Hybridization as Cooperative Parallelism for the Quadratic Assignment Problem

Danny Munera[1], Daniel Diaz[1] and Salvador Abreu[2,1]

[1] University of Paris 1-Sorbonne/CRI, France
danny.munera@malix.univ-paris1.fr, daniel.diaz@univ-paris1.fr
[2] Universidade de Évora/LISP, Portugal
spa@di.uevora.pt

Abstract. The Quadratic Assignment Problem is at the core of several real-life applications. Finding an optimal assignment is computationally very difficult, for many useful instances. The best results are obtained with hybrid heuristics, which result in complex solvers. We propose an alternate solution where hybridization is obtain by means of parallelism and cooperation between simple single-heuristic solvers. We present experimental evidence that this approach is very efficient and can effectively solve a wide variety of hard problems, often surpassing state-of-the-art systems.

Keywords: QAP, heuristics, parallelism, cooperation, hybridization, portfolio

1 Introduction

The Quadratic Assignment Problem (QAP) was introduced in 1957 by Koopmans and Beckmann [1] as a model of a facilities location problem. This problem consists in assigning a set of n facilities to a set of n specific locations minimizing the cost associated with the *flows* of items among facilities and the *distance* between them. This combinatorial optimization problem has many other real-life applications: scheduling, electronic chipset layout and wiring, process communications, turbine runner balancing, data center network topology, to cite but a few [2,3]. This problem is known to be NP-hard and finding effective algorithms to solve it has attracted a lot of research in recent years. To tackle problems of medium or large size ($n > 30$) one must resort to incomplete methods which are designed to quickly provide good, albeit potentially sub-optimal, solutions. This is the case of *metaheuristics*. Since the mid-1980s several metaheuristics have been successfully applied to the QAP: tabu search, simulated annealing, genetic algorithms, GRASP, ant-colonies [3]. For solving the hardest instances, the current trend is to specialize existing heuristics [4,5] often by *combining* different metaheuristics (*hybrid procedures*) [6,7] and/or to resort on parallelism [8,9].

We recently proposed a sequential Extremal Optimization (EO) procedure for QAP which performs well on the QAPLIB instances [10]. We developed a cooperative parallel version of this method, a process which was eased thanks to

our Cooperative Parallel Local Search (CPLS) framework [11,12] for which we developed an implementation in the X10 programming language [13,14]. This solver (called ParEO) behaves very well on the set of 33 hardest and largest instances of QAPLIB. Using 128 cores and within a short time limit of 5 minutes, ParEO is able to find the best known solution (BKS) in each replication for 15 problems. Only for 8 instances is the BKS never reached. Recent research shows that the most promising way to improve QAP resolution is to resort to hybrid procedures, in order to benefit from the strengths of different classes of heuristics. Such is the case of hybrid genetic algorithms (a.k.a memetic algorithms) [7]. The price to pay for this improvement is a significant increase in the complexity of the resulting solver code. In any case, many of the best known existing methods can be easily parallelized thanks to our CPLS framework.

In this paper we propose an alternative approach for hybridization: we resort to cooperation and parallelism to get "the best of both worlds". To this end, the parallel instances of different heuristics communicate their best solutions during execution, and are able to forgo the current computation and *adopt* a better solution (hoping it will converge faster). The expected behavior is that a solution which appears to be stagnating inside one solver can be improved by another heuristic. When the second solver can no longer improve on this (imported) solution, maybe the original one can, once again, improve the solution yet a bit more, and so on. It is worth noticing that when the first solver sends its current solution, it continues to work on it until it adopts an external solution, itself. This *cooperative portfolio* approach behaves like a hybrid solver while retaining the original simplicity of each solver. This is particularly true inside the CPLS framework since solvers need not be aware about the (nature of) other solvers.

We implemented such a hybrid solver on top of the X10 version of CPLS, combining two different solvers: Taillard's robust tabu search (RoTS) [15] and our EO-QAP [10] method. We have chosen these two solvers because they are simple and also because it turned out that they present complementary strengths: roughly speaking RoTS is stronger in intensifying the search in a given region while EO-QAP is better at widely diversifying the search. The resulting hybrid cooperative solver (called ParEOTS) displays very good performance, as we shall see further. We show that it scales very well, exhibiting a linear speedup when increasing the number of cores. This solver behaves much better than the cooperative versions of both EO-QAP and RoTS alone. On the 33 hardest instances of QAPLIB, using 128 cores and a time limit of 5 minutes, ParEOTS is able to find the BKS for 26 problems at each replication. Even for the 7 other problems, the quality of returned solutions (measured as a percentage of average solution over the BKS), is significantly improved. We also test ParEOTS on Palubeckis' `Inst`XX instances and on Drezners `dre`XX instances. Moreover we provide optimal solutions for several `Inst`XX instances and for `dre90`, `dre100` and `dre132`.

The rest of the paper is organized as follows: section 2 discusses QAP, RoTS and EO-QAP. Section 3 presents our parallel hybrid solver. Several experimental results are laid out and discussed in section 4 and we conclude in Section 5.

2 Background

In this section we recall some background topics: the Quadratic Assignment Problem (QAP) and the two heuristics we plan to combine: RoTS and EO-QAP.

2.1 QAP

Since its introduction in 1957, QAP has been widely studied and several surveys are available [16,2,17,3]. A QAP problem of size n consists of two $n \times n$ matrices (a_{ij}) and (b_{ij}). Solving such a problem consists in finding a permutation π of $\{1, 2, \ldots n\}$, minimizing the objective function: $F(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot b_{\pi_i \pi_j}$. In facility location problems, the a matrix represents inter-facility flows and b encodes the inter-location distances. Moreover, QAP can be also used to model scheduling, chip placement and wiring on a circuit board, to design typewriter keyboards, for process communications, for turbine runner balancing among many other applications [2,18].

QAP is computationally very difficult: it is a discrete problem, the objective function contains products of variables and the theoretical search space of an instance of size n has a size $n!$. QAP has been proved to be NP-hard [19] (the traveling salesman problem can be formulated as a QAP) and there is no ϵ-approximation algorithm for QAP (unless P=NP). In practice, this means that QAP is one of the toughest combinatorial optimization problems, and one with several real-life applications.

QAP can be (optimally) solved with exact methods like dynamic programming, cutting plane techniques and branch & bound algorithms for medium sizes, e.g. $n \leq 30$. For larger problems, (meta)heuristics are the most efficient tool. Over the last decades several metaheuristics were successfully applied to QAP: tabu search, simulated annealing, genetic algorithms, GRASP, ant-colonies [20].

2.2 RoTS : a Tabu Search Procedure for QAP

Tabu search as proposed by Glover [21] has been widely used since the 1990 to tackle QAP. Unquestionably, one of the most important algorithms for QAP is Taillard's robust tabu search [15] (RoTS). This algorithm uses an adaptive short-term memory for the tabu list by recording the value assigned to a variable for a while (in order to prevent "reverse assignments"). It also uses a clever *aspiration criterion* (needed to authorize a tabu move to be performed in special circumstances, e.g. if it improves on the best solution found so far). RoTS also incorporates a long-term memory to ensure a form of diversification, by encouraging moves towards to not yet visited regions. RoTS only requires two user parameters to be tuned: the *tabu tenure factor* (controlling the time an element remains tabu) and the *aspiration factor* both of which influence the adaptive memory ([22] provides good references values for these parameters). In practice RoTS is tremendously effective on a wide variety of QAP instances, being able to quickly find high quality solutions. Several BKS for QAPLIB instances have been discovered and/or improved by RoTS. A key feature explaining its speed is

that the cost of a solution resulting from a swap can be computed incrementally and further optimized using a tabling mechanism. This results in an evaluation in $O(n^2)$, while the naïve algorithm is in $O(n^3)$. In addition, Taillard put the source code in the public domain. All these reasons explain the fact that RoTS is directly or indirectly at the root of many other methods to solve QAP [4,23].

2.3 EO-QAP : an Extremal Optimization procedure for QAP

Extremal Optimization (EO) is a metaheuristics inspired by self-organizing processes often found in nature [24,25,26]. EO is based on the concept of *Self-Organized Criticality* (SOC) initially proposed by Bak and on the Bak-Sneppen's model [27]. In this model of biological evolution, *species* have a *fitness* $\in [0,1]$ (0 representing the worst degree of adaptation). At each iteration, the species with the worst fitness is eliminated (or forced to mutate). This change affects its fitness but also the fitness of all other species connected to this "culprit" element. This results in an *extremal* process which progressively eliminates the least fit species (or forces them to mutate). Repeating this process eventually leads to a state where all species have a good fitness value, i.e. a SOC. EO follows this line: it inspects the current *configuration* (assignment of variables), selects one of the *worst* variables (according to their fitness) to *mutate*. For this, it ranks the variables in increasing order of fitness (the worst variable has thus a rank $k = 1$) and then resorts to a *Probability Distribution Function* (PDF) over the ranks k to chose the culprit element. This PDF introduces uncertainty in the search process. The original EO proposes a power-law: $P(k) = k^{-\tau} \qquad (1 \leq k \leq n)$. This PDF takes a single parameter τ which is problem-dependent. Depending on the value of τ, EO provides different search strategies from pure random walk ($\tau = 0$) to deterministic (greedy) search ($\tau \to \infty$). With an adequate value for τ, EO cannot be trapped in local minima since any variable is susceptible to mutate (even if the worst are privileged). This parameter can be tuned by the user (a default value is $\tau = 1 + \frac{1}{ln(n)}$).

EO displays several a priori advantages: it is a simple metaheuristic, it is controlled by only one free parameter (a fine tuning of several parameters becomes quickly tedious) and it does not need to be aware about local minima. Nevertheless, EO has been successfully applied to large-scale optimization problems like graph bi-partitioning, graph coloring or the traveling salesman problem [25].

Recently, we proposed EO-QAP: an EO procedure for QAP [10]. One notable extension we brought to the original EO is to propose different PDFs and to allow the user to chose the most adequate one for a given problem. The sequential procedure performs well on the whole set of QAPLIB instances: 68 instances are *solved* (i.e. the BKS could be reached) at each execution, 41 are solved at least once and 25 never. The independent parallel version improves the situation significantly: 33 additional instances are systematically solved, 14 are partially solved and 19 remain unsolved. To tackle this remaining set of 33 instances (14+19) we experimented with cooperative parallelism (this version is called ParEO). In the same time limit, ParEO is able to systematically solve 15 new instances and 18 are solved at least once (8 remain unsolved).

3 A Cooperative Parallel Hybrid Method

We propose an alternative approach for constructing hybrid search methods, resorting on our Cooperative Parallel Local Search Framework (CPLS) [12,11], to provide the hybridization. In a nutshell, the procedure amounts to having several workers, each following its own strategy, some of which are significantly different from others. The cooperative framework oversees every worker, and makes it possible for it to contribute and benefit from the global effort, by managing a pool of best solution candidates (the *elite pool*). The fact that the framework is parallel entitles it to obtain performance benefits by just increasing the count of compute units (cores.) Moreover, the workers themselves need to have little or no knowledge of the environment they are running under.

To test these ideas, we experimented with a solver for QAP – an admittedly difficult problem – for which the individual metaheuristic we chose are our EO-QAP algorithm and the RoTS method.

3.1 Cooperative Parallel Local Search

Parallel local search methods have been proposed in the past [28,29,30]. Here we focus on *multi-walk* methods (also called *multi-start*) which consist in a concurrent exploration of the search space, either *independently* or *cooperatively*, the latter being achieved with communication between processes. The *Independent Multi-Walks* method (IW) [31] is easiest to implement since the solver instances need not communicate with each other. However, the resulting gain tends to flatten when scaling beyond about a hundred processors [32], largely because the inherent diversity which brings about the speedups is not sufficient. In the *Cooperative Multi-Walks* (CW) method [33], the solver instances exchange information (through communication), hoping to hasten the search process. However, the design and implementation of an efficient such method is a very challenging task: choices abound concerning the communication which impact each other, many of which are problem-dependent [33].

We designed the Cooperative Parallel Local Search (CPLS) framework [12,11]. This framework, available as an open source library in the X10 programming language, allows the programmer to tune the search process through an extensive set of parameters which, at present, statically condition the execution. CPLS augments the IW strategy with a tunable communication mechanism, which allows for the cooperation between the multiple solver instances to seek either an intensification or diversification strategy in the search. At present, the tuning process is done manually: we have not yet experimented with parameter self-adaptation in the CPLS framework (still an experimental feature).

The basic component of CPLS is the *explorer node* which consists in a local search-based solver instance. The point is to use all the available processing units by mapping each *explorer node* to a physical core. Explorer nodes are grouped into *teams*, of size NPT (see Figure 1). This parameter is directly related to the trade-off between intensification and diversification. NPT can take values from 1 to the maximum number of cores. When NPT is equal to 1, the framework

coincides with the IW strategy, it is expected that each 1-node team be working on a different region of the search space, without any effort to seek parallel intensification. When NPT is equal to the maximum number of nodes (creating only 1 team in the execution), the framework is mainly geared towards parallel intensification (however a certain amount of diversification is inherently provided by parallelism, between 2 cooperation actions).

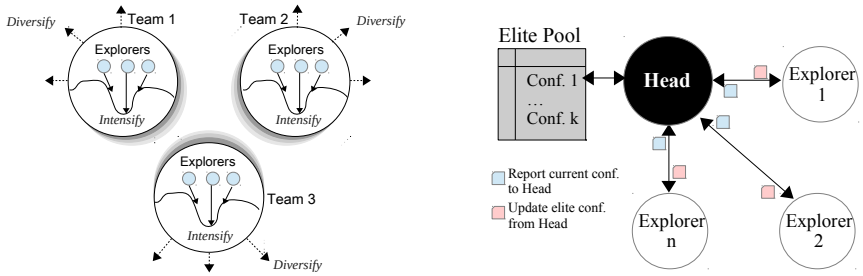


Fig. 1: CPLS framework structure

Each team seeks to *intensify* the search in the most promising neighborhood found by any of its members. The parameters which guide the intensification are the *Report Interval* (R) and *Update Interval* (U): every R iterations, each explorer node sends its current configuration and the associated cost to its *head node* (*report event*). The head node is the team member which collects and processes this information, retaining the best configurations in an *Elite Pool* (EP) whose size $|EP|$ is parametric. Every U iterations, explorer nodes randomly retrieve a configuration from the EP, in the head node (*update event*). An explorer node may *adopt* the configuration from the EP, if it is "better" than its own current configuration, with a probability *pAdopt*. Simultaneously, the teams implement a mechanism to cooperatively *diversify* the search, i.e. they try to extend the search to different regions of the search space.

Typically, each problem benefits from intensification and diversification to some extent. Therefore, the tuning process of the CPLS parameters seeks to provide an appropriate balance between the use of the intensification and diversification mechanisms, in hope of reaching better performance than the non-cooperative parallel solvers (i.e. independent multi-walks). A detailed description of this framework may be found in [11].

3.2 Using the CPLS Framework for Hybridization

The current X10 implementation of the CPLS framework already supports the use of multiple metaheuristics. Adding a new one is simple because CPLS provides useful abstraction layers and handles communication. Adding a new metaheuristic comes down to slightly adapt the sequential algorithm: every R iterations it has to send its current configuration to the Elite Pool and, every U

iterations, it needs to retrieve a configuration from the pool, which it may subsequently adopt (with probability *pAdopt*), should it be better than the current one. The overall resulting solver is thus composed of several instances of the *same* metaheuristic running in parallel, which cooperate by communicating in order to faster converge to a solution. To date, CPLS includes cooperative parallel versions of three different methods: *Adaptive Search*, *Extremal Optimization* and *Tabu Search*. In the present work, we go one step beyond and propose a new usage of the CPLS framework in order to obtain an hybrid parallel solver. For this, individual workers run instances of *different* metaheuristics, while still collaborating by communicating with the head node. The basic idea of running different metaheuristics in parallel exchanging elite solutions has been proposed [28,34] but only from a general and theoretical point of view. This can also be viewed as a *portfolio* approach [35] augmented with cooperation.

We chose to experiment with this form of hybridization on QAP combining two metaheuristics: our EO-QAP procedure and the RoTS method, resulting in a solver we call ParEOTS. The communication strategies of CPLS remain unchanged, ensuring cooperation between the explorers which now happen to be running different methods. Figure 2 presents possible interactions due to cooperation and the implementation of the hybrid strategy. The team's EP will now contain configurations stemming from explorers running different heuristics.

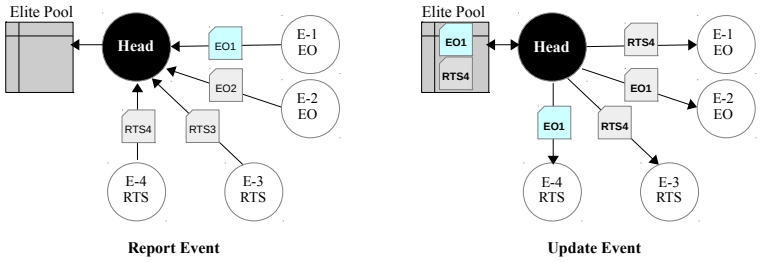


Fig. 2: Hybridization in CPLS : combining EO-QAP and RoTS

Here is a possible scenario: inside the same team, an instance E_1 of EO-QAP reports a good configuration C_1 to the EP. Later, an instance R_1 of RoTS retrieves C_1, improves on it (RoTS being strong at intensification) and obtains a better configuration C_2, on which it reports back to the EP. Later, C_2 gets adopted by an instance E_2 which, being in a diversification phase, moves to a faraway search region, which may provide yet better solutions. Obviously, other scenarios are possible, e.g. when another EO-QAP explorer E_3 also retrieves C_1 (provided by EO) it gives a "second chance" to this configuration (due to its internal stochastic state it can further improve this configuration). The whole system behaves as a hybrid solver, benefiting from cross-fertilization due to the inherent diversity of the search strategies.

4 Experimental Evaluation

In this section we present an experimental evaluation of our hybrid parallel method (source code, instances and new solutions will be soon available from `http://cri-hpc1.univ-paris1.fr/qap/`). All experiments have been carried out on a cluster of 16 machines, each with 4 × 16-core AMD Opteron 6376 CPUs running at 2.3 GHz and 128 GB of RAM. The nodes are interconnected with InfiniBand FDR 4× (i.e. 56 GBPS). We had access to 4 nodes and used up to 32 cores per node, i.e. 128 cores. Each problem is executed 10 times stopping as soon as the BKS (which is sometimes the optimum) is found. This execution is done with a short time limit of 5 minutes (in case the BKS is not reached). Such experiments give an interesting information about the quality of solutions quickly obtainable. All times are given either in seconds for small values (as a decimal number) or in a human readable form as mm:ss or hh:mm:ss). The relevant CPLS parameters controlling the cooperation are (as per [11]):

- *Team Size* (NPT): we fixed it to $NPT = 16$. There are thus 8 teams composed of 16 explorer nodes ; 8 running a EO-QAP solver and 8 running RoTS solver. This is constant over all problems. We did not yet experiment with other splits.
- *Report and Update Interval* (R and U): we manually tuned U and usually fix $R = U/2$.
- *Elite Pool* (EP): its size is fixed to 4 for all problems.
- *pAdopt*: is set to 1. Any solver instance receiving a better configuration than its current one always switches to the new one.

4.1 Scalability Analysis

We start this experimental evaluation by analyzing the scalability of ParEOTS. Such an analysis is not easy, because if the BKS cannot be reached, the runtime is only bounded by the timeout used. It is thus necessary to only consider problems that can be systematically solved by the EO sequential solver (to have a reference time using 1 core). We selected two instances of QAPLIB which require the longest sequential time: `tai35a` solved on average in $42.399\,s$ and `lipa70a` solved in $57.737\,s$. We then ran these problems with ParEOTS, varying the number of cores from 2 to 128. Figure 3 presents the speedup data and curves obtained with our algorithm (using a log-log scale). The *Ideal* curve corresponds to linear speedup: time is halved when the number of cores is doubled. For both problems the speedup is linear. Using 128 cores, the best speedup is 126, obtained for `tai35a` whose execution time now only requires $0.336\,s$.

4.2 Evaluation on QAPLIB

We here evaluate the performance of our hybrid solver ParEOTS on a set of 33 hard instances of QAPLIB. We selected this set because it is the most difficult set for the independent parallel version of our EO procedure [10]. In addition to

Cores	tai35a		lipa70a	
	time	speedup	time	speedup
1	0:42	1.0	0:57	1.0
2	0:33	1.3	0:18	3.2
4	0:20	2.1	0:17	3.4
8	8.9	4.8	8.4	6.9
16	6.3	6.8	3.8	15.2
32	2.6	16.6	2.1	27.5
64	1.4	31.4	1.1	54.3
128	0.3	126.0	0.5	106.3

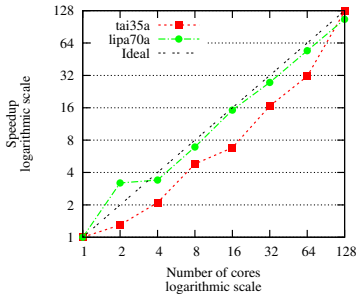
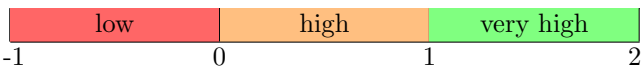


Fig. 3: Speedup profile using the Hybrid CPLS on two QAPLIB instances

raw performance, and for validation purpose, we also want to assess the gain obtained with the hybrid version compared the cooperative parallel versions of its two components: ParEO and ParRoTS (also written in X10 within CPLS). For this, all 3 systems are run under the same conditions (see Section 4). Obviously, ParEO runs 128 instances of our EO procedure, ParRoTS runs 128 instances of RoTS while ParEOTS executes 64 instances of EO-QAP and 64 of RoTS. To measure the *hybrid performance* we focus on the number of BKS found by each parallel solver. When running 50% of EO and 50% of RoTS we define a *low threshold* (*low*) as the average of #BKS found by both parallel solvers. This corresponds to what can be normally expected. Below this value, the hybrid solver is ineffective. Above, it already performs well. Moreover, we define a *high threshold* (*high*) as the maximum of the #BKS of both solvers. Above this value, the hybrid solver performs at least as well as the best single solver (a hybrid solver without gain would need twice the number of cores to obtain such a performance). Obviously *low* and *high* can be generalized to an hybridization involving more than 2 solvers. For a given problem, executed n times, the performance (*hperf*) of the hybrid solver reaching $\#bks$ times the BKS is defined as follows:

$$hperf = \begin{cases} \dfrac{\#bks - low}{low}, & \text{if } \#bks < low \\ \dfrac{\#bks - low}{high - low}, & \text{if } low < \#bks < high \text{ and } low \neq high \\ 1 + \dfrac{\#bks - high}{n - high}, & \text{if } high \leq \#bks \text{ and } n \neq high \\ 1, & \text{if } high = \#bks = n \end{cases} \quad (1)$$

The performance ranges in $[-1, 2]$. if $hperf < 0$ the hybrid solver is ineffective on that problem. For $hperf \in [0, 1)$ the performance is acceptable and when $hperf \in [1, 2]$ the performance is very good.



	ParEOTS				*hperf*	ParEO				ParRoTS			
	$\#_{bks}$	APD	time	$\#_{ad.}$		$\#_{bks}$	APD	time	$\#_{ad.}$	$\#_{bks}$	APD	time	$\#_{ad.}$
els19	**10**	0.000	0.0	2.6	1.00	**10**	0.000	0.0	0.2	**10**	0.000	0.0	0.3
kra30a	**10**	0.000	0.0	3.9	1.00	**10**	0.000	0.0	2.6	**10**	0.000	0.0	3.6
sko56	**10**	0.000	1.5	0.3	1.00	**10**	0.000	4.8	2.5	**10**	0.000	0.6	0.0
sko64	**10**	0.000	1.7	0.3	1.00	**10**	0.000	4.8	1.5	**10**	0.000	1.3	0.0
sko72	**10**	0.000	8.7	1.0	1.00	**10**	0.000	0:13	1.4	**10**	0.000	0:16	1.7
sko81	**10**	0.000	0:24	1.8	1.00	7	0.008	1:58	9.4	**10**	0.000	1:06	4.6
sko90	**10**	0.000	1:32	4.8	1.00	**10**	0.000	1:32	5.0	7	0.002	1:54	5.3
sko100a	**10**	0.000	1:09	1.3	2.00	5	0.012	3:44	4.2	7	0.002	2:46	3.3
sko100b	**10**	0.000	0:45	0.8	1.00	8	0.001	2:26	2.6	**10**	0.000	1:02	0.6
sko100c	**10**	0.000	0:56	1.0	1.00	**10**	0.000	2:25	2.4	6	0.001	3:12	3.6
sko100d	**10**	0.000	1:03	1.1	1.00	6	0.014	3:20	3.6	**10**	0.000	0:37	0.2
sko100e	**10**	0.000	0:47	0.9	1.00	**10**	0.000	1:43	1.6	5	0.002	2:47	3.0
sko100f	**10**	0.000	0:57	0.9	2.00	4	0.011	4:05	4.8	5	0.003	3:42	4.3
tai40a	**10**	0.000	1:26	1.6	1.00	7	0.022	2:51	3.4	**10**	0.000	1:04	1.0
tai50a	3	0.077	4:24	3.5	−0.33	**5**	0.026	3:28	2.4	4	0.044	4:11	2.5
tai60a	**3**	0.146	4:15	0.9	1.13	2	0.132	4:45	1.9	0	0.297	5:00	2.0
tai80a	0	0.364	5:00	4.9	1.00	0	0.385	5:00	1.0	0	0.605	5:00	1.0
tai100a	0	0.298	5:00	2.0	1.00	0	0.297	5:00	3.0	0	0.567	5:00	5.0
tai20b	**10**	0.000	0.0	1.0	1.00	**10**	0.000	0.0	0.8	**10**	0.000	0.0	0.3
tai25b	**10**	0.000	0.0	0.5	1.00	**10**	0.000	0.6	17.0	**10**	0.000	0.0	0.0
tai30b	**10**	0.000	0.1	1.9	1.00	**10**	0.000	0.1	3.0	**10**	0.000	0.1	1.2
tai35b	**10**	0.000	0.3	4.3	1.00	**10**	0.000	0.7	14.2	**10**	0.000	0.2	1.9
tai40b	**10**	0.000	0.1	0.6	1.00	**10**	0.000	0.1	0.4	**10**	0.000	0.2	2.0
tai50b	**10**	0.000	2.6	1.2	1.00	2	0.214	4:26	4.5	**10**	0.000	2.1	0.0
tai60b	**10**	0.000	4.6	1.2	1.00	3	0.205	4:16	2.6	**10**	0.000	5.3	0.0
tai80b	**10**	0.000	0:53	1.6	2.00	0	1.192	5:00	8.8	5	0.002	3:06	6.0
tai100b	**10**	0.000	1:11	0.7	2.00	0	0.465	5:00	5.5	2	0.035	4:10	4.8
tai150b	0	0.061	5:00	0.7	1.00	0	1.088	5:00	1.5	0	0.103	5:00	0.3
tai64c	**10**	0.000	0.0	0.0	1.00	**10**	0.000	0.0	0.3	**10**	0.000	0.0	0.0
tai256c	0	0.178	5:00	2.2	1.00	0	0.263	5:00	1.3	0	0.266	5:00	1.5
tho40	**10**	0.000	0.5	0.0	1.00	**10**	0.000	1.2	0.2	**10**	0.000	0.4	0.0
tho150	**1**	0.007	4:51	2.0	1.10	0	0.144	5:00	1.7	0	0.019	5:00	1.9
wil100	**10**	0.000	1:37	1.9	2.00	0	0.061	5:00	5.4	6	0.001	2:16	2.4
Summary	**267**	**0.034**	**1:24**	**1.6**	**1.12**	199	0.138	2:28	3.7	227	0.059	1:53	1.9

Table 1: ParEOTS on QAPLIB and comparison with ParEO and ParRoTS

Table 1 presents the results. The parameters used for EO are the same as in [10]. For RoTS we generally use a tabu tenure = $8n$ and an aspiration = $4n^2$. The table reports, for each solver, the number of times out of 10 runs the BKS was reached ($\#bks$), the Average Percentage Deviation (APD) which is relative deviation percentage computed as follows: $100 \times \frac{Avg-BKS}{BKS}$ (where Avg is the average of the 10 found costs), the average execution time (average of the 10 wall times for one instance) and the numbers of adoptions done by the winning explorer (#ad.). The performance value is also reported. The last row presents the averages of each column (or sums for $\#bks$ columns).

It is worth noticing that the overall performance of the cooperative parallel version of the 2 base solvers using a short time limit is rather good. Even so, the hybrid solver clearly outperforms them. Focusing on #BKS, it provides high performance ($hperf \geq 1$) for 32 instances (only for `tai50a` does it behave worse than its two components). Moreover, in 4 cases it obtains a $hperf = 2$ corresponding to cases where it performs much better than both individual solvers (to such an extent that it obtains the perfect score #BKS = 10). It found the

BKS at each replication for 26 problems; this is much better than ParEO (15) and ParRoTS (18). In only 4 cases, could ParEOTS not reach the BKS: this number is 8 for ParEO and 6 for ParRoTS. It is worth noticing that even in these 4 cases, the hybridization is still effective since the APD is lower than for its components. For instance, on the very difficult problem `tai256c`, the hybridization cannot solve the problem but the APD is 0.178 while it is around 0.263 for both components. Another remarkable case is `tho150`, for which the hybridization is very effective. The average APD is now 0.007 (0.144 for ParEO and 0.019 for ParRoTS). In fact, it turns out that this problem could even be solved once.

The "summary" row reports interesting numbers. All in one, the average APD of ParEOTS is 0.034 which is much better than 0.138 for ParEO and 0.059 for ParRoTS. Regarding execution times, it is a good surprise to see that the increase of quality does not hamper the speed. In fact, with an average execution time of $85s$ the hybrid solver is faster than ParEO ($148s$) and ParRoTS ($113s$).

4.3 Testing on Palubeckis' instances

In 2000, Palubeckis proposed a new hard problem generator with known optimum [36] and provided a set of 10 hard instances called `Inst`XX. Few results have been published about experiments with them. Palubeckis reports the best solutions found by a repeated local search procedure (called multi-start descent or MSD). In [37] the authors propose an Ant Colony Optimization algorithm (QAP-ACO) and test it on these instances (in this work these instances are called `palu`XX).

We experimented in the same setting as previously: with 128 cores and a time limit of 5 minutes. Table 2 displays the results for 3 solvers. In addition to the APD we also provide the *best* cost value found among the 10 runs. Data is taken from the aforementioned articles. We also provide execution times for QAP-ACO for indicative purposes.

	opt.	ParEOTS				QAP-ACO				MSD		
		$\#_{bks}$	APD	best value	time	$\#_{bks}$	APD	best value	time	$\#_{bks}$	APD	best value
Inst20	81536	**10**	0.000	81536	0.0	0	0.340	81817	1.2	**10**	0.000	81536
Inst30	271092	**10**	0.000	**271092**	0.1	0	0.580	272654	0:10	0	0.364	272080
Inst40	837900	**10**	0.000	**837900**	4.0	0	0.360	840930	1:02	0	0.287	840308
Inst50	1840356	**10**	0.000	**1840356**	0:17	0	0.380	1847422	3:46	0	0.354	1846876
Inst60	2967464	**10**	0.000	**2967464**	1:07	0	0.390	2978898	10:05	0	0.362	2978216
Inst70	5815290	**10**	0.000	**5815290**	2:07	0	0.300	5832460	24:24	0	0.287	5831954
Inst80	6597966	**10**	0.000	**6597966**	1:56	0	0.310	6618736	50:42	0	0.308	6618290
Inst100	15008994	**1**	0.120	**15008994**	5:00	0	0.270	15048806	1:41:02	0	0.256	15047406
Inst150	58352664	0	0.126	**58414888**	5:00					0	0.198	58468204
Inst200	75405684	0	0.125	**75498892**	5:00					0	0.183	75543960

Table 2: ParEOTS on Palubeckis' instances (128 cores, timeout $5m$)

Even with a limit of 5 minutes, the performances of ParEOTS are very good. The optimum is reached for problems whose size $n \leq 100$. In addition, for all

$n \leq 80$ ParEOTS reaches the optimum at each replication. For sizes $n > 100$, clearly a limit of 5 minutes is too short to reach the optimum. Nevertheless, the obtained solutions are of good quality with an APD around 1.12%: 2-3 times better than challengers. It is worth noticing that for $n > 20$ all published best obtained solutions are improved (in bold font in the table). Regarding execution times, ParEOTS also outperforms its competitors.

4.4 Testing on Drezner's instances

In 2005, Drezner and al. designed new QAP instances with known optimum but specifically ill conditioned to be difficult for metaheuristic methods [38]. The authors reports the best solutions found by a powerful compounded hybrid genetic algorithm (called CHG in what follows). The instances are really difficult and only very recently were some results published by Acan and Ünveren with a *great deluge* algorithm (called TMSGD) [39]. These hard instances are thus an interesting challenge for our hybrid solver.

We ran it under the same conditions as before: using 128 cores and with a time limit of 5 seconds. Table 3 presents the results for 3 solvers. Data is taken from the above mentioned articles (in the case of CHG each problem was executed 20 times, presented #BKS are divided by 2 for normalization). We also provide execution times for TMSGD for indicative purposes (TMSGD was run on a 2.1 GHz PC).

	opt.	ParEOTS				TMSGD				CHG		
		$\#_{bks}$	APD	best	time	$\#_{bks}$	APD	best	time	$\#_{bks}$	APD	best
dre15	306	**10**	0.000	306	0.0	**10**	0.000	306	2.1			
dre18	332	**10**	0.000	332	0.0	**10**	0.000	332	7.4			
dre21	356	**10**	0.000	356	0.0	**10**	0.000	356	0:18			
dre24	396	**10**	0.000	396	0.0	**10**	0.000	396	0:56			
dre28	476	**10**	0.000	476	0.1	**10**	0.000	476	1:18			
dre30	508	**10**	0.000	508	0.1	**10**	0.000	508	2:36	**10**	0.00	508
dre42	764	**10**	0.000	764	0.7	6	0.25	764	8:51	9	1.34	764
dre56	1086	**10**	0.000	1086	5.6	3	3.556	1086	18:39	3	17.46	1086
dre72	1452	**10**	0.000	1452	0:26	0	8.388	1512	47:06	1	27.28	1452
dre90	1838	**9**	0.968	**1838**	2:47	0	10.979	1959	1:36:33	0	33.88	2218
dre110	2264	**6**	6.334	**2264**	3:43	0	15.123	2479	2:41:25			
dre132	2744	**1**	22.784	**2744**	4:54	0	17.553	3023	3:31:07			

Table 3: ParEOTS on Drezner's instances (128 cores, timeout $5\,m$)

The performance of ParEOTS is very good: all problems could be optimally solved, and, to the best of our knowledge, this is the first time that an optimal solution is found for `dre90`, `dre110` and `dre132`. TMSGD performs better than CHG (but the CHG experiment is old). Regarding execution times, ParEOTS needs 2:47 to solve `dre90` while TMSGD cannot solve it even using 1:36:33 (CHG reports one hour for `dre90` and also fails to find the optimum).

5 Conclusion and Further Work

We set out to construct a hybridized solver by resorting to a parallel and cooperative multi-walk scheme, which relies on the CPLS framework to provide both the cooperation and the parallel or distributed execution.

As a testbed for the idea, we chose to tackle the Quadratic Assignment Problem, because it is recognized as a very difficult problem of significant practical interest and also because benchmark instances abound in the literature. For this we designed ParEOTS: a hybrid cooperative parallel solver combining two methods: our Extremal Optimization algorithm and Taillard's robust tabu search. This hybrid solver is much more efficient than any of its two individual base solvers. Regarding QAPLIB, our hybrid solver is able to reach the best known solution (BKS) for all instances except 4. In most cases it is even able to systematically find the BKS at each replication. Even then, for the 4 not fully solved hardest instances (`tai80a`, `tai100a`, `tai150b` and `tai256c`), the solutions obtained are very close to the BKS. We also tested the solver on other hard instances. The results on Palubeckis' instances are very good: for the first time, ParEOTS optimally solved all instances up to a size $n = 100$ (prior to this work only optimal solutions for $n = 20$ were known). We discovered optimal solutions for sizes $n = 30..100$ and 2 new best obtained solutions for $n = 150$ and $n = 200$. Regarding Drezner's instances, the results are even better: we discovered optimal solutions for all instances (including `dre90`, `dre110` and `dre132`). This is the first time that optimal solutions for these 3 instances are published.

From our experiments, it became clear that: (1) the coding effort for building a hybrid solver is much lower with our approach than for existing hybrid algorithms, and (2) the performance gain over competing approaches is very significant. The latter aspect can be construed as a sort of evolutionary algorithm, one which blends phenotypes rather than genotypes, all under the supervision of the cooperative framework. As to the former, the changes needed to fit the CPLS scheme are minimal and very simple.

We plan to further explore portfolio approaches, combining more than two types of solver as well as experimenting with techniques for parameter autotuning. Another line entails the induction of solver multiplicity by presenting several instances of the same solver, but set up with different parameters.

Acknowledgments

The authors wish to thank Prof. E. Taillard for providing the RoTS source code and explanations. The experimentation used the cluster of the University of Évora, which was partly funded by grants ALENT-07-0262-FEDER-001872 and ALENT-07-0262-FEDER-001876.

References

1. Koopmans, T.C., Beckmann, M.: Assignment Problems and the Location of Economic Activities. Econometrica **25**(1) (1957) 53–76

2. Commander, C.W.: A survey of the quadratic assignment problem, with applications. Morehead Electronic Journal of Applicable Mathematics **4** (2005) MATH–2005–01
3. Bhati, R.K., Rasool, A.: Quadratic Assignment Problem and its Relevance to the Real World: A Survey. International Journal of Computer Applications **96**(9) (2014) 42–47
4. James, T., Rego, C., Glover, F.: Multistart Tabu Search and Diversification Strategies for the Quadratic Assignment Problem. IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans **39**(3) (2009) 579–596
5. Benlic, U., Hao, J.K.: Breakout Local Search for the Quadratic Assignment Problem. Applied Mathematics and Computation **219**(9) (2013) 4800–4815
6. Drezner, Z.: The Extended Concentric Tabu for the Quadratic Assignment Problem. European Journal of Operational Research **160**(2) (2005) 416–422
7. Drezner, Z.: Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. Computers & Operations Research **35**(3) (2008) 717–736
8. James, T., Rego, C., Glover, F.: A Cooperative Parallel Tabu Search Algorithm for the Quadratic Assignment Problem. European Journal of Operational Research (2009)
9. Tosun, U.: On the Performance of Parallel Hybrid Algorithms for the Solution of the Quadratic Assignment Problem. Engineering Applications of Artificial Intelligence **39** (2015) 267–278
10. Munera, D., Diaz, D., Abreu, S.: Solving the Quadratic Assignment Problem with Cooperative Parallel Extremal Optimization. In: European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP). Lecture Notes in Computer Science, Springer (2016)
11. Munera, D., Diaz, D., Abreu, S., Codognet, P.: A Parametric Framework for Cooperative Parallel Local Search. In Blum, C., Ochoa, G., eds.: European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP). Volume 8600 of Lecture Notes in Computer Science., Springer (2014) 13–24
12. Munera, D., Diaz, D., Abreu, S., Codognet, P.: Flexible Cooperation in Parallel Local Search. In: Symposium on Applied Computing (SAC), New York, New York, USA, ACM Press (2014) 1360–1361
13. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., Von Praun, C., Sarkar, V.: X10: An Object-Oriented Approach to Non-Uniform Cluster Computing. In: SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, San Diego, CA, USA, ACM (2005) 519–538
14. Saraswat, V., Tardieu, O., Grove, D., Cunningham, D., Takeuchi, M., Herta, B.: A Brief Introduction to X10 (for the High Performance Programmer). Technical report (2012)
15. Taillard, É.D.: Robust Taboo Search for the Quadratic Assignment Problem. Parallel computing **17**(4-5) (1991) 443–455
16. Burkard, R.E.: Quadratic Assignment Problems. In Pardalos, P.M., Du, D.Z., Graham, R.L., eds.: Handbook of Combinatorial Optimization (2nd edition). Springer New York (2013) 2741–2814
17. Loiola, E.M., de Abreu, N.M.M., Netto, P.O.B., Hahn, P., Querido, T.M.: A survey for the quadratic assignment problem. European Journal of Operational Research **176**(2) (2007) 657–690
18. Zaied, A.N.H., Shawky, L.A.E.f.: A Survey of Quadratic Assignment Problems. International Journal of Computer Applications **101**(6) (2014) 28–36

19. Sahni, S., Gonzalez, T.: P-Complete Approximation Problems. Journal of the ACM **23**(3) (1976) 555–565
20. Said, G.A.E.N.A., Mahmoud, A.M., El-Horbaty, E.S.M.: A Comparative Study of Meta-heuristic Algorithms for Solving Quadratic Assignment Problem. International Journal of Advanced Computer Science and Applications (IJACSA) **5**(1) (2014)
21. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers (jul 1997)
22. Taillard, É.D.: Comparison of iterative searches for the quadratic assignment problem. Location Science **3**(2) (1995) 87–105
23. Misevicius, A.: A Tabu Search Algorithm for the Quadratic Assignment Problem. Computational Optimization and Applications **30**(1) (jan 2005) 95–111
24. Boettcher, S., Percus, A.: Nature's way of optimizing. Artificial Intelligence **119**(12) (2000) 275–286
25. Boettcher, S., Percus, A.G.: Extremal Optimization: an Evolutionary Local-Search Algorithm. In: Computational Modeling and Problem Solving in the Networked World. Volume 21. Springer US (2003)
26. Boettcher, S.: Extremal Optimization. In Hartmann, A.K., Rieger, H., eds.: New Optimization Algorithms to Physics. Wiley-VCH Verlag, Berlin (2004) 227–251
27. Bak, P., Sneppen, K.: Punctuated equilibrium and criticality in a simple model of evolution. Physical Review Letters **71**(24) (1993) 4083–4086
28. Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. Wiley-Interscience (2005)
29. Alba, E., Luque, G., Nesmachnow, S.: Parallel Metaheuristics: Recent Advances and New Trends. International Transactions in Operational Research **20**(1) (2013) 1–48
30. Diaz, D., Abreu, S., Codognet, P.: Parallel Constraint-Based Local Search on the Cell/BE Multicore Architecture. In: Studies in Computational Intelligence. Volume 315. (2010) 265–274
31. Verhoeven, M., Aarts, E.: Parallel Local Search. Journal of Heuristics **1**(1) (1995) 43–65
32. 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**(1) (2014) 1–27
33. Toulouse, M., Crainic, T., Sansó, B.: Systemic Behavior of Cooperative Search Algorithms. Parallel Computing (2004) 57–79
34. Talukdar, S., Baerentzen, L., Gove, A., De Souza, P.: Asynchronous Teams: Cooperation Schemes for Autonomous Agents. Journal of Heuristics **4**(4) 295–321
35. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence **126**(1-2) (2001) 43–62
36. Palubeckis, G.: An Algorithm for Construction of Test Cases for the Quadratic Assignment Problem. Informatica, Lith. Acad. Sci. **11**(3) (2000) 281–296
37. Wu, K.C., Ting, C.J., Gonzalez, L.C.: An Ant Colony Optimization Algorithm for Quadratic Assignment Problem. In: Asia-Pacific Conference on Industrial Engineering and Management Systems. (2011)
38. Drezner, Z., Hahn, P., Taillard, É.: Recent Advances for the Quadratic Assignment Problem with Special Emphasis on Instances that are Difficult for Meta-Heuristic Methods. Annals of Operations Research **139**(1) (2005) 65–94
39. Acan, A., Ünveren, A.: A Great Deluge and Tabu Search Hybrid with Two-stage Memory Support for Quadratic Assignment Problem. Applied Soft Computing **36**(C) (nov 2015) 185–203