Solving QAP with Auto-parameterization in Parallel Hybrid Metaheuristics

Jonathan Duque[1][0000−0002−3672−1630], Danny Múnera[1][0000−0003−0762−0571], Daniel Diaz[2][0000−0002−2700−2271], and Salvador Abreu[3][0000−0002−1613−4631]

[1] Facultad de Ingeniería, Universidad de Antioquia, Medellín Colombia
`jonathan.duque@udea.edu.co` `danny.munera@udea.edu.co`
[2] CRI, Université Paris 1, Paris France
`daniel.diaz@univ-paris1.fr`
[3] NOVA-LINCS, Universidade Évora, Évora Portugal
`spa@uevora.pt`

Abstract. The Quadratic Assignment Problem (QAP) is one of the most challenging combinatorial optimization problems with many real-life applications. Currently, the best solvers are based on hybrid and parallel metaheuristics, which are actually highly complex and parametric methods. Finding the best set of tuning parameters for such methods is a tedious and error-prone task. In this paper, we propose a strategy for auto-parameterization of QAP solvers. We show evidence that auto-parameterization can further improve the quality of computed solutions. Our auto- parameterization scheme relieves the user from having to find the right parameters while providing a high quality solution.

Keywords: QAP · Auto-parametrization · Heuristics · Parallelism

1 Introduction

The Quadratic Assignment Problem (QAP) is a hard combinatorial optimization problem with many real-life applications such as scheduling, facility location, electronic chipset layout, production, process communications, among many others [1]. QAP has been shown to be NP-Hard and finding effective algorithms to solve it is an active research topic in recent years.

Medium size problems can be solved using exact methods (e.g., size ≤ 30), which can find an optimal solution or prove that a problem has no solution [2]. Exact methods consider the entire search space: either explicitly by exhaustive search or implicitly, by pruning some portions of the search space that have been detected as irrelevant for the search.

To tackle harder problems, one must resort to incomplete methods which provide good, albeit potentially sub-optimal solutions in a reasonable time. Such is the case for metaheuristics, which are high-level procedures that make choices to efficiently explore part of the search space, so as to make problems tractable. Metaheuristics usually have several parameters to adjust their behavior depending on the problem to solve [3]. Examples of metaheuristics include genetic algorithms, tabu search, local search and simulating annealing.

Metaheuristics operate on two main working principles: intensification and diversification. The former refers to the method's ability to explore more deeply a promising region of the search space, while the latter refers to the exploration of different regions of the search space. By design, some metaheuristics methods are better at intensifying the search while others are so at diversifying it. However, the behavior of most metaheuristics can be controlled via a set of parameters. A fine tuning of these parameters is therefore crucial to achieve an effective trade-off between intensification and diversification, and hence good performance in solving a given problem. Unfortunately, selecting the best set of parameters is a tedious and error-prone task. This process is even harder because the best parameters values vary with the problem structure and even just for different instances of the same problem, as stated by the Non-Free-Lunch theorem [4].

Each metaheuristic has its own strengths and weaknesses, which may vary according to the problem or even to the instance being solved. The trend is thus to design hybrid metaheuristics, which combine diverse methods in order to benefit from the individual advantages of each one [5]. However, this increases the number of parameters (parameters of individual metaheuristics and new parameters to control the hybridization). The design and implementation of a hybrid metaheuristic is a complex process; tuning the resulting parameters, to reach the best performance, is also very challenging.

Despite the good results obtained using hybrid metaheuristics, it is still necessary to reduce the processing times needed for the hardest instances [6]. One of the most plausible options entails parallelism [7]. In parallel metaheuristics one can have multiple instances of the same (or different) metaheuristics running in parallel, either independently or cooperatively through concurrent process communications [8,9]. Not only does parallelism help to decrease processing time, but it can also be a means to easily implement hybridization.

In previous work we proposed a Cooperative Parallel Local Search solver, called CPLS [10,11]. CPLS embeds various simple local search metaheuristics and then relies on cooperative parallelization to concurrently execute several metaheuristic instances, which cooperate during the search process. We later extended CPLS, by proposing PHYSH (Parallel HYbridization of simple Heuristics) [12,13]. PHYSH supports the combination of population-based and single-solution metaheuristics. CPLS and PHYSH also require the fine tuning of a larger number of parameters, since more metaheuristics (of different types) are involved. Moreover, the configuration of the parallel interaction itself (communication between the methods) involves yet another set of parameters which need to be adjusted. Tuning this increasing number of parameters makes it even more difficult to find the appropriate setting for the algorithm to behave optimally.

Automating the task of finding good parameters is thus desirable and has attracted significant attention from researchers. We may identify two kinds of strategies for automatic tuning: *parameter tuning* and *parameter control* [14]. In parameter tuning (off-line tuning) the set of parameters are defined before applying the algorithm to a specific problem (static definition of parameters). Several strategies for automatic parameter tuning of metaheuristics have been proposed [15,16]. In contrast, parameter control strategies (online-tuning) adapts the values of the controlled parameters during the algorithm execution (dynamic adaptation of parameters). The idea is to find the best parameters setting during the solving process, using some mechanism to alter the parameter values according to the algorithm performance.

Parameter tuning can be seen as a pre-process pass which is executed before the solving in order to determine the adequate values for parameters. This does not affect the implementation of the solver. On the other hand, *parameter control* has to be implemented in the kernel of the solver. The former may appear easier but when the number of parameters become large it is hard to use in practice. Indeed, it usually requires many runs to identify the best parameter settings, making this a time-consuming process. These methods are often limited by the number of parameters and the computational power available. In that case, parameter control strategies emerge as a viable solution to deal with the high complexity of current solvers (hybrid and/or parallel).

In this paper we propose a parallel hybrid method with a parameter control strategy for solving the QAP, called DPA-QAP. DPA-QAP embeds multiple metaheuristic methods in a parallel hybrid execution and self-adapts the parameters of the metaheuristics using an iterative process, adaptation is performed based on performance measures. We carried out an experimental evaluation which shows that the auto-parametrization strategy outperforms a simpler version of DPA-QAP with no auto-parametrization, i.e., a parallel hybrid method with static parametrization. We perform the evaluation using the classical QAPLIB instances and also a particular set of very hard QAP instances.

In the remaining of this paper we present the related work on Section 2. Section 3 presents the general structure of DPA-QAP and Section 4 introduces the auto-parametrization strategy. Section 5 contains the experimental evaluation performed which validates our strategy. A short conclusion ends the paper.

2 Related work

The Quadratic Assignment Problem (QAP) was first proposed by Koopmans and Beckmann in 1957 [17] as a model for a facilities location problem. This problem consists in assigning a set of n facilities to a set of n locations, while minimizing the cost associated with the *flows* of items among facilities and the *distance* between them.

Metaheuristic methods have been successfully applied for solving QAP. From the 90s, several metaheuristic methods have emerged as a suitable option to solve this problem, e.g., Tabu Search [18], Genetic Algorithms [19], among several others. These methods perform well on a wide range of QAP instances, however, some hard instances still require very long runs to achieve quality solutions. Moreover, no method was able to get good performance on an extensive set of instances. The aforementioned problems spurred the emergence of new techniques based on hybridization and parallelization. For instance, one of the fundamental methods of hybrid metaheuristics is the memetic algorithm (MA) [20]. MA is an effective approach which combines an evolutionary algorithm with a local search procedure. Hybrid metaheuristics are intricate procedures, tricky to design, implement, debug and tune, therefore, it is unsurprising that hardly any of them only combine more that a couple of methods.

Parallelizing metaheuristics grants access to using powerful computational platforms with the aim of speeding up the search process [21]. A straightforward implementation of parallel metaheuristics is the *Independent multi-walks* approach which speeds up the search process by performing concurrent executions of multiple metaheuristic instances, therefore augmenting the probability to get quickly a good solution [22]. Another kind of parallel metaheuristics allows the concurrent instances to cooperate by exchanging information during the search process, aiming to improve the efficiency of the solver [23,24]. We identify these methods as *Cooperative multi-walk* approaches.

We proposed a way to create hybridization through cooperative parallelization in our CPLS framework [10,11]. CPLS allows the user to code the individual metaheuristics, and the framework manages parallelism and communications. In CPLS, different local search metaheuristics concurrently interact by exchanging relevant information about the search. This interaction provides a cooperative way to intensify the search. This framework has been successfully used to solve hard variants of Stable Matching Problems [25] and hard instances of QAP [26,11]. Since CPLS does not support population-based methods, we proposed an extension of the framework called PHYSH [12,13], which provides an efficient strategy to promote cooperation between population-based and single-solution methods, metaheuristics of a different nature. Both CPLS and PHYSH have proved able to efficiently solve several hard QAP instances.

Parallel hybrid metaheuristics often have many parameters which modify the algorithm behaviour. Setting these parameters has influence on the performance of the method, however, finding the optimal values for these parameters is usually a hard task [15]. Using hybridization and parallelism makes this task even more difficult for mainly two reasons: First, hybrid metaheuristics inherit the parameters of each "low level" metaheuristic, so one needs to find the setting of more parameters, since a parameter configuration for one algorithm usually is not suitable for another. Second, cooperative parallel strategies require parameters to define their behaviour, e.g., for determining how frequently metaheuristics should interact or how each metaheuristic has to use the received information,...

Tuning metaheuristic parameters (i.e., *offline-tuning*) has been carried out in different ways, in earlier times the tuning process was done by hand, another approach was to take parameters values from similar algorithms reported in the literature. More recently, the use of specialized tools for automatic parameter tuning has become prevalent, these techniques use advanced methodologies and tools from a theory of experiment design to machine learning approaches, among others [27]. Several methods have been proposed for parameter tuning, for instance F-Race [28], ParamILS [15], SMAC [16], HORA [29]. However, these methods have limitations when tuning a large number of parameters or when they require significant computational resources to perform the test runs [14].

Parameter control (*online-tuning*) emerges as a reasonable option. Some strategies have been proposed for specific metaheuristics such as [30] for swarm intelligence and [31] for evolutionary algorithms. Also, some specific strategies has been proposed for the QAP, such as [32] which proposes a strategy for self-control parameters on a Tabu Search method.

Hyper-heuristics present another way to face the problem of metaheuristic parameter control. These form a novel research approach in which a high level strategy selects or generates the best metaheuristics with their respective parameters and acceptance criteria. Aiming to have more general methods, not designed for a single problem or for a few instances of a problem [33]. To the best of our knowledge, only one hyperheuristic method solves the QAP and uses parallelism in its design [34]: the authors propose a parameter control method using a genetic algorithm (GA) acting as a high-level strategy in the hyper-heuristic approach. The GA, generation by generation, performs the adaptation of parameters through cross-over and mutation operators, ending up with the parameters at their best adjustment for each method.

We achieve a form of hyperheuristic using cooperative parallelism. The key idea is to use the parallel computational power to not only create a hybrid metaheuristic but also to automatically control the parameters of the metaheuristic involved in the parallel hybrid method.

3 DPA-QAP method

This section presents the general structure of DPA-QAP, a Dynamic Parameter Adaptation method for solving the Quadratic Assignment Problem. DPA-QAP is build on the top of a parallel hybrid metaheuristic solver, similar to the one presented in [11]. Figure 1 presents the two main components of DPA-QAP, the *Worker* nodes and a *Master* node (workers and master to simplify). Workers run a set of metaheuristics, in parallel, carrying out the search process. We design each worker to run in a separate thread, ideally bound to its own dedicated core, each thread runs a specific metaheuristic instance.

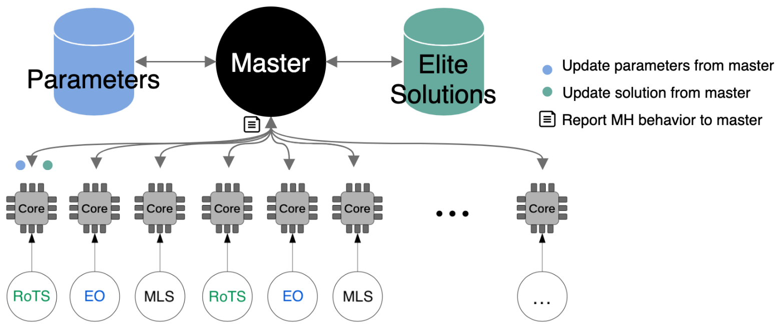
Parameters
Master
Elite Solutions
Update parameters from master
Update solution from master
Report MH behavior to master
Core
RoTS
EO
MLS
RoTS
EO
MLS
...

Fig. 1: DPA-QAP top-level view.

Each worker reports periodically its current candidate solution and some contextual information (e.g., solution cost, performance metrics, etc.) to the master, which stores best intermediate solutions into an elite pool. When the master receives a solution from a worker, it merges it into the elite pool. If the incoming solution is already present, it gets mutated by performing two random swaps. This mechanism promotes some diversity for the candidate solutions in the pool. When the elite pool is full, the master sends solutions to workers, ensuring the receiver implements a different metaheuristic from the one that inserted that solution into the pool. This process constitutes a flexible interaction feature which eases the hybridization of metaheuristics promoting cross-fertilization among different types. The size of this pool is equal to the number of workers, since the pool must have a solution for each method.

On the top of this cooperative parallel search, DPA-QAP implements a dynamic adaptation strategy which is tasked to automatically adjust the parameters of the metaheuristics during their execution, looking for the best setting and trying to ensure a balance between intensification and diversification.

3.1 Metaheuristics used in the DPA-QAP method

We select three different metaheuristic methods for the workers: Robust Tabu Search (RoTS), Extremal Optimization (EO), and Multistart Local Search (MLS). We select these metaheuristics because they are commonly used in combinatorial optimization problems particularly for the QAP. We now present a brief description of each of these methods.

Robust Tabu Search The name Tabu Search (TS) refers to the use of an adaptive memory and special problem-solving strategies, to get a better local search method [35]. The idea is to memorize within a structure the elements that for the LS will be forbidden to use (*tabu*) and thus avoid getting trapped in local optima. TS looks for the best solution within the neighborhood but does not visit the solutions of previous neighbors if they have been visited before or have been marked as prohibited locations [34]. RoTS is an adaptation of TS to the QAP and has been one of the best performing methods for this problem [18].

Extremal Optimization EO is a metaheuristic inspired by self-organizing processes as frequently found in nature: for EO this is *self-organized criticality* (SOC). The version proposed by [36] has only one adjustable parameter: τ, and uses of a Probability Distribution Function (PDF). EO proceeds like this: it inspects all candidate configurations assigning a fitness value, by means of the goal function. The configurations are then ranked from worst to best. EO resorts to the PDF to choose a solution from organized configurations. The role of the τ parameter is to provide different search strategies from pure random walk ($\tau = 0$) to deterministic (greedy) search ($\tau \Rightarrow \infty$). In previous work, we extended the basic EO metaheuristic to support not only a power-law PDF, but also an exponential and a gamma-law PDFs [26].

Multistart Local Search Local Search (LS) is one of the oldest and most frequently used metaheuristics. LS starts from an initial solution and repeatedly improves it within a defined neighborhood. Neighbor solutions can be generated by applying minor changes to the initial solution. LS ends when no improved solutions are found in the neighborhood achieving a local optima [37]. Multistart Local Search (MLS) is a modification of LS that iteratively performs multiple different searches, executing each LS from a different starting point. When MLS reaches a local optimum, it tries to escape by restarting the search from scratch or performing some random moves in the current solution.

Table 1: Metaheuristic's parameter ranges (n stands for QAP instance's size).

Metaheuristic	Parameter name	Range
RoTS	Tabu duration factor	[$4n$ - $20n$]
	Aspiration factor	[n^2 - $10n^2$]
EO	PDF	Power - Exponential - Gamma
	τ	[0,1]
MLS	Start type	Restart from scratch - Random swaps

Metaheuristics Parameters Table 1 presents the parameters considered for each metaheuristic, together with the range of variation for each parameter. These ranges are picked from the best performances, as reported in the literature. For RoTS we use the parameters reported in [18], for EO we select the parameters reported in [26] and for MLS, the only parameter used is the restart process, then no range is needed.

4 Automatic Parameter Adaption in DPA-QAP

The DPA-QAP method operates within an iterative process. At the beginning, workers are initialized with random parameters. DPA-QAP dynamically adapts the best setting of parameters in every worker (which is executing a metaheuristic instance). Parameter control depends on the performance in the solving process for an individual worker at each iteration. Each worker periodically reports relevant information to the master. With this information, the master evaluates the worker's performance and tweaks its parameters, trying to strike a balance between intensification and diversification in the search. Figure 2 depicts the flow diagram of this process. Gray boxes represent the functionality executed by workers, in parallel. White boxes specify the iterative adaptation process by the master. The master waits while the workers perform the search. When it receives a metaheuristic report, it develops a *performance evaluation* for each worker and executes the *parameters' adaptation* procedure. The master then sends a new, evolved, set of parameters and a new configuration back to the workers. Workers resume the search with the settings they received: parameters and restarting from a new initial solution (from the master's elite pool). DPA-QAP repeats this process until an established number of iterations is accomplished or when the solution target is reached.

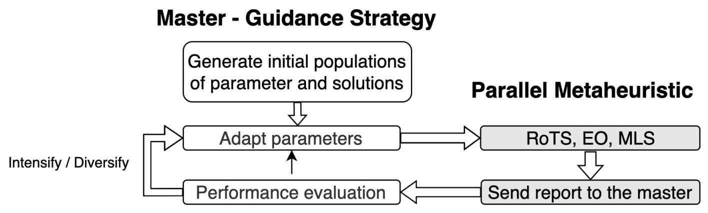
Master - Guidance Strategy
Generate initial populations of parameter and solutions
Parallel Metaheuristic
Adapt parameters
RoTS, EO, MLS
Intensify / Diversify
Performance evaluation
Send report to the master

Fig. 2: DPA-QAP flow diagram.

4.1 Metaheuristics Performance Metrics

At each iteration of the parameters' adaptation process, each metaheuristic runs for a given time, `iteration_time`. When the `iteration_time` is running out, workers report to the master the initial solution and the best found solution in the interval with theirs associated costs. In order to assess the performance of a worker using a specific set of parameters, the master computes the distance between the initial and final solution (pair-wise difference) and the percentage gain for that iteration. The percentage gain is defined as: gain = $\frac{cost_{initial} - cost_{final}}{cost_{initial}}$.

Evaluating the performance of the metaheuristics is a critical process, and selecting the right set of metrics affects the overall performance of the parameters' adaptation process. In this work we consider two classical metrics, the percentage gain in the cost of the objective function and the distance between solutions. The gain acts as a direct indicator of the metaheuristic's performance, meanwhile the distance is assessing how diverse the search is. Other metrics can be also considered, for instance, the time spend on local optima, the number of iterations without improvements, among many others.

4.2 Performance Evaluation

The parameters' adaptation process evaluates the workers' performance by processing the percentage gain and the similarity between the initial and final solution. Through experimentation we verify that the gain is usually bigger at initial stages of the search than at the final stage. For this reason, DPA-QAP changes the value of the diversification gain limit during the search process, inspired by how the temperature decreases in simulated annealing [38]. Figure 3 shows how the diversification gain limit decreases in DPA-QAP during the search process. Using this dynamic limit, DPA-QAP diversifies the search more easily at the beginning than at the end of the search process. The similarity criterion is computed comparing the distance between the initial and final solutions. If this distance is lower than one-third of the QAP size (i.e., 66% of the variables are equal), we consider both solutions as "very similar".

Considering these two criteria, we defined the following rules to determine which action must be taken for adapting the worker's parameters: If the gain obtained by the method and its pair-wise difference is lower than the corresponding limits, the component adapts the metaheuristic parameters to *diversify* the search. If the gain is higher than the corresponding diversification gain limit or the pair-wise difference is higher than the distance solution limit, the component adapts the parameters to *intensify* the search. Both the dynamic diversification limits and the distance solution limit are hyper-parameters of the auto-parametrization strategy. We plan to test different limits in future work.

Adapting the Parameters. The evaluation of the worker's performance outputs a mandate which can be, intensify or diversify. This output is used as input for the parameters adaptation process. For each possible case we define a behavior depending of the metaheuristic type.

In EO the parameter τ is in the range 0 to 1 and, depending on its value and the PDF, this may lead the metaheuristic to intensify or diversify the search, by adding or subtracting a delta value belonging to the range (see Figure 4). The parameters are then adjusted by adding to their values using deltas, so the master performs a search process that looks for the best parameters setting for a given metaheuristic.

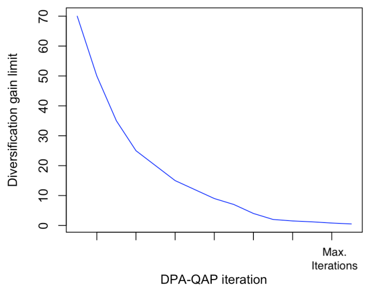

Fig. 3: Gain diversification limits.

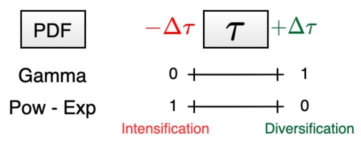

Fig. 4: EO parameters adaptation.

We define the parameter adaptation process for Robust Tabu search as follows: if the parameter adaptation component returns *diversify*, a delta of $n/2$ is added to the tabu duration and a delta of $n^2/2$ is added to the aspiration parameters. If the parameter adaptation component returns *intensify*, the tabu duration is subtracted by $n/2$ and the aspiration is decreased by $n/2$. For intensification, the delta for the adaptation of the aspiration parameter is different to diversification. This is done intending to slow down the intensification process, avoiding to stagnates on a local optimum. For the case of MLS, if there is any gain in cost, the type of restart is retained. If there is no gain, the algorithm changes to the other option.

5 Experimental Evaluation

In this section we present an experimental evaluation of our proposed method, DPA-QAP, comparing its performance against an independent parallel hybrid metaheuristic method. We consider three sets of very hard benchmarks: the 20 hardest instances of QAPLIB [39] and two sets of even harder instances:

Drezner's [40] `dre`XX and Palubeckis's [41] `Inst`XX instances. Each instance is executed 30 times stopping as soon as the Best Known Solution (BKS) is found or when a time limit of 5 minutes is hit, in case the BKS is not reached. All experiments have been carried out on a quad-AMD Opteron 6380 system, totaling 64 cores running at $2.5GHz$ and 128 GB of RAM.

At present, DPA-QAP is systematically configured with 30 worker nodes: 10 running RoTS, 10 running EO and 10 running MLS. Each worker node randomly initializes each parameter of its metaheuristic by randomly picking a value from the admissible values (see Table 1). These parameters are then periodically adapted as explained in the previous section. In this experiment, parameter control is triggered every 15 or 20 seconds, depending on the size of the problem. Each metaheuristic can thus adapt its parameters up to 20 times during the 5 minutes global execution cap. We plan to study the impact of varying this interval and determine if it is also possible and useful to dynamically adapt it.

We compare DPA-QAP to a *base solver* (BASE-QAP) which is statically parametrized (this solver is actually derived from DPA-QAP by disabling the parameter control mechanisms). Other than that, BASE-QAP is identical to DPA-QAP: it also creates 30 metaheuristic instances (10 of each type of metaheuristic); each metaheuristic instance also randomly initializes its parameters, which instead remain fixed during the execution. Our goal is to compare this pre-process parameterization (parameters fixed) with self-parameterization. Usually the parameter tuning pre-process is a time-consuming task, the idea is to avoid this offline tuning step by having an online method able to adapt its parameters meanwhile the problem solution is carry out.

Both methods are similarly implemented in Java 11 using the `ForkJoinPool` and `Atomic`***`Type`*** classes to handle the parallelism in a shared memory model [4]. In all cases we made sure that each worker node is actually mapped by the JVM onto a different physical core, at runtime.

5.1 Evaluation on QAPLIB

We evaluated the performance of the DPA-QAP on QAPLIB, a well-known collection of 134 QAP problems of various sizes and difficulties [39]. The instances are named as `name`XX where `name` corresponds to the first letters of the author and XX is the size of the problem. For each instance, QAPLIB also includes the Best Known Solution (BKS), which is sometimes the optimum. Many QAPLIB instances are easy for a parallel solver, we therefore selected the 20 hardest ones (removing all systematically solved instances). We ran both DPA-QAP and BASE-QAP under the same conditions (30 repetitions, time limit of 5 minutes).

Table 2 presents the results. For each solver, the table lists the number of times the BKS is reached across the 30 executions (**#BKS**), the Average Percentage Deviation (**APD**), which is the average of the 30 relative deviation percentages computed as follows: $100 \times \frac{Avg-BKS}{BKS}$, where Avg is the average of the 30 found costs, and finally the average execution time (**Time**). Execution

[4] Source code and instances can be found here.

times are given in seconds (as a decimal number). This time is the elapsed (wall clock) time, and includes the time to install all solver instances, solve the problem, communications and the time to detect and propagate the termination. To compare the performance of both solvers, we first compare the number of BKS found, then (in case of tie), the APDs and finally the execution times. For each benchmark, the best-performing solver row is highlighted and the discriminant field is enhanced in bold font.

DPA-QAP outperforms BASE-QAP on 14 out of 20 of the hardest QAPLIB instances, while the reverse only occurs for 6 instances. 7 instances can never been solved by any solver. Clearly, a time limit of 5 minutes is too short for those hard problems: we plan to experiment with larger time limits. The "summary" row shows that DPA-QAP obtains a better *#BKS* than BASE-QAP (192 vs. 153, a 25% increase). The average APD is also better (0.174 vs. 0.180). Notice that solutions of better quality are obtained in a slightly shorter average execution time (269.5*sec* vs. 276.7*sec*).

Notice that BASE-QAP is indeed an efficient solver for this benchmark, it implements a parallel hybridization strategy and its parameters, despite being randomly initialized, are selected within a range taken from state-of-the-art solvers which report competitive results. Still, DPA-QAP managed to outperform BASE-QAP in most instances.

Table 2: Evaluation of dynamic adaptation on 20 hardest instances of QAPLIB.

		DPA-QAP			BASE-QAP		
	BKS	#BKS	APD	Time	#BKS	APD	Time
sko72	66256	**28**	0.010	130.9	24	0.012	161.2
sko81	90998	**20**	0.012	209.6	10	0.011	242.3
sko90	115534	**9**	0.022	262.2	8	0.016	274.9
sko100a	152002	**12**	0.027	245.0	4	0.029	279.3
sko100b	153890	**20**	0.012	223.1	14	0.014	242.9
sko100c	147862	**27**	0.010	268.7	20	0.010	287.2
sko100d	149576	6	0.024	287.7	**9**	0.021	285.9
sko100e	149150	**20**	0.012	266.1	16	0.015	271.2
sko100f	149036	8	0.018	267.8	**9**	0.017	265.9
tai40a	3139370	**4**	0.082	272.7	3	0.085	290.6
tai50a	4938796	0	**0.386**	300.0	0	0.401	300.0
tai60a	7205962	0	**0.479**	300.0	0	0.519	300.0
tai80a	13499184	0	**0.689**	300.0	0	0.780	300.0
tai100a	21044752	0	**0.647**	300.0	0	0.685	300.0
tai80b	818415043	**14**	0.031	282.1	13	0.028	254.5
tai100b	185996137	5	0.084	282.9	**10**	0.077	285.3
tai150b	498896643	0	0.654	300.0	0	**0.601**	300.0
tai256c	44759294	0	0.183	300.0	0	**0.179**	300.0
tho150	8133398	0	0.095	300.0	0	**0.086**	300.0
wil100	273038	**19**	0.011	292.0	13	0.013	293.0
Summary		**192**	**0.174**	**269.5**	153	0.180	276.7

5.2 Evaluation on harder instances

We evaluated DPA-QAP on two more sets of instances, artificially crafted to be very difficult for metaheuristics: the `dre`*XX* instances proposed by Drezner [40] and the `Inst`*XX* instances by Palubeckis [41]. These problems are generated with a known optimum. For this test we used the same machine and configuration as for QAPLIB (30 cores and a time limit of 5 min with 30 repetitions).

Table 3 presents the results for Drezner's instances. We have omitted small instances which are systematically solved by both solvers in less than 15 seconds. We start with `dre42` which is solved by both solvers at each replication; even on this case DPA-QAP is much faster than BASE-QAP: 34*sec* vs. 61*sec*. In all instances, DPA-QAP outperforms BASE-QAP. As a whole, DPA-QAP reaches more BKS (60 vs. 38) and, when the optimum is not reached, solutions provided by DPA-QAP are of much better quality than BASE-QAP as shown by the APDs (23.558 vs. 32.408), and it does so in a shorter period of time.

Table 3: Evaluation on Drezner instances.

	DPA-QAP			BASE-QAP		
	#BKS	APD	Time	#BKS	APD	Time
`dre42`	30	0.0	**34**	30	0.0	61
`dre56`	**21**	14.1	213	8	21.0	259
`dre72`	**9**	27.4	265	0	34.9	300
`dre90`	0	**22.1**	300	0	28.0	300
`dre110`	0	**36.1**	300	0	52.1	300
`dre132`	0	**41.7**	300	0	58.3	300
SMRY	**60**	**23.6**	**235**	38	32.4	254

Table 4: Evaluation on Palubeckis' instances.

	DPA-QAP			BASE-QAP		
	#BKS	APD	Time	#BKS	APD	Time
`Inst40`	**29**	0.15	108	26	0.17	151
`Inst50`	**23**	0.10	199	18	0.12	238
`Inst60`	**20**	0.16	188	11	0.15	249
`Inst70`	**9**	0.12	267	3	0.16	293
`Inst80`	2	**0.18**	292	2	0.19	292
`Inst100`	0	**0.18**	300	0	0.18	300
`Inst150`	0	**0.14**	300	0	0.14	300
`Inst200`	0	**0.14**	300	0	0.14	300
SMRY	**83**	**0.15**	**244**	60	0.16	265

Table 4 presents the results for Palubeckis' instances. As in the previous case, we did not include small instances which are systematically solved by both solvers in less than 15 seconds. Here again, DPA-QAP performs better than BASE-QAP on all instances of the benchmark. As for Drezner's instances, the time limit of 5 minutes appears too short to solve large instances. However, DPA-QAP does find more BKS (83 vs. 60) and dynamic parameter adaptation makes it possible to improve the quality of solutions wrt. BASE-QAP as shown by the APDs (0.147 vs. 0.157).

6 Conclusions and future work

We have proposed a dynamic parameter adaptation scheme for parallel and hybrid solvers based on metaheuristics to solve the QAP. The basic principle of this approach is to have a master node which periodically collects the progress of each metaheuristic. This node has a global view of the overall search progress, therefore it can provide each metaheuristic with new parameter values in order to increase its effectiveness. We proposed DPA-QAP: an implementation of this architecture in Java, embedding three well-known metaheuristics: Robust Tabu Search, Extremal Optimization and Multistart Local Search. The first experiments performed on very difficult instances of QAP validate our approach by significantly improving solution quality.

We plan to extend this work in several directions. First, we will experiment on machines with more cores and with time limits greater than the 5 minutes cap which was allowed in this work. We will also try to determine the best settings for parameter reporting and adjustment: in this experiment we used a constant interval which needs to be refined. Another line of potential experiments consists in including efficient metaheuristics, such as Ant Colony Optimization [42]; or embedding population-based methods, e.g. genetic algorithms. Finally, we plan to address larger instances of the QAP as well as other difficult problems. As an outcome, we aim to design and propose a general framework for self-adaptation able to address a wide variety of combinatorial search and optimization problems.

Acknowledgements. This research was supported by the CODI project PRV19-1-01 funded by the University of Antioquia in Medellín, Colombia.

References

1. Bhati, R.K., Rasool, A.: Quadratic Assignment Problem and its Relevance to the Real World: A Survey. International Journal of Computer Applications **96** (2014) 42–47
2. Abdel-Basset, M., Manogaran, G., Rashad, H., Zaied, A.N.H.: A comprehensive review of quadratic assignment problem: variants, hybrids and applications. Journal of Ambient Intelligence and Humanized Computing **0** (2018) 1–24
3. Boussaïd, I., Lepagnot, J., Siarry, P.: A survey on optimization metaheuristics. Information Sciences **237** (2013) 82–117
4. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1** (1997) 67–82
5. Blum, C., Puchinger, J., Raidl, G.R., Roli, A.: Hybrid metaheuristics in combinatorial optimization: A Survey. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **7505 LNCS** (2011) 1–10
6. Saifullah Hussin, M.: Stochastic Local Search Algorithms for Single and Bi-objective Quadratic Assignment Problems. PhD thesis, Université de Bruxelles (2016)
7. Crainic, T.G., Toulouse, M.: Parallel Meta-heuristics. In Gendreau, M., Potvin, J.Y., eds.: Handbook of Metaheuristics. Volume 146 of International Series in Operations Research & Management Science. Springer US (2010) 497–541
8. Caniou, Y., Codognet, P., Richoux, F., Diaz, D., Abreu, S.: Large-scale parallelism for constraint-based local search: the costas array case study. Constraints **20** (2014) 1–27
9. Silva, A., Coelho, L.C., Darvish, M.: Quadratic assignment problem variants: A survey and an effective parallel memetic iterated tabu search. European Journal of Operational Research (2020)

10. Munera, D., Diaz, D., Abreu, S., Codognet, P.: A Parametric Framework for Cooperative Parallel Local Search. In: European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP). Volume 8600., Granada, Spain (2014)
11. Munera, D., Diaz, D., Abreu, S.: Hybridization as Cooperative Parallelism for the Quadratic Assignment Problem. In: 10th International Workshop, HM 2016. Volume 9668 of Lecture Notes in Computer Science., Plymouth, UK, Springer International Publishing (2016) 47–61
12. López, J., Múnera, D., Diaz, D., Abreu, S.: Weaving of Metaheuristics with Cooperative Parallelism. In Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, D., eds.: Parallel Problem Solving from Nature – PPSN XV. Volume 11101 LNCS., Cham, Springer International Publishing (2018) 436–448
13. Lopez, J., Munera, D., Diaz, D., Abreu, S.: On integrating population-based metaheuristics with cooperative parallelism. In: Proceedings - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018. (2018)
14. Huang, C., Li, Y., Yao, X.: A Survey of Automatic Parameter Tuning Methods for Metaheuristics. IEEE Transactions on Evolutionary Computation **24** (2020) 201–216
15. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: an Automatic Algorithm Configuration Framework. Journal of Artificial Intelligence Research **36** (2009) 267–306
16. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **6683 LNCS** (2011) 507–523
17. Koopmans, T.C., Beckmann, M.: Assignment Problems and the Location of Economic Activities. Econometrica **25** (1957) 53–76
18. Taillard, E.: Robust taboo search for the quadratic assignment problem. Parallel Computing **17** (1991) 443–455
19. Tate, D.M., Smith, A.E.: A genetic approach to the quadratic assignment problem. Computers & Operations Research **22** (1995) 73–83
20. Moscato, P., Cotta, C., Mendes, A.: Memetic Algorithms. In: Handbook of Heuristics. Volume 1-2. Springer Berlin / Heidelberg (2004) 53–85
21. Cotta, C., Talbi, E.G., Alba, E.: Parallel Hybrid Metaheuristics. In: Parallel Metaheuristics. John Wiley & Sons, Inc., Hoboken, NJ, USA (2005) 347–370
22. Codognet, P., Munera, D., Diaz, D., Abreu, S.: Parallel local search (2018)
23. Talbi, E.G., Bachelet, V.: COSEARCH: A parallel cooperative metaheuristic. Journal of Mathematical Modelling and Algorithms **5** (2006) 5–22
24. Loukil, L., Mehdi, M., Melab, N., Talbi, E.G., Bouvry, P.: A parallel hybrid genetic algorithm-simulated annealing for solving Q3AP on computational grid. IPDPS 2009 - Proceedings of the 2009 IEEE International Parallel and Distributed Processing Symposium (2009)
25. Munera, D., Diaz, D., Abreu, S., Rossi, F., Saraswat, V., Codognet, P.: Solving Hard Stable Matching Problems via Local Search and Cooperative Parallelization. In: AAAI, Austin, TX, USA (2015)
26. Munera, D., Diaz, D., Abreu, S.: Solving the Quadratic Assignment Problem with Cooperative Parallel Extremal Optimization. In: The 16th European Conference on Evolutionary Computation in Combinatorial Optimisation, Porto (2016)

27. Hoos, H.: Automated Algorithm Configuration and Parameter Tuning. In Hamadi, Y., Monfroy, E., Saubion, F., eds.: Autonomous Search. Springer Berlin Heidelberg, Berlin (2012) 37–71
28. Birattari, M., Kacprzyk, J.: Tuning metaheuristics: a machine learning perspective. Volume 197. Springer (2009)
29. Barbosa, E.B., Senne, E.L.: A heuristic for optimization of metaheuristics by means of statistical methods. ICORES 2017 - Proceedings of the 6th International Conference on Operations Research and Enterprise Systems **2017-January** (2017) 203–210
30. Parpinelli, R.S., Plichoski, G.F., Silva, R.S.D., Narloch, P.H.: A review of techniques for online control of parameters in swarm intelligence and evolutionary computation algorithms. International Journal of Bio-Inspired Computation **13** (2019) 1–20
31. Karafotias, G., Hoogendoorn, M., Eiben, Á.E.: Parameter control in evolutionary algorithms: Trends and challenges. IEEE Transactions on Evolutionary Computation **19** (2014) 167–187
32. Fescioglu-Unver, N., Kokar, M.M.: Self Controlling Tabu Search algorithm for the Quadratic Assignment Problem. Computers & Industrial Engineering **60** (2011) 310–319
33. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R.: A Classification of Hyper-Heuristic Approaches: Revisited. In: International Series in Operations Research and Management Science. Springer Berlin Heidelberg (2019) 453–477
34. Dokeroglu, T., Cosar, A.: A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem. Engineering Applications of Artificial Intelligence **52** (2016) 10–25
35. Glover, F.: Tabu search—part II. ORSA Journal on Computing **2** (1990) 4–32
36. Boettcher, S., Percus, A.: Nature's way of optimizing. Artificial Intelligence **119** (2000) 275–286
37. Yagiura, M., Ibaraki, T.: Local Search (2002)
38. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220 4598** (1983) 671–80
39. Burkard, R.E., Karisch, S., Rendl, F.: QAPLIB - a Quadratic Assignment Problem Library. European Journal of Operational Research **55** (1991) 115–119
40. Drezner, Z.: The Extended Concentric Tabu for the Quadratic Assignment Problem. European Journal of Operational Research **160** (2005) 416–422
41. Palubeckis, G.: An Algorithm for Construction of Test Cases for the Quadratic Assignment Problem. Informatica, Lith. Acad. Sci. **11** (2000) 281–296
42. Hani, Y., Amodeo, L., Yalaoui, F., Chen, H.: Ant colony optimization for solving an industrial layout problem. European Journal of Operational Research **183** (2007) 633–642