Weaving of Metaheuristics with Cooperative Parallelism

Jheisson López[1,2], Danny Múnera[2], Daniel Diaz[3], and Salvador Abreu[4](✉)

[1] National University of General Sarmiento, Buenos Aires, Argentina
jalopez@ungs.edu.ar
[2] University of Antioquia, Medellin, Colombia
danny.munera@udea.edu.co
[3] University of Paris 1/CRI, Paris, France
daniel.diaz@univ-paris1.fr
[4] University of Évora/LISP, Évora, Portugal
spa@uevora.pt

Abstract. We propose PHYSH (Parallel HYbridization for Simple Heuristics), a framework to ease the design and implementation of hybrid metaheuristics via cooperative parallelism. With this framework, the user only needs encode each of the desired metaheuristics and may rely on PHYSH for parallelization, cooperation and hybridization. PHYSH supports the combination of population-based and single-solution metaheuristics and enables the user to control the tradeoff between intensification and diversification. We also provide an open-source implementation of this framework which we use to model the Quadratic Assignment Problem (QAP) with a hybrid solver, combining three metaheuristics. We present experimental evidence that PHYSH brings significant improvements over competing approaches, as witness the performance on representative hard instances of QAP.

1 Introduction

Metaheuristics are often the most efficient approach to address the hardest Combinatorial Optimization Problems (COP). 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 (S-MH) maintain, modify and stepwise improve on a single candidate solution, hence the term *trajectory-based* metaheuristics. On the other hand, population-based metaheuristics (P-MH), modify and improve a *population*, i.e. a set of individuals corresponding to candidate solutions.

Metaheuristics generally implement two main search strategies: *intensification* and *diversification*, also called exploitation and exploration [1]. Intensification guides the solver to deeply explore a promising part of the search space. In

This work was partly funded by FCT under grant UID/CEC/4668/2016 (LISP).

contrast, diversification aims at extending the search onto different parts of the search space [8]. In order to obtain the best performance, a metaheuristic should provide a useful balance between intensification and diversification. By design, some heuristics are good at one but not at the other.

More generally, each metaheuristic can perform differently according to the problem or even instance being solved. A single metaheuristic will also vary depending on its chosen tuning parameters. The current trend is thus to design *hybrid* metaheuristics, by combining different methods in order to benefit from the individual advantages of each one [9]. An effective approach consists in combining an evolutionary algorithm with a single-solution method (very often a local search procedure). These hybrid methods are called *memetic algorithms* [10]. Hybrid metaheuristics tend to be complex procedures, tricky to design, implement and tune, therefore, most of them only combine two methods.

Despite the good results obtained with the use of hybrid metaheuristics, it is still necessary to reduce the processing times needed for harder instances [18]. One possible answer entails resorting to parallel execution [5]. 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 first is easiest to implement on parallel computers, as the metaheuristics run oblivious to each other and execution stops as soon as any of them finds a solution [16,22]. For some problems this provides very good results [3] but in many cases the speedup tends to taper off when increasing the number of processors [13]. A cooperative approach entails adding a communication mechanism in order to share or exchange information among solver instances during the search process [20]. However, designing an efficient cooperative method is a dauntingly complex task [4], and many issues must be solved: *What information is exchanged? Between which processes is it exchanged? When is it exchanged? How is it exchanged? How is the imported data used?* [21]. 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 cooperative scheme may significantly outperform the independent approach.

In 2014, we proposed the Cooperative Parallel Local Search framework (CPLS) for the cooperative parallel execution of local search metaheuristics [13,14]. The user only has to encode the LS procedure and can rely on CPLS to obtain a parallel application able to run concurrently and cooperatively several instances of this LS procedure. At runtime, the outcome is a parallel exploration of the search space with candidate solution interchange. All low-level parallel mechanisms (task creation/destruction, mapping to physical resources, synchronization, communication, termination, ...) are transparently handled by CPLS. CPLS has been successfully used to tackle stable matching problems [15] and very difficult instances of the Quadratic Assignment Problem (QAP) [12]. We later extended CPLS to allow the user to run *different* metaheuristics in parallel. CPLS has enabled a simpler way to hybridize metaheuristics, by exploiting its solution-sharing cooperative parallelism mechanism. At runtime, the parallel instances of each different metaheuristic communicate their best solutions, and one of them may forgo its current computation to *adopt* a better solution from the others, hoping to 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 [11]. However, CPLS was designed for local search metaheuristics: its cooperation mechanisms can only handle single-solution metaheuristics. When pursuing hybridization this limitation becomes too severe.

In this paper we propose a framework for the Parallel HYbridization of Simple Heuristics (PHYSH), which eases the implementation of hybrid metaheuristics using cooperative parallelism. As in CPLS, the user only needs to code each of the desired metaheuristics, independently, and may rely on PHYSH to provide both parallelism and cooperation to get "the best of both worlds". PHYSH is highly parametric and the user has control over the trade-off between intensification and diversification. Single-solutions methods are in charge of intensifying the search while population-based methods can be used to provide diversification through the evolution of a population. We also sketch a prototype implementation, available as an open source library written in the IBM X10 concurrent programming language. Needs only code the desired metaheuristic, PHYSH API. We used this implementation to develop a parallel solver for QAP by hybridizing 3 metaheuristics: a Genetic Algorithm, an Extremal Optimization procedure and a Tabu Search method. The resulting solver performs extremely well on the hardest instances of QAP.

The rest of this paper is organized as follows: in Sect. 2 we describe the framework, while in Sect. 3 we discuss implementation issues. In Sect. 4 we carry out an experimental evaluation on hard QAP instances. Finally, we summarize our results and draw plans for future developments in Sect. 5.

2 The PHYSH Framework

The aim of PHYSH is to offer the user an environment for the development of hybrid and parallel metaheuristics. By transparently managing all of the technical details of parallel programming as well as mechanisms for hybridization, PHYSH allows the user to focus on metaheuristic codings and problem modeling. The resulting parallel hybrid search process starts from different points in the search space, attempting to ensure convergence on proper solutions while escaping local extrema. We achieve this with multiple concurrent worker *teams*, each one tasked with visiting a different region of the search space. Figure 1 depicts a search space where red regions contain high-quality solutions which is explored by 4 teams in parallel: 2 teams are intensifying the search in a promising region while the 2 others are diversifying the search in order to reach other rich region.

Teams are composed of the following components: a set of search units, a diverse and an elite populations. The main active element of the framework is the *search unit* (SU) which encapsulates a single metaheuristic that can be

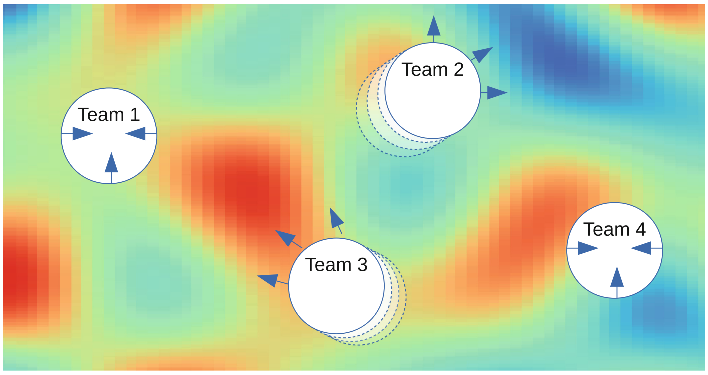

Fig. 1. PHYSH search process (Color figure online)

either a S-MH or a P-MH. If the SU contains a S-MH, it takes the role of an *intensifier* otherwise (implementing P-MH) it takes the role of a *diversifier*. The elite population (EP) retains the best individuals found by the intensifiers, while the diverse population (DP) holds individuals sent by diversifiers. The interaction patterns between the different components that make up a team establish a parametric four-way migratory flow process (see Fig. 2). In each case a parameter controls the migration frequency.[1]

- *Elite Emigration* (`ee`): from the intensifier worker to the EP.
- *Diverse Emigration* (`de`): from the diversifier worker to the DP.
- *Elite Immigration* (`ei`): from the EP to the diversifier worker.
- *Diverse Immigration* (`di`): from the DP to the intensifier worker.

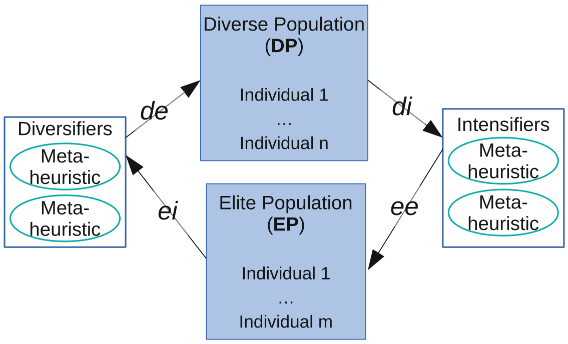

Fig. 2. PHYSH team structure

[1] Terms "immigration" and "emigration" are from the metaheuristics point-of-view.

The intensifiers (resp. diversifiers) must apply a *selection policy* to determine which individuals emigrate to the EP (resp. DP). EP and DP population implement an *acceptance policy* for deciding whether the incoming individual is accepted or rejected (discarded). For immigration flows, intensifiers and diversifiers request individuals respectively from DP and EP. Once again a *selection policy* is implemented on the populations to define how to chose an individual and send it to the corresponding entity.

Our framework follows the design principle of separating policy form mechanism. As a result, this process constitutes a flexible interaction model between intensifiers and diversifiers which eases the hybridization of simple metaheuristics, effectively promoting cross-fertilization among different types.

Different mechanisms can be implemented for the same policy e.g., an *elitist* or *non-elitist* mechanism. In the first case we favor elite individuals, while in the second we may, for instance, select the most diverse individual or even adopt a stochastic stance. We may assign several mechanisms for the same policy to a component, in that case the mechanisms are applied in a round-robin fashion until they succeed in the (selection/acceptance) pipeline.

An intuitive configuration could assign elitist mechanism to the intensifiers, non-elitist mechanism to the diversifiers, and both types of mechanism to the populations. We decided to make this a configurable option, as it provides rich choices of search strategy.

In PHYSH, the programmer may easily control the balance between intensification and diversification (see Fig. 3). Take the proportion of SUs used for the intensifiers vs. diversifiers: it may be tuned to achieve a specific balance. For instance, if more intensification is needed for a given instance, one may increase the number of SUs in the role of intensifier. The intensification/diversification level may also be tweaked by varying the number of teams in the execution: given a fixed number of processing units, using more teams with a lower SU count will increase the diversification on the search.

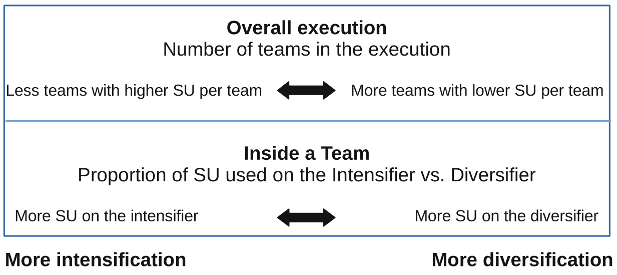
Overall execution
Number of teams in the execution
Less teams with higher SU per team
More teams with lower SU per team
Inside a Team
Proportion of SU used on the Intensifier vs. Diversifier
More SU on the intensifier
More SU on the diversifier
More intensification
More diversification

Fig. 3. PHYSH intensification-diversification control

The PHYSH framework is designed to adapt to different parallel architectures: shared-memory multiprocessors as well as distributed systems with network-connected MP nodes. SUs are meant to be mapped to physical processors, while teams may be configured very flexibly.

3 PHYSH×10: A Prototype Implementation

We implemented our prototype in the X10 programming language which is a high level object-oriented programming language, focused on concurrency and distribution. X10 supports a wide range of parallel platforms and it has been in active developemnt by IBM research since 2004. X10 is based on the Asyncronous Partitioned Global Address Space model (APGAS). Using this model, computation and data are partitioned into *places* which are abstractions for mutable, shared-memory regions that can contain global references to locations in other places, as well as worker threads operating on this memory.

In adoption of common practice for metaheuristics tools, PHYSH×10 presents a clear separation between available metaheuristics and the problems that can be solved. We have implemented a genetic algorithm (GA), a robust tabu search (RoTS) and an extremal optimization (EO) procedure. Consequently, the diversifiers are built from SUs that contain a GA, while the other two metaheuristics are available for the SUs in the intensifiers. Figure 4 displays the main classes of PHYSH×10, a few application-specific ones and their relationships.

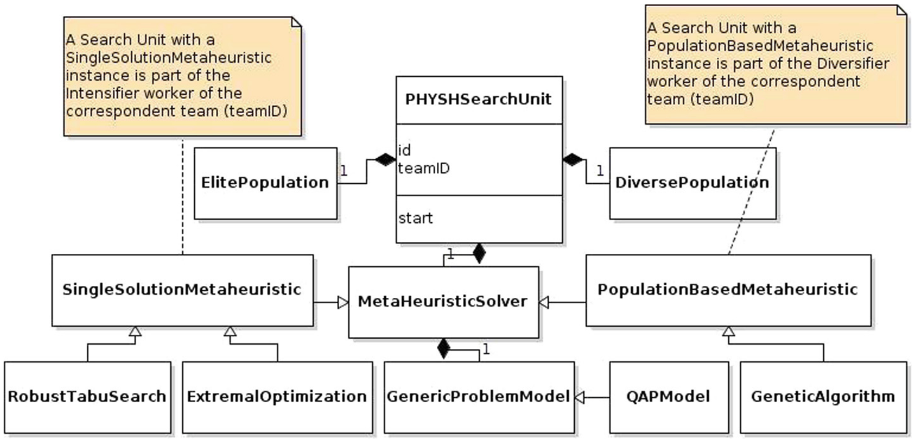
A Search Unit with a SingleSolutionMetaheuristic instance is part of the Intensifier worker of the correspondent team (teamID)
A Search Unit with a PopulationBasedMetaheuristic instance is part of the Diversifier worker of the correspondent team (teamID)
PHYSHSearchUnit
id
teamID
start
ElitePopulation
DiversePopulation
1
1
1
1
SingleSolutionMetaheuristic
MetaHeuristicSolver
PopulationBasedMetaheuristic
RobustTabuSearch
ExtremalOptimization
GenericProblemModel
QAPModel
GeneticAlgorithm

Fig. 4. PHYSH×10 UML diagram of the main classes

PHYSH×10 uses the features offered by X10-APGAS model to assign available physical processing resources. Accordingly, each SU is allocated to an X10 *place*, so that intensifiers and diversifiers operate as a distributed system. As explain above, SUs are grouped to form teams. Each team is composed of `tz` SUs. The number of teams is thus #`cores`/`tz`. *EP* and *DP* populations are bound to a single SU within each team. These populations have a parametric size i.e., `epz` individuals for EP and `dpz` individuals for DP. Each component implements the most convenient mechanism for the acceptance and selection criteria.

At present, PHYSH×10 provides the following *selection mechanisms*:

– *Best*: best individual found in the search process.
– *Current*: all eligible individuals are selected (for S-MH the current configuration is the unique eligible individual.)
– *Random*: an individual is randomly selected from the elegible set.

The following *acceptance mechanism* are also provided:

– *Elitist*: The individual is accepted if it is better than the worst in the target population (if it is not present yet.)
– *Probabilistic*: The individual is accepted, regardless of its cost, with a given probability (if it is not present yet.)
– *Maximizer*: The individual is accepted if its average distance to the other individuals is greater than a defined threshold.

Intensifiers implement the *current* mechanism for the selection policy i.e., SU sends its current configuration to perform the emigration to the EP. Parameter elite emigration period (`eep`) controls the periodicity of this communication. Intensifiers also request an immigrant individual from DP each diverse immigration period (`dip`). To accept or deny this individual intensifiers implement an *elitist* mechanism for the acceptance policy (for S-MH the target "population" is current solution of the metaheuristic).

Diversifiers implements a *random* mechanism for the selection policy. This mechanism requires a parameter to define the percentage of the population eligible for emigration (`ppfe`). The individual to emigrate is randomly chosen among the top `ppfe%` of the SU's population (the best individuals). Parameter diverse emigration period (`dep`) controls the periodicity of this emigration process. Diversifiers also request an immigrant from EP each elite immigration period (`eip`). Individual diversifiers implement an *elitist* acceptance mechanism.

To simplify the assignment of these parameters we define two general values: emigration period `ep` and immigration period `ip`. Considering teams of size `tz` (a team embeds `tz` SUs) and a problem of size of `n`, the default values are computed as follows: `eip` = `ep`/`tz`, `dep` = `ip`/`n`, `eep` = `ep`/`n` and `dip` = `ip`.

4 Experimental Evaluation

To evaluate the performance of our framework, we developed PHYSH-QAP[2]: a parallel hybrid solver for QAP which combines three metaheuristics: a Genetic Algorithm (GA) [7], a Robust Tabu Search (RoTS) [19] and an Extremal Optimization procedure (EO) [12]. PHYSH-QAP is built on top of PHYSH×10. We consider three sets of very hard benchmarks: the 33 hardest instances of QAPLIB and two sets of even harder instances: Drezners `dre`*XX* and Palubeckis `Inst`*XX* instances. 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.

[2] The source code is available from https://github.com/jlopezrf/COPSolver-V_2.0.

4.1 Evaluation of PHYSH-QAP on QAPLIB

QAPLIB is a collection of 134 QAP problems of different sizes [2]. The instances are generally named as `name`XX where `name` corresponds to the first letters of the author and XX is the size of the problem. For each instance, QAPLIB also includes the Best Known Solution (BKS), which is sometimes the optimum. Many QAPLIB instances are easy for a parallel solver, we therefore only considered the 33 hardest instances, as reported in [12]. Each problem instance is executed 30 times, stopping as soon as the BKS is reached or when a time limit of 5 min is hit, using 64 cores. PHYSH-QAP was configured with four teams, each of size `tz` $= 16$ embedding 1 diversifier running GA, 8 intensifiers running RoTS and 7 intensifiers running EO. The size for the elite population and the diverse population was set to 4 (`epz` $=$ `dpz` $= 4$). The `ppfe` parameter is instance-dependent (we only experimented with values 0, 50 and 100).

Table 1 has all the results. For each instance we have the BKS, the `ppfe` parameter used, the number of times the BKS is reached (across the 30 executions), the Average Percentage Deviation (ADP) which is the average of the 30 relative deviation percentages computed as follows: $100 \times \frac{Sol-BKS}{BKS}$, the Best Percentage Deviation (BPD) which corresponds to the relative deviation percentage of the best solution found among the 30 executions, the Worst Percentage Deviation (WPD) which corresponds to the worst solution, the average execution time given in seconds which corresponds to the elapsed (wall) time, and includes the time to install all solver instances, solve the problem communications and the time to detect and propagate the termination and, finally, the average number of times the winning SU adopted an individual from the diverse/elite populations.

On this set of 33 hardest instances, even with a limit of time of 5 min PHYSH-QAP is able to find the BKS at least once for 29 instances. Moreover, it is even able to reach the BKS systematically at each replication for 21 instances. For the 4 remaining instances (`tai80a`, `tai100a`, `tai150b` and `tai256c`), the quality of solutions returned by PHYSH-QAP is very good, around 0.2% of the BKS. The summary row has interesting numbers. The average ADP is only 0.051%, the average BPD is 0.024% and the average WPD is 0.079%. These numbers confirm that all runs provide high quality solutions; even the worst runs provide good results. For instance, in the worst case (`tai80a`), the worst solution among 30 runs is within just 0.547% of the BKS. Performance-wise, PHYSH-QAP averages just 96 s to find a solution. If we do not take into account the 4 unsolved instances (whose time is bounded by the time limit), the average run time is 70 s. The number of adopted configurations on the wining SU is 4.2, on average, showing that the hybridization is effectively taking place.

Comparison with Another Parallel Hybrid Solver for QAP: ParEOTS is a hybrid solver for QAP built on the top of the CPLS framework. ParEOTS combines RoTS and EO and has shown to perform very well. Indeed, on the hardest instances of QAPLIB, it outperforms most of state-of-the-art methods [11].

For this comparison we selected the 15 hardest instances from Table 1. We then ran ParEOTS using the parameters reported in [11] in the same execution

Table 1. PHYSH-QAP on hard QAPLIB instances (64 cores, timeout = 5 min)

	BKS	ppfe	#BKS	APD	BPD	WPD	Time	#adopt
els19	17212548	50	30	0	0	0	0.0	0.1
kra30a	88900	100	30	0	0	0	0.0	0
sko56	34458	50	30	0	0	0	1.8	0.5
sko64	48498	50	30	0	0	0	2.0	0.3
sko72	66256	50	30	0	0	0	9.8	1.2
sko81	90998	50	30	0	0	0	22.4	1.6
sko90	115534	100	30	0	0	0	104.4	6.3
sko100a	152002	100	27	0.001	0	0.016	129.3	3.4
sko100b	153890	0	30	0	0	0	52.4	1.0
sko100c	147862	0	30	0	0	0	77.5	1.3
sko100d	149576	0	30	0	0	0	64.9	1.2
sko100e	149150	0	30	0	0	0	49.4	0.9
sko100f	149036	100	29	0.000	0	0.005	103.7	2.4
tai40a	3139370	50	20	0.025	0	0.074	173.9	4.7
tai50a	4938796	100	8	0.133	0	0.336	262.0	10.3
tai60a	7205962	0	1	0.242	0	0.368	292.7	9.5
tai80a	13499184	50	0	0.460	0.335	0.547	300.0	8.6
tai100a	21052466	0	0	0.352	0.167	0.463	300.0	22.6
tai20b	122455319	100	30	0	0	0	0.0	0.0
tai25b	344355646	50	30	0	0	0	0.0	0.1
tai30b	637117113	50	30	0	0	0	0.1	1.3
tai35b	283315445	0	30	0	0	0	0.3	1.8
tai40b	637250948	0	30	0	0	0	0.4	2.5
tai50b	458821517	0	30	0	0	0	6.7	0
tai60b	608215054	0	30	0	0	0	10.9	0
tai80b	818415043	0	30	0	0	0	42.0	1.3
tai100b	1185996137	0	29	0.001	0	0.024	143.4	4.9
tai150b	498896643	50	0	0.190	0.085	0.410	300.0	10.1
tai64c	1855928	0	30	0	0	0	0.2	0.1
tai256c	44759294	50	0	0.264	0.211	0.312	300.0	4.4
tho40	240516	0	30	0	0	0	1.1	0.1
tho150	8133398	0	1	0.021	0	0.043	298.8	29.7
wil100	273038	100	26	0.000	0	0.002	144.7	5.2
Summary			**771**	**0.051**	**0.024**	**0.079**	**96.8**	**4.2**

environment as for PHYSH-QAP: same machine, using 64 cores with a time limit of 5 min and 30 repetitions per instance.

Table 2. PHYSH-QAP vs ParEOTS (64 cores, timeout = 5 min)

	PHYSH-QAP			ParEOTS		
	#BKS	APD	Time	#BKS	APD	Time
`sko81`	**30**	0	22.4	25	0.002	70.6
`sko90`	**30**	0	104.4	29	0.000	116.5
`sko100a`	**27**	0.001	129.3	25	0.003	128.9
`sko100c`	**30**	0	77.5	29	0.000	127.3
`tai40a`	20	0.025	**173.9**	20	0.025	184.2
`tai50a`	**8**	0.133	262.0	3	0.144	289.8
`tai60a`	**1**	0.242	292.7	0	0.270	300.0
`tai80a`	0	0.460	300	0	0.460	300.0
`tai100a`	0	**0.352**	300	0	0.358	300.0
`tai100b`	**29**	0.001	143.4	22	0.015	181.4
`tai150b`	0	0.190	300.0	0	**0.130**	300.0
`tai64c`	**30**	0	0.2	28	0.004	20.0
`tai256c`	0	**0.264**	300.0	0	0.272	300.0
`tho150`	**1**	0.021	298.8	0	0.019	300.0
`wil100`	**26**	0	144.7	14	0.001	213.9
Summary	**232**	**0.113**	**190.0**	195	0.114	208.8

Table 2 presents the results. To compare the two solvers, compare the number of BKS found, then (in case of a tie), the APDs and finally the execution times. For each benchmark, the best-performing solver row is highlighted and the discriminant field is enhanced in bold font. PHYSH-QAP outperforms ParEOTS on 13 out of 15 of the hardest QAPLIB instances while the reverse only occurs for one instance (`tai150b`). Our implementation systematically solves 4 instances which are not fully solved on ParEOTS (`sko81`, `sko90`, `sko100c` and `tai64c`). The summary row shows that PHYSH-QAP obtains a total *#BKS* higher than ParEOTS (232 vs. 195). It is worth noticing that this quality of solutions is obtained in a shorter execution time (190 s vs. 208 s).

4.2 Evaluation of PHYSH-QAP on Harder Instances

We evaluated our hybrid solver on two sets of instances, artificially crafted to be very difficult for metaheuristics: the `dre`*XX* instances proposed by Drezner et al. [6] and the `Inst`*XX* instances by Palubeckis [17]. These instances are generated with a known optimum. For this test we used the same machine, with 128 cores and a time limit of 10 min with 30 repetitions. We used the same framework configuration as in Sect. 4.1 for QAPLIB. We could not yet experiment with different values for `ppfe` so we use `ppfe = 100` for all instances.

Table 3. PHYSH-QAP on Drezner and Palubeckis (128 cores, timeout = 10 min)

	#BKS	APD	best	Time
`dre21`	**30**	0	**356**	0.0
`dre24`	**30**	0	**396**	0.0
`dre28`	**30**	0	**476**	0.0
`dre30`	**30**	0	**508**	0.1
`dre42`	**30**	0	**764**	0.9
`dre56`	**30**	0	**1086**	11.5
`dre72`	**30**	0	**1452**	90.9
`dre90`	**23**	2.757	**1838**	281.2
`dre110`	**6**	14.997	**2264**	549.4
`dre132`	**5**	11.404	**2744**	558.2
Summary	244	2.915		149.2

	#BKS	APD	best	Time
`Inst20`	**30**	0	**81536**	0.0
`Inst30`	**30**	0	**271092**	0.1
`Inst40`	**30**	0	**837900**	3.2
`Inst50`	**30**	0	**1840356**	7.7
`Inst60`	**30**	0	**2967464**	11.8
`Inst70`	**30**	0	**5815290**	35.7
`Inst80`	**30**	0	**6597966**	78.0
`Inst100`	**17**	0.038	**15008994**	476.4
`Inst150`	0	0.122	**58411484**	600.0
`Inst200`	0	0.123	**75495960**	600.0
Summary	227	0.028		181.3

Table 3 presents the results obtained on both benchmarks. Regarding Drezner's instances, PHYSH-QAP is able to optimally solve all instances. To best of our knowledge, no other dedicated solver for QAP has ever reported an optimal solution either for `dre110` or `dre132` (highlighted in green in the table). Moreover, all instances of size $n \leq 72$ are systematically solved at each replication. Regarding Palubeckis' instances, the optimum is found for instances with $n \leq 100$ (and systematically found at each replication for $n \leq 80$). For size $n > 100$, clearly a limit of 10 min is too short. Nevertheless the quality of obtained solutions within this time limit is very good with an APD around 0.12%. It is worth noting that for `Inst150` and `Inst200`, the solution computed by PHYSH-QAP improves on the best solutions ever published (corresponding best costs computed are highlighted in green in Table 3).

5 Conclusion and Future Directions

We have proposed PHYSH: a new framework for the efficient resolution of Combinatorial Optimization Problems combining single-solution metaheuristics, population-based metaheuristics, cooperative parallelism and hybridization. We have used our X10 implementation of this framework to construct a hybrid solver for the Quadratic Assignment Problem which combines up to three metaheuristics. This solver turns out to perform exceptionally well, particularly on very hard instances of QAP.

We plan to study the impact of each parameter in more detail; including experimentation with techniques for parameter auto-tuning, e.g. using F-Race. We also plan to add new metaheuristics to the prototype, particularly population-based methods. This enriched implementation we will enable uas to address a wider range of problems. Finally, it will be interesting to experiment on different parallel architectures, for instance GPGPUs or Intel MIC, using the X10 language, which greatly abstracts on machine architectural specificities.

References

1. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. **35**(3), 268–308 (2003)
2. Burkard, R.E., Karisch, S., Rendl, F.: QAPLIB - a quadratic assignment problem library. Eur. J. Oper. Res. **55**(1), 115–119 (1991)
3. 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), 30–56 (2015)
4. Crainic, T., Gendreau, M., Hansen, P., Mladenovic, N.: Cooperative parallel variable neighborhood search for the p-median. J. Heuristics **10**(3), 293–314 (2004)
5. Crainic, T., Toulouse, M.: Parallel meta-heuristics. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. ISOR, vol. 146, pp. 497–541. Springer, Boston (2010). https://doi.org/10.1007/978-1-4419-1665-5_17
6. Drezner, Z.: The extended concentric tabu for the quadratic assignment problem. Eur. J. Oper. Res. **160**(2), 416–422 (2005)
7. Drezner, Z.: Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. Comput. Oper. Res. **35**(3), 717–736 (2008)
8. Hoos, H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann/Elsevier, Burlington (2004)
9. Misevicius, A.: A tabu search algorithm for the quadratic assignment problem. Comput. Optim. Appl. **30**(1), 95–111 (2005)
10. Moscato, P., Cotta, C.: Memetic algorithms. In: Handbook of Applied Optimization, vol. 157, p. 168 (2002)
11. Munera, D., Diaz, D., Abreu, S.: Hybridization as cooperative parallelism for the quadratic assignment problem. In: Blesa, M.J., et al. (eds.) HM 2016. LNCS, vol. 9668, pp. 47–61. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39636-1_4
12. Munera, D., Diaz, D., Abreu, S.: Solving the quadratic assignment problem with cooperative parallel extremal optimization. In: Chicano, F., Hu, B., García-Sánchez, P. (eds.) EvoCOP 2016. LNCS, vol. 9595, pp. 251–266. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30698-8_17
13. Munera, D., Diaz, D., Abreu, S., Codognet, P.: A parametric framework for cooperative parallel local search. In: Blum, C., Ochoa, G. (eds.) EvoCOP 2014. LNCS, vol. 8600, pp. 13–24. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44320-0_2
14. Munera, D., Diaz, D., Abreu, S., Codognet, P.: Flexible cooperation in parallel local search. In: Symposium on Applied Computing, SAC 2014, pp. 1360–1361. ACM Press, Gyeongju (2014)
15. Munera, D., Diaz, D., Abreu, S., Rossi, F., Saraswat, V., Codognet, P.: Solving hard stable matching problems via local search and cooperative parallelization. In: AAAI, Austin, TX, USA (2015)
16. Novoa, C., Qasem, A., Chaparala, A.: A SIMD tabu search implementation for solving the quadratic assignment problem with GPU acceleration. In: Proceedings of the 2015 XSEDE Conference on Scientific Advancements Enabled by Enhanced Cyberinfrastructure - XSEDE 2015, pp. 1–8 (2015)
17. Palubeckis, G.: An algorithm for construction of test cases for the quadratic assignment problem. Inform. Lith. Acad. Sci. **11**(3), 281–296 (2000)
18. Saifullah Hussin, M.: Stochastic local search algorithms for single and bi-objective quadratic assignment problems. Ph.D. thesis. Université de Bruxelles (2016)

19. Taillard, É.: Robust taboo search for the quadratic assignment problem. Parallel Comput. **17**(4–5), 443–455 (1991)
20. Talbi, E.G., Bachelet, V.: COSEARCH: a parallel cooperative metaheuristic. J. Math. Model. Algorithms **5**(1), 5–22 (2006)
21. Toulouse, M., Crainic, T., Gendreau, M.: Communication issues in designing cooperative multi-thread parallel searches. In: Osman, I., Kelly, J. (eds.) Meta-Heuristics: Theory & Applications, pp. 501–522. Kluwer Academic Publishers, Norwell (1995)
22. Tsutsui, S., Fujimoto, N.: An analytical study of parallel GA with independent runs on GPUs. In: Tsutsui, S., Collet, P. (eds.) Massively Parallel Evolutionary Computation on GPGPUs. NCS, vol. 8, pp. 105–120. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37959-8_6