On Integrating Population-Based Metaheuristics with Cooperative Parallelism

Jheisson Lopez, Danny Munera, Daniel Diaz, Salvador Abreu

On Integrating Population-Based Metaheuristics with Cooperative Parallelism

Jheisson Lopez
University of Antioquia/UNGS
Colombia/Argentina
jalopez@ungs.edu.ar

Danny Munera
University of Antioquia
Colombia
danny.munera@udea.edu.co

Daniel Diaz
University of Paris 1/CRI
France
daniel.diaz@univ-paris1.fr

Salvador Abreu
University of Évora/LISP
Portugal
spa@uevora.pt

***Abstract*—Many real-life applications can be formulated as Combinatorial Optimization Problems, the solution of which is often challenging due to their intrinsic difficulty. At present, the most effective methods to address the hardest problems entail the hybridization of metaheuristics and cooperative parallelism. Recently, a framework called CPLS has been proposed, which eases the cooperative parallelization of local search solvers. Being able to run different heuristics in parallel, CPLS has opened a new way to hybridize metaheuristics, thanks to its cooperative parallelism mechanism. However, CPLS is mainly designed for local search methods. In this paper we seek to overcome the current CPLS limitation, extending it to enable population-based metaheuristics in the hybridization process. We discuss an initial prototype implementation for Quadratic Assignment Problem combining a Genetic Algorithm with two local search procedures. Our experiments on hard instances of QAP show that this hybrid solver performs competitively w.r.t. dedicated QAP parallel solvers.**

***Index Terms*—parallelism, cooperation, metaheuristics, genetic algorithm, hybridization, QAP**

I. Introduction

Combinatorial Optimization Problems (COP) are widely used to model and solve real-life problems in several application domains, such as shortest/cheapest round trips, scheduling, planning, time-tabling, resource allocation, network configuration and monitoring, matching, business product line modeling, system management... Solving these problems rapidly represents a real challenge due to their inherent difficulty (most of them are NP-hard). The study of COPs as well as the search for efficient algorithms to solve them, has therefore been a very active research area for many years. Due to the computational intractability of most problems, we can find (or generate) problem instances of varying degrees of difficulty.

Easy and medium-sized problems can be solved with *exact methods*, which guarantee the optimality of the computed solution or prove that a problem does not admit any solution. This is possible because these methods consider the entire search space: either explicitly by exhaustive search or implicitly, by detecting when a portion of the search space can be ignored and get pruned off.

For harder problems, *approximation methods* are more appropriate. These methods aim at finding a solution of "good" quality (possibly sub-optimal) in a reasonable amount of time. Metaheuristics have proven to be the most efficient approximation methods to address this class of problems. Metaheuristics are high-level procedures using choices (i.e., heuristics) to limit the part of the search space which actually gets visited, in order to make problems tractable. Metaheuristics can be classified in two main categories: *single-solution* and *population-based* methods. Single-solution metaheuristics maintain, modify and stepwise improve on a single candidate solution (also called a *configuration*). Examples of single-solutions methods include: Simulated Annealing [1], Local Search [2], Tabu Search [3], Variable Neighborhood Search [4], Adaptive Search [5], Extremal Optimization [6]... On the other hand, population-based methods, modify and improve a *population*, i.e. sets of candidate solutions. These methods include: Genetic Algorithms [7], Ant Colony Optimization [8], Particle Swarm Optimization [9]... Metaheuristics generally implement two main search strategies: *intensification* and *diversification*, also called exploitation and exploration [10]. Intensification guides the solver to deeply explore a promising part of the search space. In contrast, diversification aims at extending the search onto different parts of the search space [11]. In order to obtain the best performance, a metaheuristic should provide a useful balance between intensification and diversification. By design, some heuristics are better at intensifying the search while others are so at diversifying it. More generally, each metaheuristic has it own strengths and weaknesses, which may greatly vary according to the problem or even instance being solved.

For yet more difficult problems, the current trend is therefore to design *hybrid* metaheuristics, by combining different metaheuristics in order to benefit from the individual advantages of each method. An effective approach consists in combining a population-based method with a single-solution method (often a local search procedure). These hybrid methods are called *memetic algorithms* [12]. However, hybrid metaheuristics are complex procedures, tricky to design, implement and tune.

An orthogonal approach to address very difficult problems consists in using parallel computation. For instance, several instances of a given metaheuristic can be executed in parallel in order to develop concurrent explorations of the search space, either *independently* or *cooperatively* by means of communication between concurrent processes. The independent

approach is easiest to implement on parallel computers, since no communication is needed between the processes running a metaheuristics. The whole execution simply stops as soon as any process finds a solution. For some problems this approach provides very good results but in many cases the speedup tends to taper off when increasing the number of processors. A cooperative approach entails adding a communication mechanism in order to share or exchange information among solver instances during the search process. However, designing an efficient cooperative method is a dauntingly complex task [13], and many issues must be solved: *What information is exchanged? Between which processes is it exchanged? When is the information exchanged? How is it exchanged? How is the imported data used?* [14]. Moreover, most cooperative choices are problem-dependent (and sometimes even instance-dependent). Bad choices result in poor performance, possibly much worse than what could be obtained with independent parallelism. However, a well-tuned cooperation scheme may significantly outperform the independent approach. To this end, a framework – called CPLS – which eases the cooperative parallelization of local search solvers has been recently proposed in [15], [16]. This framework, available as an open source library written in the IBM X10 concurrent programming language, allows the programmer to tune the search process through an extensive set of parameters. This framework has been successfully used to tackle stable matching problems [17] and very difficult instances of the Quadratic Assignment Problem (QAP) [18].

More interestingly, as it is able to run *different* heuristics in parallel, CPLS has opened a new way to hybridize metaheuristics, by exploiting its solution-sharing cooperative parallelism mechanism. The user only needs to code (in X10) each of the desired metaheuristics, independently, and may rely on CPLS to provide both parallelism and cooperation to get "the best of both worlds". At runtime, the parallel instances of each different metaheuristic communicate their best solutions, and one of them may forgo its current computation and *adopt* a better solution from the others, hoping it will converge faster. The expected outcome is that a solution which may be stagnating for one solver, has a chance to be improved on by another metaheuristic. CPLS has been successfully used to develop a very efficient hybrid solver for QAP, called ParEOTS, by implicitly combining Extremal Optimization and Tabu Search [19].

In this work we are interested in hybridizing single-solution metaheuristics with population-based metaheuristics and in particular in memetic algorithms. Theoretically it is possible to plug a population-based metaheuristics into CPLS, we thus tried to hybridize a genetic algorithm with the local search procedures used in ParEOTS (extremal optimization and tabu search) [19]. It turned out that the resulting solver performed worse than ParEOTS! An explanation for this observation is that the CPLS basic cooperation mechanisms are designed for single-solution metaheuristics, and also that CPLS is better at intensification than diversification. ParEOTS performed well because extremal optimization can be conveniently tuned to provide an appropriate level of diversification. Even though some single-solution metaheuristics are rather good at diversification, when well tuned, population-based metaheuristics offers a wider variety of diversification thanks to the large possibilities of evolution offered by a population. However, to be effective in an hybrid procedure, this population should encompass many solutions (e.g. all solution candidates computed by all solver instances).

In this paper we seek to overcome the current CPLS limitation, extending it to enable population-based metaheuristics in the hybridization process. We experimented with an initial prototype for QAP, which we show to already perform competitively w.r.t. ParEOTS and other state-of-the-art parallel solvers.

The rest of this paper is organized as follows: in section II, we describe the existing CPLS and discuss some of its limitations. In section III we present the adaptation of CPLS used in our prototype implementation in order to hybridize a genetic algorithm with two local search procedures to solve QAP. Section IV provides an initial experimental assessment of our prototype. The paper ends with some considerations for future directions.

II. The Cooperative Parallel Local Search framework

The Cooperative Parallel Local Search framework (CPLS) [15], [16] was designed to facilitate the implementation of cooperative parallel local search solvers. CPLS is available as an open source library written in the IBM X10 concurrent programming language. From a user point of view, one needs only code the desired metaheuristic, in X10, using the CPLS Application Programming Interface (API). This API provides object-oriented abstractions to hide all parallel management details from the user, who writes a purely sequential program, without having to worry about parallelism and communication. The resulting solver can be executed on a wide variety of machine (multicores, manycores, clusters,...), implicitly using the parallelism provided by the X10 programming system. Furthermore, CPLS augments the independent parallelism strategy with a tunable communication mechanism, which enables cooperation between the multiple metaheuristic instances. It also offers several parameters which control this mechanism. To activate cooperative parallelism, the user simply annotates the existing code with calls to CPLS functions to periodically execute 2 actions[1]:

- report on its current configuration (*Report* action).
- retrieve a solution[2] and, if it is good enough, the solver *adopts* it, i.e. it abandons its current solution and replaces it with the received one (*Update* action).

At runtime, each local search instance is encapsulated in an *explorer node*. The point is to use all available processing

[1]These calls must be inserted inside the main loop of the metaheuristic.

[2]The received configuration is most likely reported by another local search instance.

units by mapping each explorer node to a physical core. CPLS organizes the explorers into *teams*, where a team consists of `NPT` explorers which aim at intensifying the search in a particular region of the search space using *intra-team communication*. `NPT` can range from 1 to the maximum number of cores. This parameter is directly related to the trade-off between intensification and diversification since it is expected that different teams explore different regions of the search space. When `NPT` is 1, the framework coincides with the independent parallelism 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 hits the maximum level of intensification (note that a certain amount of diversification is inherently provided by parallelism, between 2 cooperation actions, due to the stochastic nature of metaheuristics.) Tuning the value of `NPT` allows the user to adjust the trade-off between intensification and diversification.

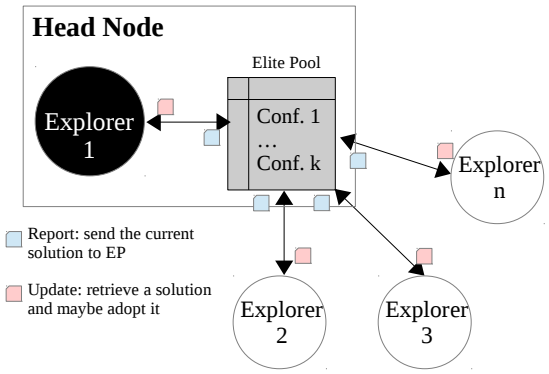

Fig. 1: CPLS team structure

By design, a 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 metric to its *head node*. The head node is a particular explorer inside the team which periodically collects and processes this information, retaining the best configurations in the *Elite Pool* `EP` of size `|EP|`. Every `U` iterations, explorer nodes randomly retrieve a configuration from `EP`, in the head node. An explorer node may *adopt* the configuration from the `EP`, if it is "better" than its own current configuration with a probability `pAdopt`. Figure 1 depicts the structure of a team.

In addition to the notion of teams of parametric size, CPLS provides an additional mechanism for diversification through *inter-team communications*. For this, a particular node, called the *master node*, periodically compares each team's best configuration (taken from its elite pool) using a distance function (e.g., using a Hamming distance). If two or more teams are too close, the master node selects a team to perform a corrective action (this can be the "worst" team, i.e. the one whose best configuration cost is highest). The corrective action – which is indeed a diversification action – aims at forcing all explorers (or a fraction thereof) in the selected team, to explore a different area of the search space.

It is worth noting that CPLS abstracts the notion of *metaheuristic* and thus accepts any metaheuristic, including others besides local search. Several configurations are possible; the most common one consists in executing in parallel several instances of exactly the same metaheuristic (clones). One may also execute the same metaheuristic but configured differently, i.e. with different parameter tuning. For instance, the same tabu search procedure could be executed with different tabu tenure values. A portfolio approach can be obtained with the independent parallel execution of different metaheuristics. Finally, it is also possible to execute different metaheuristics, within the cooperative scheme just described. The resulting solver behaves like a hybrid solver, just that it's achieved without the complexity required by the design and coding of the hybrid solver.

ParEOTS [19] is a powerful hybrid solver for QAP combining Extremal Optimization and Tabu Search. Being interested in memetic algorithms, we tried to add a Genetic Algorithm (GA) as third component to this hybrid solver. The integration was simple but we the resulting solver was not any better than ParEOTS. We sought the CPLS characteristics which prevent the efficient use of population-based metaheuristics for hybridization. It turns out that the CPLS basic cooperation mechanisms are designed for single-solution metaheuristics; with the Elite Pool, CPLS is intrinsically better at intensification than diversification. The diversification mechanism described above is very difficult to tune (we could never obtain any gain from it). Additionally, ParEOTS does not use it. This mechanism has two major drawbacks:

1) It is a *centralized process*: the master node has the big responsibility of making the right decision (it risks breaking a winning team). It acts with a very limited view of the computation: basically the best solutions computed so far by a team. In a sense, diversification is not sufficiently well informed to be effective when the likely next moves are taken from larger sets, as happens with population-based metaheuristics such as GA.
2) CPLS is biased towards execution on distributed architectures based on clusters of multicore computers: teams are mapped to nodes and explorers run on individual cores. In this setting, inter-team communications tend to be relatively expensive because they require node-to-node communication.[3] It can be argued that diversification actions are costly and should therefore be used with parsimony.

ParEOTS performs well because extremal optimization can be conveniently tuned to provide an appropriate level of diversification. Even though some single-solution metaheuristics are rather good at diversification, population-based metaheuristics

[3]This aspect tends to be mitigated by high-throughput and low-latency network interconnects, such as Infiniband.

generally offer a wider spectrum of diversification, when well tuned, because of the large evolutionary possibilities afforded by a population. However, to be effective in a hybrid procedure, the population should encompass many solutions (e.g. all solutions computed inside a team). This was not the case with our previous GA experiment: the GA was treated like any other metaheuristic, it has its own private data structures and only sends and receives *single solutions* from time to time. This drove us to conjecture that the notion of population should be promoted so as to share a global vision of what happens across the team, thereby contributing to a better diversification decision. This is the idea underlying our development, which we now flesh out.

III. Hybridizing with GA

In order to provide an efficient interaction, within the CPLS framework, between population-based and local search metaheuristics, we conducted several experiments seeking to enhance the intra-team communication strategy. We tested several architectures, parameters and components, and finally we developed a prototype, dedicated to solving QAP, that actually improves on the performance of the original CPLS.

Our proposal, as shown in Figure 2, considers the inclusion of a *genetic algorithm* (GA) in the head node interacting with three explorer nodes: two implementing the robust tabu search (RoTS) method and one implementing the extremal optimization (EO) method. These methods are detailed in [18], [19]. We extend the CPLS model by having two independent pools of configurations inside the head node: the *diversification pool* (DivPool) and the *intensification pool* (IntPool).

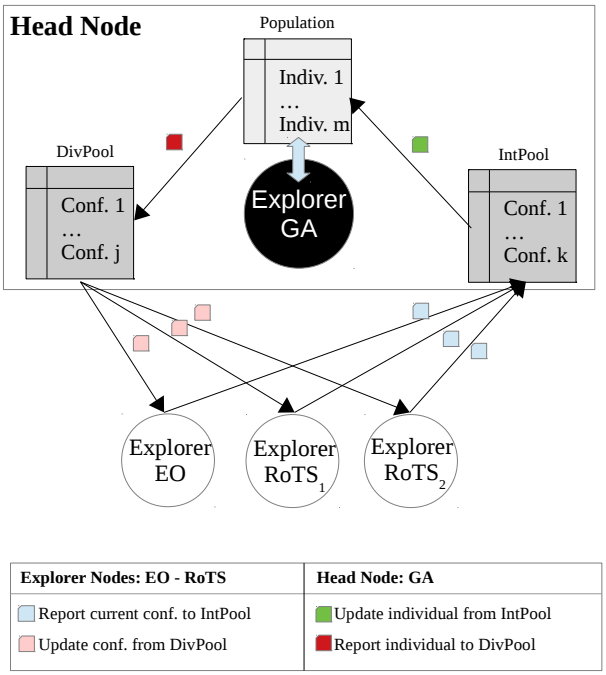

Fig. 2: CPLS new team structure

LS nodes start with a random configuration and the head node with a random *population* composed of *individuals* (i.e. configurations). They all interact with the pools as follows:

1) *LS Explorers to IntPool*: every R_S iterations, explorer nodes implementing local search methods (EO and RoTS) report their current configuration and corresponding cost to the IntPool.
2) *IntPool to Population*: every U_P iterations, the GA replaces a random individual in its population with a random configuration from the IntPool.
3) *Population to DivPool*: every R_P iterations, the GA reports an individual to the DivPool.
4) *DivPool to LS Explorer*: every U_S iterations, the explorers nodes implementing LS methods randomly retrieve a configuration from the DivPool and *may* adopt this configuration, if it is "better" than its current configuration.

During the execution of the solving process, the head node's IntPool collects elite configurations from LS explorer nodes. Then, the GA generates offspring configurations (possibly stemming from some those directly generated by the LS metaheuristics) and feeds the DivPool with some individuals. LS explorer nodes consume configurations from the DivPool and possibly adopt them, closing the interaction loop inside the team. Note that this process emulates the basic idea of memetic algorithms, while keeping the main functionality of each metaheuristic unaltered and providing hybridization through cooperative parallelism.

Finally, we propose a mechanism to ensure entropy in the GA's population, mainly because GA has the role of a "diversification agent" inside the team. If the GA's population becomes uniform (not diverse), it could mean that enough intensification has been done on this region, and it would be more interesting to explore other regions of the search space. The diversity of the population may be measured every *diversity check interval* `dCkI` with a *diversity check function* `dCkF`. If the `dCkF` value is lower than a *minimum allowed diversity value* `mdV`, then a percentage of the population (`pPr`) gets restarted. Nevertheless, a `1 - pPr` portion of the population is kept as memory of the search process, and it only gets replaced if the explorations in the new region give rise to better configurations.

A. Parameter selection

Implementing these changes requires finding suitable values for a new set of parameters. We now proceed with discussing the design and implementation decisions which we followed to select particular values for these parameters.

In our proposal, IntPool and DivPool behavior is controlled by the CPLS pool parameters, e.g., *pool size*, *entry policy* and *request policy* [15], [16]. We use the CPLS default values for both pools' parameters (the size of the pool is 4 and stores the best solutions; on demand a random solution is picked form the pool). This turns out to be convenient for our model, as we already ensure that configurations reported by a given metaheuristic only compete with other configurations reported by metaheuristics of the same nature.

To simplify the values selection for the report/update parameters, we use global R and U values, as in the original CPLS. Based on these global values, we computed the individual values for R_P, U_P, R_S and U_S, as follows: $R_p = R/NPT$ and $U_p = U/NPT$; whereas in the explorer nodes, $R_s = R/(problem\_size)$ and $U_s = U$.

We decided to use the normalized entropy, proposed in [20], to measure the diversity of the GA's population in the range from 0 to 1. An entropy value equal to 0 indicates a population consisting of multiple repetitions of the same individual, while an entropy value equal to 1 specify a population composed of individuals that are all different from each other.

B. The Genetic Algorithm Design

Genetic algorithms (GA) are inspired by Darwin's law of natural selection. The basic idea is that by combining different bits of the best adapted individuals you can get even better individuals. Hybridization of GAs is one of the most successful strategies that have been proposed to solve QAP [21]. For this reason, we have designed a GA and implemented it in CPLS, to test the new intra-team communication scheme that we described above. On one hand, a genetic algorithm basically consists of five stages that are repeated cyclically (each of these cycles is named generation): initialization of its population, selection of individuals for crossing, crossing, mutation and selection of the population for the next generation. On the other hand, a GA must define a way to encode individuals as well as two main parameters: the population size and the mutation rate. We now discuss the decisions we made for the stages and the parameters mentioned above. The pseudo-code for our GA is presented as Algorithm 1.

Algorithm 1 GA for hybridization

```
Input: population size pz, mutation rate mr, time limit timeout
  #swaps for mutation operator mS = mr * pz/2
  Random initialization of the population p
  p is ordered according to cost
  while BKS not reached and timeout not reached do
      Select an elite individual
      Select a non-elite individual
      Apply UX over previously selected individuals
      Mutate each offspring
      Sort offspring by cost
      for each offspring do
          if offspring.cost ≤ p.worstCost and offspring ∉ p then
              add offspring to p
          end if
      end for
  end while
```

We encode the individuals as permutations, as this allows for direct evaluation of the cost (it does not require a decoding process) and therefore takes less processing time. Besides, as suggested by Ahuja, et al. [22], the GA is largely independent from the initialization mechanism for its population, as long as sufficiently diverse populations are generated. For this reason, we chose a random initialization of the population as it produces individuals uniformly distributed throughout the search space.

We decided to use a population size (pz) equal to the instance size. As the size of the population determines the size of the search space region to be explored by each team, it makes sense to size it proportionately to the instance. Moreover, experimentation shows that populations greater than the instance size have a negative impact on the performance of the algorithm: the population must be sorted and the offspring should be compared with it in each generation. In a probatory experiment, we observed that population sizes less than the instance size n have a very low initial entropy, while population sizes greater that n have an initial entropy barely greater than one of size n.

Regarding the selection for crossing, in [22] the authors conclude that the use of a mechanism that favors the best individuals (i.e. an elitist one) does not necessarily lead to better results, and can even negatively affect them by hastening the convergence towards local optima. In order to establish an intermediate point of elitism, we adopt the strategy proposed by Lalla-Ruiz et al. [23]: the total population is divided into a group of elite individuals and another of non-elite ones; an element from each group is randomly selected and they are used to carry out the crossing process. The individuals generated through this process are called the offspring. We have used Uniform Crossover (UX) because, according to Benlic and Hao [24], it offers the best results for QAP. The crossing process generates two offspring in each generation.

Classically, for QAP, the mutation operator consists in swapping two of the individual's positions (2-exchanges are widely used for permutation problems). The mutation rate is a parameter which determines the probability of applying the mutation operator on the offspring. However, in our prototype, the mutation rate (mr) is the percentage of individuals that will be mutated (taking into account that a swap modifies 2 individuals). In this setting, we use a mutation rate around 40%. This value is very high for classical genetic algorithms but in our contect, the genetic algorithm is mainly used to ensure a high degree of diversification for other explorers.

Finally, we decided to use an elitist criterion in the evaluation of the offspring for deciding whether they are to be included in the population for the next generation: an offpring individual replaces the worst individual of the population if it has better cost and is not already present in the population.

IV. EXPERIMENTATION

We performed an experimental evaluation of our prototype implementation[4]. All experiments have been carried out on server with 4 × 16-core AMD Opteron 6376 CPUs, running at 2.3 GHz and 128 GB of RAM. This is the same machine as used for ParEOTS evaluation in [19] but we could only use 32 cores (while the reference paper presents results for 128 cores).

For this evaluation we selected the 33 hardest instances of the QAPLIB benchmarks highlighted in [18], [19]. Each

[4]The source code and QAP instances are available from https://github.com/jlopezrf/COPSolver-V_2.0

problem is executed 10 times, stopping as soon as the Best Known Solution (BKS) is reached. This execution is done with a time cap of 5 minutes (in case the BKS is not reached). Such experiments give useful information regarding the quality of quickly obtainable solutions.

	GA-CPLS 32 cores				ParEOTS 32 cores			
	#BKS	APD	BPD	time	#BKS	APD	BPD	time
els19	10	0.000	0.000	0.0	10	0.000	0.000	0.0
kra30a	**10**	0.000	0.000	0.0	8	0.268	0.000	60.1
sko56	10	0.000	0.000	**3.2**	10	0.000	0.000	3.5
sko64	10	0.000	0.000	**2.9**	10	0.000	0.000	5.2
sko72	10	0.000	0.000	**7.3**	10	0.000	0.000	13.7
sko81	10	0.000	0.000	56.8	10	0.000	0.000	**48.9**
sko90	**10**	0.000	0.000	14.3	5	0.003	0.000	225.5
sko100a	**10**	0.000	0.000	43.7	8	0.003	0.000	163.6
sko100b	10	0.000	0.000	**36.8**	10	0.000	0.000	64.9
sko100c	10	0.000	0.000	**57.4**	10	0.000	0.000	98.9
sko100d	10	0.000	0.000	**39.3**	10	0.000	0.000	92.0
sko100e	10	0.000	0.000	**39.1**	10	0.000	0.000	54.3
sko100f	9	0.001	0.000	**83.1**	9	0.001	0.000	133.2
tai40a	**7**	0.022	0.000	184.1	4	0.045	0.000	216.8
tai50a	2	0.180	0.000	268.2	2	**0.155**	0.000	255.4
tai60a	0	0.294	0.036	300.0	0	**0.235**	0.036	300.0
tai80a	0	0.536	0.397	300.0	0	**0.486**	0.413	300.0
tai100a	0	0.378	0.247	300.0	0	**0.337**	0.227	300.0
tai20b	**10**	0.000	0.000	0.0	8	0.091	0.000	60.0
tai25b	10	0.000	0.000	**0.0**	10	0.000	0.000	0.0
tai30b	10	0.000	0.000	**0.0**	10	0.000	0.000	0.2
tai35b	10	0.000	0.000	**0.1**	10	0.000	0.000	18.8
tai40b	10	0.000	0.000	**0.1**	10	0.000	0.000	0.6
tai50b	10	0.000	0.000	8.4	10	0.000	0.000	**7.6**
tai60b	10	0.000	0.000	18.4	10	0.000	0.000	**17.7**
tai80b	**10**	0.000	0.000	24.9	5	0.008	0.000	195.2
tai100b	**10**	0.000	0.000	48.6	1	0.033	0.000	276.9
tai150b	0	**0.126**	0.052	300.0	0	0.412	0.137	300.0
tai64c	**10**	0.000	0.000	1.2	7	0.014	0.000	90.0
tai256c	0	**0.238**	0.215	300.0	0	0.308	0.272	300.0
tho40	10	0.000	0.000	**2.1**	10	0.000	0.000	2.7
tho150	**3**	0.003	0.000	251.0	0	0.021	0.001	300.0
wil100	**10**	0.000	0.000	39.3	4	0.001	0.000	218.4
Summary	**261**	**0.054**	**0.029**	**82.7**	221	0.073	0.033	125.0

TABLE I: Comparison of GA-CPLS with ParEOTS

We first compare our prototype implementation (called GA-CPLS in the rest of this section) with ParEOTS, our reference. Table I reports for each benchmark:

- the number of times the BKS is reached out of 10 runs (#BKS).
- the Average Percentage Deviation (APD) which is the average of the 10 relative deviation *percentages*, computed as follows: $100 \times \frac{F(sol)-BKS}{BKS}$, where $F(sol)$ is the cost of the obtained solution. This information is useful for comparing the average quality of solutions (across the 10 executions) found by the solvers.
- the Best Percentage Deviation (BPD) which corresponds to the relative deviation percentage of the best solution found among the 10 executions.
- the average execution time given in second which corresponds to the actual elapsed times, and include the time to install all solver instances, solve the problem, communications and the time to detect and propagate the termination.

To compare the two solvers, we first compare the number of BKS found, then (in case of a tie), the APD and finally the execution time. For each benchmark, the best-performing solver row is highlighted and the discriminant field is enhanced in bold font. GA-CPLS clearly outperforms ParEOTS. Several instances are now systematically solved at each replication (e.g. `sko90` or `tai100b` or `wil100`). Moreover, for the very difficult `tho150` which was never solved by ParEOTS, GA-CPLS can reach the BKS 3 times. ParEOTS is better on `tai50a`-`tai100a`. These problems require a very strong intensification to reach the BKS. It would be interesting to experiment with different parameters for GA-CPLS to increase its level of intensification on these problems. The "Summary" row gives relevant information: GA-CPLS reaches the BKS 261 times (over the 330 possible) while ParEOTS only reaches 221. The APD are on average better than with ParEOTS; similarly for the best reached solution. Finally, regarding execution time, our prototype solver is, on average, 1.5 times fastet than ParEOTS. The difference is particularly impressive on instances like: `tai35b` (ParEOTS needs 18.784*sec* while GA-CPLS only needs 0.068*sec*. Clearly, the hybridization with a GA now works really well and definitely improves performance.

It is also interesting to compare GA-CPLS with state-of-arts parallel solvers for QAP. We have selected the following challengers:

- COSEARCH [25], which is a cooperative parallel method based on tabu search, genetic algorithms and a kick operator. The (published) results are obtained from a heterogeneous grid computing platform using around 150 processors.
- CPTS [26]: a cooperative parallel tabu search algorithm in which processors exchange information to intensify or diversify the search. The results are obtained from the cited paper using 10 Intel Itanium processors (1.3 GHz).
- PHA [27]: a hybrid approach based on parallel island genetic algorithm and robust tabu search. Experimentation was performed on a cluster of 46 nodes, each with 8 cores.
- TLBO-RTS [28]: a hybrid teaching-learning based optimization approach. This algorithm combines the teaching-learning based optimization to optimize a population, then, the population is given to a robust tabu search technique. TLBO-RTS supports a parallel execution using a parallel island technique. The results are obtained using 40 to 50 processors.

Table II reports all available information from the cited papers. Execution 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). These times are mainly given for information purpose. Unfortunately, some methods do not provide the number of obtained BKS (obviously an APD $= 0.000$ corresponds to #BKS = 10). For this reason

	GA-CPLS 32 cores			COSEARCH 150 cores		CPTS 10 cores		PHA 46 cores		TLBO-RTS 40-50 cores		
	APD	BPD	time	APD	BPD	APD	time	APD	time	APD	BPD	time
els19	**0.000**	0.000	0.0	**0.000**	0.000	**0.000**	6					
kra30a	**0.000**	0.000	0.0									
sko56	**0.000**	0.000	3.2			**0.000**	21:00	**0.000**	16:14	**0.000**	0.000	1:21:36
sko64	**0.000**	0.000	2.9	0.003	0.000	**0.000**	42:54	**0.000**	23:06	**0.000**	0.000	1:59:18
sko72	**0.000**	0.000	7.3			**0.000**	1:09:36	**0.000**	33:37	**0.000**	0.000	2:50:48
sko81	**0.000**	0.000	0:56			**0.000**	2:01:24	**0.000**	39:52			
sko90	**0.000**	0.000	0:14			**0.000**	3:13:42	**0.000**	40:31	**0.000**	0.000	5:42:48
sko100a	**0.000**	0.000	0:43	0.054	0.000	**0.000**	5:04:48	**0.000**	41:43	0.003	0.000	9:54:18
sko100b	**0.000**	0.000	0:36			**0.000**	5:09:36	**0.000**	42:19	0.005	0.000	8:02:36
sko100c	**0.000**	0.000	0:57			**0.000**	5:16:06	**0.000**	42:11	**0.000**	0.000	8:28:30
sko100d	**0.000**	0.000	0:39			**0.000**	5:09:48	**0.000**	41:54	0.009	0.007	8:29:24
sko100e	**0.000**	0.000	0:39			**0.000**	5:09:06	**0.000**	42:28	0.005	0.000	10:14:30
sko100f	0.001	0.000	1:23			0.003	5:10:18	**0.000**	41:58	0.005	0.000	8:02:36
tai40a	0.022	0.000	3:04			0.148	3:30	**0.000**	10:36	**0.000**	0.000	29:00
tai50a	0.180	0.000	4:28			0.440	10:18	**0.000**	12:44	0.360	0.321	55:00
tai60a	0.294	0.036	5:00			0.476	26:24	**0.000**	19:34	0.410	0.388	1:35:18
tai80a	**0.536**	0.397	5:00			0.691	1:34:48	0.644	39:58	0.870	0.850	3:59:30
tai100a	**0.378**	0.247	5:00	0.861	0.723	0.589	4:21:12	0.537	1:11:53	0.596	0.575	8:03:18
tai20b	**0.000**	0.000	0.0			**0.000**	6	**0.000**	0:22			
tai25b	**0.000**	0.000	0.0			**0.000**	0:24	**0.000**	0:34			
tai30b	**0.000**	0.000	0.0			**0.000**	1:12	**0.000**	0:48			
tai35b	**0.000**	0.000	0.1	**0.000**	0.000	**0.000**	2:24	**0.000**	1:06	**0.000**	0.000	22:24
tai40b	**0.000**	0.000	0.1			**0.000**	4:30	**0.000**	1:34			
tai50b	**0.000**	0.000	8.4			**0.000**	13:48	**0.000**	5:49			
tai60b	**0.000**	0.000	0:18			**0.000**	30:24	**0.000**	9:29			
tai80b	**0.000**	0.000	0:24			**0.000**	1:50:54	**0.000**	27:42	**0.000**	0.000	3:59:00
tai100b	**0.000**	0.000	0:48	0.135	0.000	0.001	4:01:00	**0.000**	42:30	**0.000**	0.000	8:28:12
tai150b	0.126	0.052	5:00	0.439	0.008	0.076	122:57:48	0.026	2:57:26	**0.015**	0.011	7:08:30
tai64c	**0.000**	0.000	1.2	**0.000**	0.000	**0.000**	20:36	**0.000**	12:56	**0.000**	0.000	1:57:42
tai256c	0.238	0.215	5:00	0.170	0.120	0.136	122:57:48	**0.116**	11:27:10			
tho40	**0.000**	0.000	2.1									
tho150	**0.003**	0.000	4:11	0.066	0.010	0.013	33:11:42	0.009	2:57:24	0.030	0.027	9:16:36
wil100	**0.000**	0.000	0:39	0.009	0.000	**0.000**	5:16:36	**0.000**	41:58	**0.000**	0.000	8:02:36
Summary	0.054	0.029	**1:22**	0.158	0.078	0.083	10:51:24	**0.044**	58:15	0.105	0.099	5:25:36

TABLE II: Comparison of GA-CPLS with other parallel solvers

we consider the APD, and highlight the solver with the best value. GA-CPLS has a better, or at least equal, APD than COSEARCH for all common instances. Comparing the performance with COSEARCH, out of 11 instances, it is apparent that COSEARCH performs better than GA-CPLS in only one instance: tai256c; note that no execution time data is available for COSEARCH. Regarding CPTS, out of 31 common instances, it appears that it only performs better than GA-CPLS in two cases: tai150b and tai256c, but taking a very long time: more than 122 hours! Here, we obtain a slightly worse value but in just 5 minutes. A similar situation appears with TLBO-RTS which only performs better than GA-CPLS on two instances: tai40a and tai150b, for which it requires 7 hours. The comparison with PHA is more balanced; both systems are able to reach the BKS of most problems at each replication. PHA is better in 6 cases (sko100f, tai40a-tai60a, tai150b and tai256c). These instances require strong intensification and/or are large size (greater than 100). Clearly, a 5 minutes time limit is too short for these problems (PHA needs around 3 hours to obtain its performance on tai150b and more than 11 hours for tai256c). However, it is worth mentioning, our method returns better solutions for 3 large instances (tai80a, tai100a and the very difficult tho150). The "summary" row confirms that PHA provides slightly better solutions than GA-CPLS while our prototype is clearly faster. In conclusion, in terms of solution quality, GA-CPLS is one of the best performing parallel methods. Moreover, in terms of execution time, our method has the best run times while maintaining a high quality of the solution and making efficient use of parallel resources. We are aware that COSEARCH and CPTS were reported on several years ago, on slower machines, and that both TLBO-RTS and PHA run with a different (higher) core count, but the results are significant nonetheless.

V. Conclusion and Further Work

Hybrid cooperative parallel metaheuristics is currently one of the most efficient approaches to address very hard Combinatorial Optimization Problems. Designing such a solver is a very complex task which requires very strong skills and considerable effort in parallel programming, modeling, metaheuristics, hybridization and runtime tuning. Recently, the CPLS framework has been proposed to ease the design and the implementation of such hybrid parallel solvers. However, CPLS is oriented towards single-solution metaheuristics, a bias which we have experimentally confirmed. It is all the more

regrettable, considering that population-based metaheuristics have shown to be very efficient when combined with other local search procedures, as testified by the success of memetic algorithms.

In this paper we analyzed the base mechanisms of CPLS which prevent efficient hybridization with population-based metaheuristics. We then proposed a modification of CPLS, for which we did an initial prototype for the very difficult Quadratic Assignment Problem (QAP). Our experiments with this prototype indicate very good performance on the hardest instances of QAPLib, making it already competitive with other state-of-the-art parallel solvers.

Our medium-term goal is to propose a new framework, based on CPLS, to fully support hybridization with population-based metaheuristics. For this we first need to experiment more extensively, in order to analyze the impact of parameters such as the size of the population, the reset percentages of the population, different values for parameters which control the communication. Regarding QAP, we will try to further improve our Genetic Algorithm (e.g. testing with the Insert Path Crossover as operator and crossover).

We will then test this improved solver on even more difficult QAP instances (e.g. instances proposed by Drezner, Palubeckis and Carvalho & Rahmann). We also plan to intensify the experimental evaluation, to cover a wider base of test cases and runtime configurations, namely the progression in speedup attained by scaling the computational resources (cores and nodes).

REFERENCES

[1] S. Kirkpatrick, C. Gelatt Jr, and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: http://www.sciencemag.org/cgi/doi/10.1126/science.220.4598.671

[2] E. Aarts and J. K. Lenstra, *Local Search in Combinatorial Optimization*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1997.

[3] F. Glover and M. Laguna, *Tabu Search*. Kluwer Academic Publishers, jul 1997.

[4] N. Mladenovic and P. Hansen, "Variable Neighborhood Search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[5] P. Codognet and D. Diaz, "Yet Another Local Search Method for Constraint Solving," in *Stochastic Algorithms: Foundations and Applications*, K. Steinhöfel, Ed. London: Springer Berlin Heidelberg, 2001, pp. 342–344.

[6] S. Boettcher and A. Percus, "Nature's way of optimizing," *Artificial Intelligence*, vol. 119, no. 12, pp. 275–286, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370200000072

[7] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[8] C. Solnon, "Ants Can Solve Constraint Satisfaction Problems," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 347–357, 2002.

[9] J. Kennedy and R. Eberhart, "Particle swarm optimization," *IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995.

[10] C. Blum and A. Roli, "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[13] T. Crainic, M. Gendreau, P. Hansen, and N. Mladenovic, "Cooperative Parallel Variable Neighborhood Search for the p-Median," *Journal of Heuristics*, vol. 10, no. 3, pp. 293–314, 2004.

[11] H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.

[12] P. Moscato and C. Cotta, "Memetic algorithms," *Handbook of Applied Optimization*, vol. 157, p. 168, 2002.

[14] M. Toulouse, T. Crainic, and M. Gendreau, "Communication Issues in Designing Cooperative Multi-Thread Parallel Searches," in *Meta-Heuristics: Theory&Applications*, I. Osman and J. Kelly, Eds. Norwell, MA.: Kluwer Academic Publishers, 1995, pp. 501–522.

[15] D. Munera, D. Diaz, S. Abreu, and P. Codognet, "Flexible Cooperation in Parallel Local Search," in *Symposium on Applied Computing, SAC'2014*. Gyeongju, Korea: ACM Press, 2014, pp. 1360–1361. [Online]. Available: http://dblp.uni-trier.de/db/conf/sac/sac2014.html{#}MuneraDAC14

[16] D. Munera, D. Diaz, S. Abreu, and P. Codognet, "A Parametric Framework for Cooperative Parallel Local Search," in *European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP)*, ser. Lecture Notes in Computer Science, C. Blum and G. Ochoa, Eds., vol. 8600. Granada, Spain: Springer, 2014, pp. 13–24. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44320-0{_}2

[17] D. Munera, D. Diaz, S. Abreu, F. Rossi, V. Saraswat, and P. Codognet, "Solving Hard Stable Matching Problems via Local Search and Cooperative Parallelization," in *AAAI*, Austin, TX, USA, 2015.

[18] D. Munera, D. Diaz, and S. Abreu, "Solving the Quadratic Assignment Problem with Cooperative Parallel Extremal Optimization," in *The 16th European Conference on Evolutionary Computation in Combinatorial Optimisation*, Porto, 2016.

[19] D. Munera, D. Diaz, and S. Abreu, "Hybridization as Cooperative Parallelism for the Quadratic Assignment Problem," in *10th International Workshop, HM 2016*, ser. Lecture Notes in Computer Science, vol. 9668. Plymouth, UK: Springer International Publishing, 2016, pp. 47–61. [Online]. Available: http://link.springer.com/10.1007/978-3-319-39636-1

[20] C. Fleurent, Jacques, and J. A. Ferland, "Genetic hybrids for the quadratic assignment problem," in *DIMACS Series in Mathematics and Theoretical Computer Science*. American Mathematical Society, 1993, pp. 173–187.

[21] Z. Drezner, "Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem," *Computers & Operations Research*, vol. 35, no. 3, pp. 717–736, 2008. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2006.05.004

[22] R. K. Ahuja, J. B. Orlin, and A. Tiwari, "A greedy genetic algorithm for the quadratic assignment problem," *Computers & Operations Research*, vol. 27, no. 10, pp. 917 – 934, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054899000672

[23] E. Lalla-Ruiz, C. Expsito-Izquierdo, B. Melin-Batista, and J. M. Moreno-Vega, "A hybrid biased random key genetic algorithm for the quadratic assignment problem," *Information Processing Letters*, vol. 116, no. 8, pp. 513 – 520, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020019016300254

[24] U. Benlic and J.-k. Hao, "Memetic Search for the Quadratic Assignment Problem," *Expert Systems With Applications*, vol. 42, no. 1, pp. 584–595, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2014.08.011

[25] E.-G. Talbi and V. Bachelet, "COSEARCH: A parallel cooperative metaheuristic," *Journal of Mathematical Modelling and Algorithms*, vol. 5, no. 1, pp. 5–22, 2006.

[26] T. James, C. Rego, and F. Glover, "A Cooperative Parallel Tabu Search Algorithm for the Quadratic Assignment Problem," *European Journal of Operational Research*, 2009.

[27] U. Tosun, "On the Performance of Parallel Hybrid Algorithms for the Solution of the Quadratic Assignment Problem," *Engineering Applications of Artificial Intelligence*, vol. 39, pp. 267–278, 2015.

[28] T. Dokeroglu, "Hybrid Teaching-Learning-Based Optimization Algorithms for the Quadratic Assignment Problem," *Computers and Industrial Engineering*, vol. 85, pp. 86–101, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.cie.2015.03.001