25.89
40/1500	6 000 000	13.06%	49.69

In order to validate this hypothesis, two scenarios are considered, the first one is the reduction of the number of TSiterations from 1 500 to 750 and 350 iterations while keeping the same number of processes (lower budget). The number of processes can be 10 or 20 or 40 in our case. The second experiment is to keep the same budget, but with a different distribution (variation of the number of processes). For example, the budget of 1 500 000 can be distributed with (1 500 TSiterations $\times$ 100 Ro-Ts $\times$ 10 processes) or (750 TSiterations $\times$ 100 Ro-Ts $\times$ 20 processes).

Table IV shows the average deviation of our experiments for the 20 independent runs. The CPU time (CPUt) is computed in minutes. It represents the average time for the 20 independent runs. Naturally, the execution time increases with the increase of the number of iterations, because the sequential part is more important. For example, for 20 processes and 750 TSitrations, the average time is 12.52 minutes. This average increases to 24.17 minutes when we increase the number of TSiterations to 1 500. On the other hand, if we keep the same number of TSiteration and we increase the number of parallel processes, the average time remains constant. For example, for a TSiteration of 350, using 10 processes or 20 processes gives the same average time of 6 minutes. We can observe that this does not remain true when we use 40 processes. This is simply due to a physical limitation of the architecture, because we actually only have 20 completely parallel CPUs at our disposal for the experiments. In this case, the last 20 processes are forced to wait for the first 20 processus, which has the effect of doubling the execution time. If we had 40 CPUs completely parallel we could have obtained exactly the same results in twice less time. If we take into account only the quality of the solutions, two main informations can be extracted from this table.

*1) **Impact of the number of iterations**:* Firstly, when fixing the same number of processes and varying the number of iterations, we do not observe any significant change in performance, even with a much larger budget. Indeed, whether it is for 10, 20 or 40 processes, we almost have the same deviation even with a twice same budget.

From Figure 3, we can observe the evolution of the average deviation (for 20 independents runs) through the algorithm

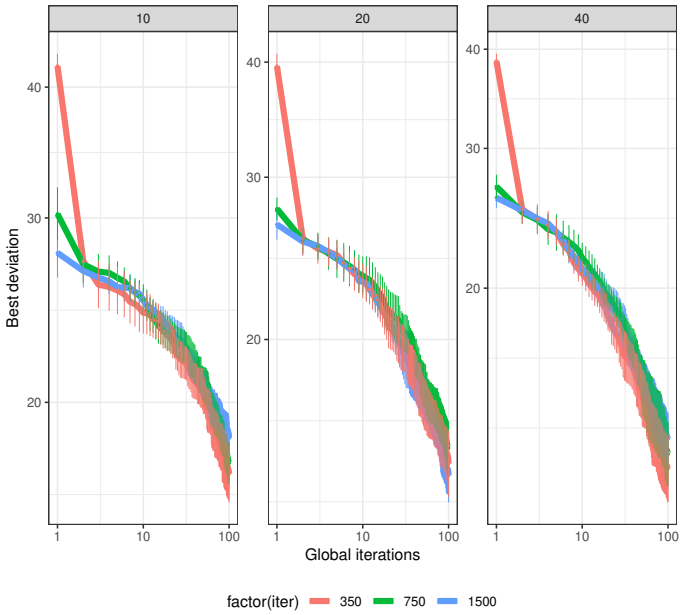

Fig. 3. Impact of increasing the number of iterations

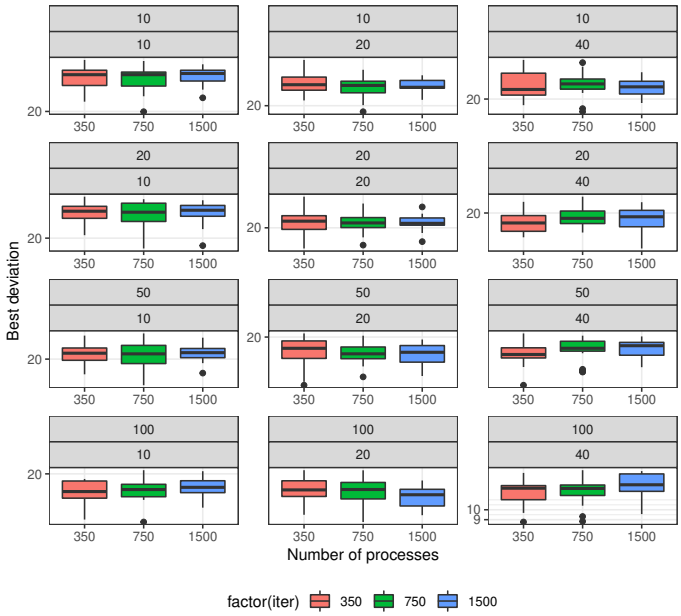

Fig. 4. Impact of increasing the number of iteration – details.

progress. For 10 and 40 processes, increasing the number of TSiterations from 350 to 1 500 does not improve quality. The only change that can be noted (but which remains very small and cannot be distinguished on the figure) is when using 20 processes. In this case, doubling the budget from 1 500 000 to 3 000 000 evaluations brings an improvement of 0.8% in the average deviation.

Our conclusion is confirmed in Figure 4, showing the best solution obtained by each algorithm setting after 10, 20, 50 and 100 global iterations (Giterations). We want to see if, at any moment of the execution, one of the configuration takes the advantage. All we can say is that for all the reported results, none of the configurations take the advantage, considering the same number of processes.

These results suggest that increasing the number of iterations in the TS does not help the search process while increasing the overall number of evaluations. Moreover, for an equivalent budget, the experiments show that it is more beneficial to set a smaller number of iterations. For example, for a budget of 1 500 000 evaluations, it is more beneficial to have 20 processes with 750 iterations each (which gives an average deviation of 15.01%) than 10 processes with 1500 iterations each (which gives an average deviation of 17.96%)

*2) **Impact of the number of processes**:* The second main finding from Table IV is the behavior of the algorithm when using the same number of iterations and varying the number of processes. Increasing the number of processes has obviously the effect of increasing the overall budget. Nevertheless, unlike the first case, this increase leads to a clear improvement of performance. Indeed, if we consider a setting with 750 TSiterations, doubling the budget from 1 500 000 (20 processes and 750 TSiterations) to 3 000 000 (40 processes and 750 TSiterations) allows us to improve the average deviation by 2.69% (from 15.01% to 12.32%).

Figure 5 clearly shows that scaling the number of processes has then a positive impact on the results. The use of 40 parallel processes, with the same budget for each process, improves significantly the average deviation. This is confirmed in Figure 6 showing the obtained deviation at different steps of the search process (i.e., using different termination criteria). Through this figure, we can observe the best solution obtained by each algorithm setting after 10, 20, 50 and 100 global iterations (Giterations). If we consider the case of the 350 iterations (left column), the use of 40 processes outperforms the use of 20 and 10 processes for all the iterations level (10, 20, 50 and 100) that can be seen in Figure 6. This results means that the increase of the budget with the increase of the number of processes is beneficial.

D. Discussion

In this work, we propose a very simple parallel model to solve QAP, with a particular focus on the tai343e01 instance. The landscape of this instance seems to need a very small number of iterations (few intensification search). Indeed, increasing the budget has no impact, unless we increase the number of processes. Through this point, we try to show that, in the parallelization, it is essential to find the good distribution, so that the search process becomes more efficient.

Generally speaking, in the literature, the use of parallelization is focused on maximizing the budget without worrying about the landscape and how to distribute this budget.

A good instance for parallelization is an instance that can be solved efficiently when we maximize the number of processes

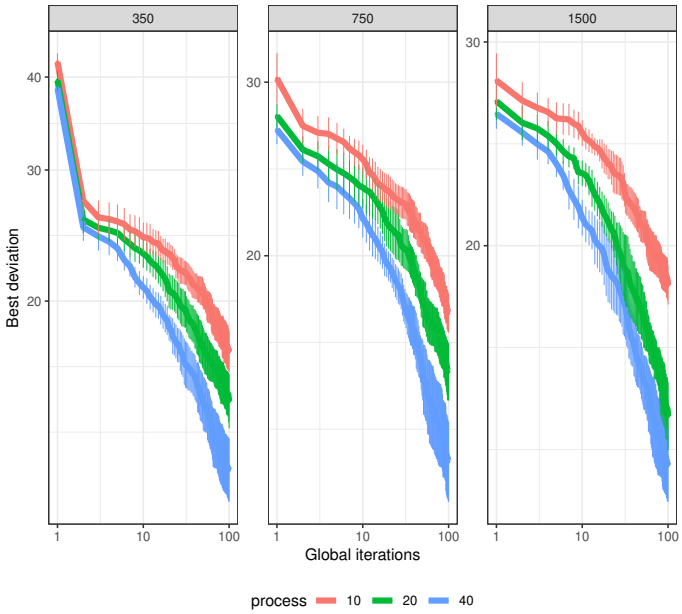

Fig. 5. Impact of the variation in the number of processes.

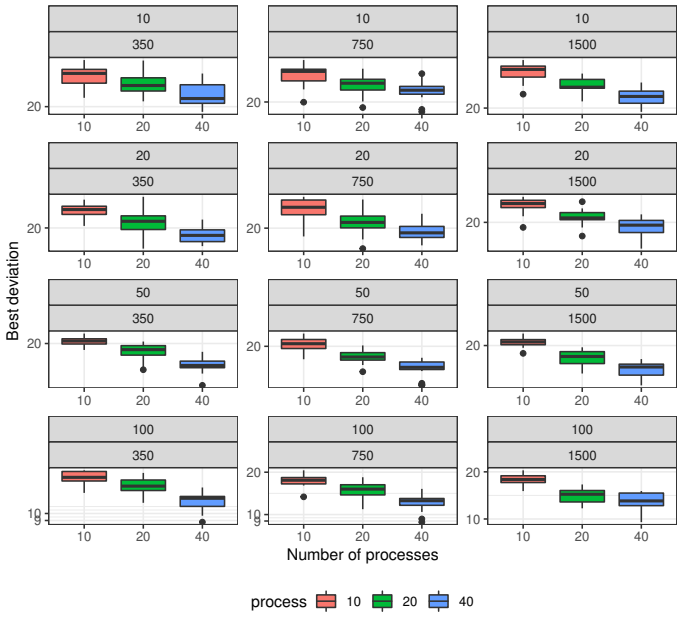

Fig. 6. Impact of the variation in the number of processes.

TABLE V
20 TRIALS FOR THE TAI343E01 INSTANCE (40 PROCESSES/350 ITERATIONS)

Instance(20 trials)	PITS	
	Deviation	times(min)
tai343e01-1	10.82	12.82
tai343e01-2	12.75	13.09
tai343e01-3	11.15	13.03
tai343e01-4	14.80	13.04
tai343e01-5	14.58	12.98
tai343e01-6	11.38	13.00
tai343e01-7	10.64	13.04
tai343e01-8	12.93	13.03
tai343e01-9	11.93	13.05
tai343e01-10	**8.78**	**13.00**
tai343e01-11	12.90	12.96
tai343e01-12	12.58	12.98
tai343e01-13	12.57	13.02
tai343e01-14	13.45	13.05
tai343e01-15	13.41	13.06
tai343e01-16	12.93	13.01
tai343e01-17	12.38	13.01
tai343e01-18	11.01	13.02
tai343e01-19	12.93	12.95
tai343e01-20	9.67	13.02

and minimize the execution time on each process. This is the case of instances that need many search from multiple starting solutions such as the instance tai343e01.

The best results we could get are with 350 iterations $\times$ 100 Ro-Ts $\times$ 40 processes (a total budget of 1 400 000). The average deviation from the BKS on 20 independent executions is 12.19% and we obtain a deviation of 8.78% for the best attempt in only 13 minutes. These results outperform the deviation obtained in the literature with the Ro-Ts algorithm [18] using a smallest budget (1 400 000 evaluations against 1 619 120 for [18]).

Table V shows the average deviation as well as the calculated time in minutes for the 20 independent executions of our best configuration (350 iterations $\times$ 100 Ro-Ts $\times$ 40 processes).

The results obtained in this paper are easily exploitable and interesting on hybrid genetic algorithms. Indeed, each process can represent an individual of the population. The random perturbation can be replaced by a crossover between the best solution found in the population and the best local solution of each process. The hybridization of the algorithm can be done thanks to the tabu search proposed in this work. Moreover, the analysis shows that in the case of the instance studied in this article, it is beneficial to increase the number of processes that can execute a small number of iterations. This configuration can be very effective in the context of a hybrid genetic algorithm.

Another obvious use of this paper results, is the improvement of the existing evolutionary algorithm. For example, in the case of [13], it is possible to improve the last part of their algorithm with the parallelization of the tabu search applied to the best individual obtained by their genetic algorithm.

V. Conclusions and perspectives

In this work, we have presented and experimented a parallel approach for QAP. This approach demonstrates efficient results on the set of benchmark instances experimented. We evaluated our approach on 5 benchmark instances from the QAPLIB, and we showed that the algorithm was able to obtain competitive results compared against several variants from the literature. We also analyzed the tai343e01 instance from [17]. This analysis represents the main motivation for this work.

In summary, two main contributions are proposed in this work. The first one is to propose a parallel variant of the Ro-Ts, able of performing well on small- to medium-size instances as well as on large-size instances, simply by changing a few set of parameters. The second, and most important contribution, is a deeper analysis of a large-size instance for the well known QAP, in order to find the right distribution of the budget in the parallel context.

In future works, there are several possible ways to extend this work. One possibility is to experiment the impact of other potential solution exchange strategies through the creation of a set of local optima produced by the parallel processes, and by choosing one of them following a particular strategy. Another possibility is to study the landscape of the other large-size instances in order to propose a good tuning to solve these instances and to see if they share a similar landscape topology.

Acknowledgement

Experiments presented in this paper were carried out using the Grid'5000 testbed (http://www.grid5000.fr). Therefore, we would like to thank the Grid'5000 team. Grid'5000 is supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations.

References

[1] S. Mahdavi, M. E. Shiri, S. Rahnamayan, Metaheuristics in large-scale global continuous optimization: A survey, Information Sciences, Elsevier 295, pp 407-428, 2015.

[2] X. Li, K. Tang, P. N. Suganthan, Z. Yang, Nature-inspired algorithms for large scale global optimization, Information Sciences 316,437-439, 2015.

[3] C. Blum, A.Roli, Metaheuristics in combinatorial optimization : Overview and conceptual comparison, ACM Computing Surveys 35, pp 368-308, 2003.

[4] O. Abdelkafi, L. Idoumghar, J.Lepagnot, A survey on the metaheuristics applied to QAP for the graphics processing units, Parallel Processing Letters 26, pp 1-26, 2016.

[5] T. Koopmans, M. Beckmann, Assignment problems and the location of economic activities, Econometrica, vol. 25, no. 1, pp. 53-76, 1957.

[6] E. Taillard, Robust taboo search for the quadratic assignment problem, Parallel computing 17, pp. 443-455 ,1991.

[7] T. James, C. Rego, F. Glover, Multistart Tabu Search and Diversification Strategies for the Quadratic Assignment Problem, IEEE TRANSACTIONS ON SYSTEMS, Man, And Cybernetics-part a: systems and humans, vol. 39, no. 3, May 2009.

[8] U. Benlic, J.K. Hao, Breakout local search for the quadratic assignement problem, Applied Mathematics and Computation 219, pp. 4800-4815, 2013.

[9] T. James, C. Rego, F. Glover, A cooperative parallel tabu search algorithm for the quadratic assignment problem, European Journal of Operational Research 195, pp. 810-826, 2009.

[10] M. Czapinski, An effective Parallel Multistart Tabu Search for Quadratic Assignment Problem on CUDA platform, J. Parallel Distrib. Comput. 73, pp. 1461-1468, 2013.

[11] T. Stützle, Iterated local search for the quadratic assignment problem, European Journal of Operational Research 174, pp. 1519-1539, 2006.

[12] J.S. Kapov, Tabu search applied to the quadratic assignment problem, ORSA Journal on Computing, 2(1) pp.33-45, 1990.

[13] U. Tosun, On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem, Engineering Applications of Artificial Intelligence 39, pp 267-278, 2015.

[14] O. Abdelkafi, L. Idoumghar, J. Lepagnot, Comparison of two diversification methods to solve the quadratic assignment problem, Procedia Computer Science 51, pp 2703-2707, 2015.

[15] R.B. et al, Grid'5000: A large scale and highly reconfigurable experimental grid testbed, IJHPCA 20(4), pp 481-494, 2006.

[16] R.E. Burkard, S.E Karisch, F. Rendl, QAPLIB - A quadratic assignment problem library, journal of global optimization Volume: 10 Issue: 4, pp. 391-403, Jun 1997.

[17] Z. Drezner, P. M. Hahn, E.D. Taillard, Recent Advances for the Quadratic Assignment Problem With Special Emphasis on Instances that are Difficult for Meta-Heuristic Methods, Annals of Operations research 139, pp 65-94, 2005.

[18] Z. Drezner, P. M. Hahn, E.D. Taillard, Recent Advances for the Quadratic Assignment Problem With Special Emphasis on Instances that are Difficult for Meta-Heuristic Methods, Annals of Operations research 139, pp 65-94, 2005.

[19] M. S. Hussin, T. Stützle, Hierarchical Iterated Local Search for the Quadratic Assignment Problem, IRIDIA - Technical Report Series, 2009.