Parallel Local Search

Philippe Codognet, Danny Munera, Daniel Diaz and Salvador Abreu

Abstract Local search metaheuristics are a recognized means of solving hard combinatorial problems. Over the last couple of decades, significant advances have been made in terms of the formalization, applicability and performance of these methods. Key to the performance aspect is the increased availability of parallel hardware, which turns out to be largely exploitable by this class of procedures. As real-life cases of combinatorial optimization easily degrade into intractable territory for exact or approximation algorithms, local search metaheuristics hold undeniable interest. This situation is further compounded by the good adequacy exhibited by this class of search procedures for large-scale parallel operation. In this chapter we explore and discuss ways which lead to parallelization in local search.

1 Introduction

Stemming from the pioneering work on the Traveling Salesman Problem (TSP) by Flood [44] and Croes [36] in the 1950s and then Lin [72] in the 1960s, the interest in Local Search for solving large combinatorial problems has been growing since the last decade of the twentieth century and has attracted much attention from both the Operations Research and the Artificial Intelligence communities. Local search is used for finding optimal or near-optimal solutions to real-life problems when the

Philippe Codognet
University Pierre & Marie Curie/LIP6, France, e-mail: philippe.codognet@upmc.fr

Danny Munera
University of Antioquia, Medellin, Colombia, e-mail: danny.munera@udea.edu.co

Daniel Diaz
University Paris 1/CRI, France, e-mail: daniel.diaz@univ-paris1.fr

Salvador Abreu
University of Évora/LISP/CRI, Portugal, e-mail: spa@di.uevora.pt

search space is too large to be explored by complete search algorithms, such as Mixed Integer Programming or Constraint Solving [1, 65, 56]. Efficient general-purpose systems for local search now exist, for instance the Comet [115], which has been parallelized for small clusters of PCs, [83, 84], or the Localsolver system [49].

Local search algorithms start from a random configuration and try to improve this configuration, little by little, by small changes in the values of the problem variables. Hence the term "local search" as, at each time step, only new configurations that are "neighbors" of the current configuration are explored. The definition of what constitutes a neighborhood will of course be problem-dependent, but basically it consists in changing the value of a few variables only (usually one or two). The advantage of local search methods is that they will usually quickly converge towards a solution (if the optimality criterion and the notion of neighborhood are defined correctly) and not exhaustively explore the entire search space. These methods naturally lead to concurrent execution, by considering the development of several configurations at the same time. This can be done sequentially by maintaining a pool of candidate configurations or in parallel if adequate hardware is available. Due to their simple algorithmic structure, local search methods therefore naturally exhibit various forms of parallelism, either with or without communication, and can be implemented on various types of parallel architectures such as multicore machines, grids or clusters, GPUs, or massively parallel machines. Indeed parallel implementation of local search methods has been studied since the early 1990s, when parallel machines started to become widely available; see [117, 116] for a general survey and concepts, or [97] for basic parallel versions of tabu search, simulated annealing, GRASP and genetic algorithms. With the increasing availability of PC clusters in the early 2000s this domain became active again [6, 35], and can further take advantage of the major advances in hardware in the last decade such as GPUs and massively parallel machines with thousands or tens of thousands of cores. However, although many methods have been developed and implemented in the last two decades, most of these experiments have been done for small-scale multiprocessors, thus giving performance evaluation for a few tens of cores at best. Only very few implementations of efficient local search solvers on larger machines have ever been reported, leaving open the question of the scalability of parallel local search in the age of exascale machines [99].

In the rest of this chapter we will present a general panorama of parallel local search methods. After a presentation of the basic mechanisms of local search methods in Section 2 and their sources of parallelism in Section 3, we will detail Single-walk approaches in Section 4, then Independent multi-walk methods in Section 5 and finally Cooperative multi-walk approaches in Section 6. Section 7 shows the effectiveness of parallel local search on two hard problems: the Stable Matching Problem and the Quadratic Assignment Problem. A short conclusion and future work end the chapter.

2 Local Search Metaheuristics

Metaheuristic methods aim at finding the optimal solutions (among all possible solutions) of a Combinatorial Optimization Problem. They have been proven to be very efficient on a wide variety of these problems. A metaheuristic is defined as a set of strategies for exploring the *search space* of a problem by using different methods [19]. Metaheuristics are high-level procedures using choices (i.e., heuristics) to limit the part of the search space that actually gets visited, in order to make problems tractable.

Metaheuristics generally implement two main search strategies: *intensification* and *diversification*, also called exploitation and exploration [19]. Intensification guides the solver to deeply explore a promising part of the search space. In contrast, diversification aims at extending the search into different parts of the search space [63]. In order to obtain the best performance, a metaheuristic should provide a useful balance between intensification and diversification. However, by design, some heuristics are better at intensifying the search while others are better at diversifying it. More generally, each metaheuristic has it own strengths and weaknesses. The current trend is therefore to design *hybrid* metaheuristics, by combining different metaheuristics in order to benefit from the individual advantages of each method.

In this chapter we are especially interested in local search metaheuristics; the interested reader can consult several surveys on metaheuristics [109, 98, 19, 24, 105, 106].

Local search methods (also known as trajectory methods) explore the search space by iteratively making small changes to a single solution (the current solution). These methods generally start from a randomly generated solution candidate but other strategies exist to start from a more promising initial solution constructed heuristically. At each iteration a local search method performs a single *move* (i.e., a small change to the current solution). The set of all possible moves is called the *neighborhood* (see Figure 1).

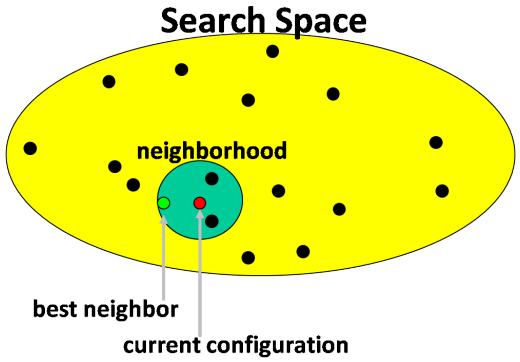

Fig. 1 Local search

At each iteration, a solution from the neighborhood of the current solution is selected to become the new current solution. Different strategies may be used to select the next move, for instance selecting the *best* move from the neighborhood (*hill climbing*), or the *first* move that improves the current solution (which is thus dependent on the order in which the moves are considered), or selecting a *random* improving solution.

When the neighborhood does not contain any improving solution, the metaheuristic has reached a *local optimum*[1]. Metaheuristics must provide strategies to avoid becoming trapped in local optima. They detect this situation in order to move to some other region of the search space. The simplest strategies to escape from a local optimum are to restart the search from a new (usually random) point or to perform a large perturbation of the current solution. These strategies are called multi-start local search (MLS) or iterative local search (ILS) [73]. There are other approaches and it is also possible to combine them.

By design, local search methods are very efficient at intensifying the search. However, they generally include some (simple) strategies to diversify the search (which are often executed when a local optimum is reached). Here we present the most important local search methods.

Tabu search methods [54, 52, 53] use a memory structure to avoid getting trapped in a local optimum. The main idea is to improve the basic hill-climbing algorithm by maintaining a *tabu list* of recently visited solutions (in practice, some approximations are necessary to avoid memory explosion). These solutions become prohibited (hence the term "tabu") to discourage the search from returning to previously visited places. Generally, an *aspiration criterion* is used to authorize an otherwise tabu move to be performed, in special circumstances (e.g., if it improves on the best solution found so far). The time an element remains tabu is called the *tabu tenure*. This parameter has a great influence on the efficiency of tabu search procedures and must be well tuned.

Simulated Annealing (SA) [70] is based on the annealing process of a crystalline solid used in metallurgy to improve the quality of a solid. For this, the cycles of slow cooling and heating (annealing) are alternated in order to reach a minimal energy state, which corresponds to a stable structure of the metal. Starting from a high temperature (at which the material is liquid), the cooling phase solidifies the material by a gradual decrease of the temperature. The SA method is based on this process to allow moves that result in solutions of worse quality than the current solution, in order to escape from local optima. At each iteration, it randomly selects a neighbor among its neighborhood. If it improves the current solution the move is adopted. Otherwise (a local optimum is reached) the probability of making this move is controlled by a parameter called the *temperature*. This temperature decreases during the search process; thus at the beginning of the search the probability of accepting worse moves is high but it gradually decreases, converging to a simple iterative improvement algorithm. Usually a Boltzmann distribution is used to compute the

[1] The term is opposed to *global optimum* which is the best possible solution to the optimization problem. The reached local optimum may actually coincide with the global optimum, but the method is generally unable to detect this occurrence.

probability to accept a worse quality solution (taking into account both the current temperature and how much the solution is degraded).

Variable neighborhood search (VNS) methods [86] escape from a local optimum by changing the neighborhood structure using different move types. The basic idea in VNS is that a local optimum relative to a given move type can be improved using a different move type (since the optimum is w.r.t. the neighborhood of the current solution). The search concludes when the current solution cannot be improved with all possible move types. It is thus important to correctly define the number and types of neighborhoods to be considered and the order in which they are tried. When these parameters are well tuned the VNS metaheuristic provides high-quality solutions.

Adaptive Search (AS) [31] is a generic, domain-independent, constraint-based local search method. AS takes advantage of the structure of the problem, in terms of constraints and variables, in order to guide the search more precisely than a single global cost function. Indeed, a cost is also associated with each constraint that models the problem, measuring the degree of violation of the constraint in the current solution candidate. This cost is then spread over all variables involved in the constraint (e.g., using a weight linked to the coefficient of the variable in a linear constraint). The worst variable is selected for update (i.e., to *move*), with the neighborhood being the set of all possible values for this "culprit" variable. Finally, AS maintains a tabu list of recently modified variables which led to local optima, but also implements a reset mechanism as used in ILS methods.

Extremal Optimization (EO) [21, 22, 20] is a metaheuristic inspired by self-organizing processes often found in nature. It is based on the concept of *Self-Organized Criticality* (SOC) initially proposed by Bak [15, 13], and in particular on the Bak-Sneppen model of SOC [14]. 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 value is updated, i.e., its fitness is replaced by a new random value. This change also affects all other species connected to this "culprit" element and their fitness value also gets updated. This results in an *extremal* process that 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. The EO metaheuristic follows this line: it inspects the current solution, selects the worst variable (the one with the lowest fitness) and replaces its value by a random value (this corresponds to a move). However, always selecting the worst variable can lead to a deterministic behavior and the algorithm may stay blocked in a local minimum. To avoid this, the authors propose an extended algorithm; which first ranks the variables in increasing order of fitness (the worst variable has thus a rank $k = 1$) and then resorts to a probability function over the ranks k in order to introduce uncertainty in the search process: $P(\tau;k) = k^{-\tau}$. This power-law probability distribution depends on a single parameter τ, which is problem-dependent. Depending on the value of τ, EO provides a wide variety of search strategies from pure random walk ($\tau = 0$) to deterministic (greedy) search ($\tau \to \infty$). With an adequate value of τ, EO cannot be trapped in local minima since any variable is likely to mutate (even if the worst ones are privileged). This parameter can be tuned by the user.

While simple, local search procedures have been successfully used to find high-quality solutions for many Combinatorial Optimization Problems. They are also often a part of a hybrid metaheuristic to intensify the search around a promising solution found by another metaheuristic. However, there are some hard (real-life) problems for which the limit to consider the execution time as "reasonable" is rapidly reached, even using metaheuristics. It is unquestionable that the more computational resources are available, the more complex problems may be solved. It is therefore natural to consider exploiting the various forms of augmented computational power that are currently available, as conveniently as feasible.

3 Sources of Parallelism

Apart from domain-decomposition methods and population-based methods (such as genetic algorithms), [117] distinguishes between single-walk and multiple-walk methods for local search. Single-walk methods consist in using parallelism inside a single search process, e.g., for parallelizing the exploration of the neighborhood (see for instance [74] for such a method making use of GPUs for the parallel phase). Multiple-walk methods (parallel execution of multi-start methods) consist in developing concurrent explorations of the search space, either independently or cooperatively with some communication between concurrent processes. Sophisticated cooperative strategies for multiple-walk methods can be devised by using solution pools [34], but require shared memory or emulation of central memory in distributed clusters, thus impacting on performance.

3.1 Single-Walk and Multiple-Walk Methods

Figures 2 and 3 below show in a graphical way the different parallel trajectories of single-walk and multiple-walk methods.

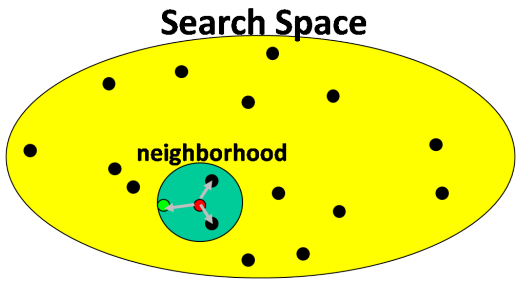

Fig. 2 Single-walk parallelism

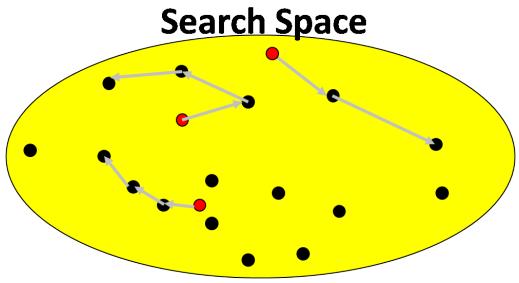

Fig. 3 Multiple-walk parallelism

Single-walk parallelism is limited to the neighborhood of the current solution and parallel processes need to be synchronized in order to choose the most promising neighbor and commit to the next solution. Multiple-walk parallelism explores a wider portion of the search space, limited only by the number of available concurrent processes. A key point is that independent multiple-walk methods are the easiest to implement on parallel computers, as they require no communication between processes; hence they are equivalent to parallel multi-start methods. On the other hand, one has to take care to ensure a good diversification of the search processes, which can only be achieved through communication between concurrent processes. Therefore, communication of information between concurrent processes could, if implemented without much overhead, improve the overall search. This type of parallelism is called cooperative multiple-walk parallelism. We will detail in the following sections the different methods that have been proposed in the literature for the single-walk approach, the independent multiple-walk approach and the cooperative multiple-walk approach.

3.2 Parallel Speedups and Runtime Distributions

Since [117, 116], it has been believed that combinatorial problems can enjoy a linear speedup when implemented in parallel by independent multiple-walks if solutions are uniformly distributed in the search space and if the method is able to diversify correctly. Thus, *in theory*, if such a method is implemented on a machine with n processors, the initial problem instance will be solved with a speedup factor of n. We will see that this is in fact not so easy to achieve in practice, especially when considering implementation on massively parallel multiprocessors, e.g., with thousands of processors. Moreover, when considering the latest cooperative methods and hybridization between different types of solvers, better performance can be achieved amounting to super-linear speedups.

But let us first see how to better analyze the execution times of local search algorithms, both sequentially and in parallel, in order to better understand the behavior and potential parallelization of such algorithms on different problem instances. In-

deed the parallel speedup depends not only on the algorithm at work, but also on the structure of the problem instance which it is attempting to solved. Most papers on the performance of stochastic local search algorithms focus on the average execution time in order to measure the performance of the method, both for sequential and parallel executions. However, a more detailed analysis could be done by looking at the whole series of execution times. Indeed, because of the many stochastic choices within any local search method, the runtime on the same problem instance might vary significantly from one execution to another. Thus by considering the execution time of a local search method on a given problem instance as a random variable and by observing the execution time over many runs, the runtime behavior can be characterized by its statistical distribution. This study of so-called runtime distributions has been initially proposed in [62] for stochastic local search algorithms for the SAT problem. In this context, the property of having a linear parallel speedup in solving a given problem instance by a stochastic algorithm has been proven only under the assumption that the probability of finding a solution in a given time t follows an exponential law, that is, if the runtime behavior follows a pure exponential distribution (non-shifted). This behavior has been conjectured for local search solvers on the SAT problem in [61, 62], and shown experimentally for the GRASP metaheuristics on some combinatorial problems [4], but it is not always the case for other types of problems. Although it is very difficult to formally prove that the execution of some stochastic algorithm on a given problem instance follows an exponential distribution, it is easy to verify this experimentally. Indeed, as introduced in [5, 103], this can be done by constructing so-called *time-to-target plots*, in which the probability of having found a solution as a function of the elapsed time is measured.

However, when considering not only exponential distributions, one has to look directly at the runtime distributions and analyze them with statistical tools. Such an analysis of the scalability of independent multiple-walk local search methods has been proposed in [114] and developed in [113], where a general framework is presented in order to estimate the parallel performance of any Las Vegas algorithm [12] by analyzing the runtime behavior of the sequential version of the algorithm. Indeed, by approximating the runtime distribution of the sequential process with statistical methods, the runtime behavior of a multiple-walk parallel process can be predicted by a model based on order statistics [38]. Experiments show that the estimation is quite accurate and predicts performance close to the empirical data, with a deviation limited to about 20%. It also shows that, depending on the problem, runtime distributions can be approximated by two types of distributions, exponential (shifted and non-shifted) and lognormal, being much more complex than a pure (non-shifted) exponential distribution, which would give rise to a linear parallel speedup. In the cases of a shifted exponential distribution (the most common one) or a lognormal distribution, the speedup is no longer linear, but admits a finite limit when the number of processors goes toward infinity, and is thus bounded.

4 Single-Walk Approaches

Single-walk methods use parallelism *within* a single search process, e.g., by parallelizing the most computationally expensive functions of the algorithm. Runtime profiling of local search procedures reveals that one of the most resource-consuming parts is the evaluation of the neighborhood. This situation makes this function an attractive target to be parallelized with single-walk search procedures. The basic idea is to divide the neighborhood into different parts, which are then independently evaluated, in parallel. This strategy is called *neighborhood decomposition*.

In [107], Taillard presents one of the first implementations of the single-walk strategy for local search methods. He proposes a neighborhood decomposition strategy applied to the tabu search method for solving large instances of the Quadratic Assignment Problem. The implemented prototype ran on a network of Transputers.

In 1994, Garcia et al. [47] presented a new parallel version of the tabu search metaheuristic, applied to solving the vehicle-routing problem with time windows constraints. They propose a master-slave architecture where the master creates a partition of the neighborhood and assigns the portions to the available processors (slaves). Each processor then explores its own neighborhood, identifies its best move, and sends this move back to the master processor.

Note that parallel activities involved in the neighborhood decomposition task need to be performed at *each iteration* of the algorithm. These activities have to be spawned and joined several times during the main algorithm execution, thereby inducing a significant overhead due to the management of fine-grained tasks. Dealing with this overhead is considered a major challenge in single-walk parallelization. For instance, in the aforementioned work by Taillard, the authors report a maximal parallel efficiency[2] of 85% using only 10 processors.

Recent years have seen a proliferation of GPUs; which, even though they are designed to perform mostly intensive graphical operations, have significant general compute ability and relatively low cost, so as to attract research on several different applications. Such is the case for single-walk parallelization in local search methods, where GPUs have emerged as a suitable architecture to implement the neighborhood decomposition. When the operations happen to be within their reach, GPUs can effectively operate on data much faster than traditional CPU architectures: doing neighborhood decomposition in parallel on GPUs has the potential to noticeably reduce the overhead of single-walk approaches. Luong et al. in [74, 75] present a parallel local search method that uses the neighborhood decomposition strategy performed by a GPU unit. They propose guidelines to efficiently implement the parallel evaluation of the neighborhood considering the idiosyncrasies of a GPU architecture (e.g., memory management and access, thread control, mapping of neighborhood solutions to GPU threads, etc.). This approach proved to be effective in solving different optimization problems, as witness the authors' report on parallel speedups, which range from 50 when using an entry-level GPU, up to 240

[2] Parallel efficiency: the division of the theoretical CPU time with an ideal speedup by the CPU time effectively observed.

with a higher-performance GPU board. This approach was tailored for embedding within the ParadisEO framework, as reported in [82].

Arbelaez and Codognet [9] present a parallel version of the adaptive search (AS) algorithm using both multiple-walk and single-walk parallelization. The solver takes advantage of the GPU architecture by executing multiple instances of the AS solver, but also and at the same time performing the evaluation of large neighborhoods in parallel, as previously described. The authors report a maximum speedup of 17 in solving two classical constraint satisfaction problems, and a speedup of 3 in solving the Costas Array problem.

Single-walk parallelization in GPU architectures presents rather good performance, however the implementation of local search methods on GPUs is far from trivial and the scalability of these approaches is limited, if nothing else, by Amdahl's law [8]. Amdahl's law states that the maximum speedup that may be expected from the parallelization of an algorithm is $1/s$ where s is the fraction of non-parallelizable parts of the algorithm. For instance, if a sequential algorithm is 90% parallelizable, then the theoretical maximum speedup one can ever expect by parallelizing this algorithm is 10, regardless of the number of processors in use.

5 Independent Multiple-Walk Approaches

Multiple-walk methods develop concurrent explorations of the search space, either *independently* or *cooperatively*. The independent multiple-walk scheme derives from the observation that local search processes, being mostly stochastic in nature, will exhibit different behavior from one run to the next. This will directly impact on the time it takes to complete an individual search, which will vary accordingly. The base insight is thus to have several instances execute concurrently, so as to collect the earliest or the best result.

Because they are concerned with processes whose execution is unrelated, independent multiple-walk methods tend to be relatively straightforward to implement on parallel computers and can lead – at least in theory – to linear speedups [117]. It should be noted, however, that this holds under the assumption that the time it takes to reach a solution obeys an exponential distribution. We will see that a more complex model may be required in order to explain the performance actually observed in larger-scale parallel executions.

5.1 Early Independent Multiple-Walk Methods

Early work, in 1996, by Rego and Roucairol [100] introduced a parallel variant of the tabu search metaheuristic, which they apply to the Vehicle-Routing Problem. This system uses the PVM parallel platform to perform independent parallel searches, starting from a common point but following different paths. Each search

reports back to a central hub, which in turn collects solutions, looking for a local optimum, which, in turn, is used to relaunch a new batch of searches. This algorithm mixes functional with data parallelism, and it uses slightly different instances of the tabu search procedure, in the hopes that the ensuing diversity will promote better collective performance. The authors report that the parallel system begets *higher-quality* solutions, although at the expense of a sometimes significantly *slower* computation. The reason for the performance impact is not very clear, but may be related to the parallel library overheads.

In 1999, Eikelder et al. [43] proposed a Sequential and Parallel Local Search Algorithm, applied to the Job Shop Scheduling problem. In this work, the authors recognize the impact of non-determinism in performing multiple instances of a local search procedure, and establish a process whereby the parallel speedup of a simple independent multiple-walk local search algorithm may be modeled. The proposed approach takes into account the success or failure of the search procedures, as well as the quality of the solutions found, for the definition of parallel speedup. The predicted times are a good match to the observed times in the authors' experiments, scaling to about 40 large-granularity processors. The predicted and observed speedups both appear to have a largely linear section, up to about 10 processors. Beyond that, performance gains suffer a visible drop, yet there remains an undeniable benefit from running independent multiple-walk searches in parallel.

A system by Mori and Ogita [87] was proposed in 2000, which also does tabu search in parallel, applying it to the reconfiguration of power distribution systems problem. One of the driving ideas is that carrying out multiple search processes in parallel, each with just a distinct value for the tabu tenure parameter, will lead to a faster convergence on an optimal solution, because of the subsequent diversity. The authors combine this with a parallel decomposition of the neighborhood, i.e., a form of functional parallelism. The results indicate that tabu search produces the best quality solutions among several metaheuristics (which include genetic algorithms and simulated annealing), in both the sequential and parallel versions. Likewise, the parallel tabu search procedure exhibits the highest performance of the set, notably so in the case where a moderate amount of parallelism is dedicated to the parallel neighborhood decomposition (two to four sub-neighborhoods).

Finding different approaches to structure the neighborhood of a candidate solution was essential to the work of Garcia-Lopez et al. [48], published in 2002. The authors propose a parallel method to do Variable Neighborhood Search, and apply it to the p-Median problem, taking large instances from TSPLIB [101]. This proposal follows three different takes on parallelism: either the local search, the variable neighborhood search or both become subject to parallel execution. In all cases, the parallel procedures execute independently, and the runtimes reflect a near-linear speedup with up to eight processors. The prototype implementation runs on a multicore system, resorting to a shared-memory configuration using OpenMP [37], and is therefore tied to that multiprocessor organization.

Another system was described in 2003, by Bortfeldt et al. [23], which carries out multiple independent tabu search procedures, running on top of a distributed system in the form of a network of workstations. The network of parallel processes

keeps tabs on the solutions found by each worker, storing them in a storage object for possible reuse by others. Even though the architecture is essentially that of independent multiple-walk parallelism, it may include various forms of information exchange among workers, as a consequence of the solution storage access pattern, by each participant. Solutions found by workers are made available to the entire network or just part of it, e.g., workers may be arranged in a ring topology. Workers may be selective as to which external solutions to look at and, should they perform better, adopt. The authors apply their prototype implementation to the Container-Loading Problem, with measurable solution quality improvements over competing approaches, namely the sequential tabu search and genetic algorithms-based solvers. Performance-wise, the parallel system actually requires more time to achieve its results and communication among workers only seems to yield minute improvements.

The 2010 work by Yazdani et al. [119] supplies another case of a parallel local search procedure: in this instance, Variable Neighborhood Search benefits from the diversification of neighborhood structures via the parallel independent exploration thereof. The parallel architecture adopted is that of shared-memory multicore processors. The authors apply their system to Flexible Job Shop Scheduling, a harder variant of the base problem, and provide experimental validation in a parallel setup with up to five processors. The results indicate that the Parallel Variable Neighborhood Search procedure computes good-quality solutions, when compared to competing approaches.

5.2 Recent Experiments and Performance Results

Work by Caniou et al. [28, 30, 29] presents a simple parallel scheme based on independent multiple-walks with no communication between processes during search, the sequential engine being based on the adaptive search metaheuristic. It was built using the MPI [45] parallel programming interface and was tested on different hardware platforms, of varying scale: up to a few hundred cores on the GRID'5000 platform in France and the Hitachi HA8000 and Fujitsu FX10 machines at the University of Tokyo and up to 8,000 cores on the JUGENE supercomputer at Jülich Supercomputing Centre. Performance evaluation on large instances of some classical Constraint Satisfaction Problems from CSPLIB [51], such as the Magic Square, Perfect Square and All-Interval problems, shows that speedups are very good for a few tens of cores (e.g., speedup of a factor of 20-25 on 32 cores), and correct up to a few hundreds of cores (e.g., speedup of a factor of 50-60 on 256 cores), but speedup then degrades, showing that not much parallelism could be further extracted even with a larger number of cores. Figure 4 shows the performance results of the parallel adaptive search method on these problems in the form of runtime speedups for a given number of cores.

However, another hard combinatorial benchmark, the Costas Arrays Problem (CAP), was also tested with instances of CAP up to 23 (large instances) and the experimental evaluation shows better parallel scalability. Indeed, parallel speedup

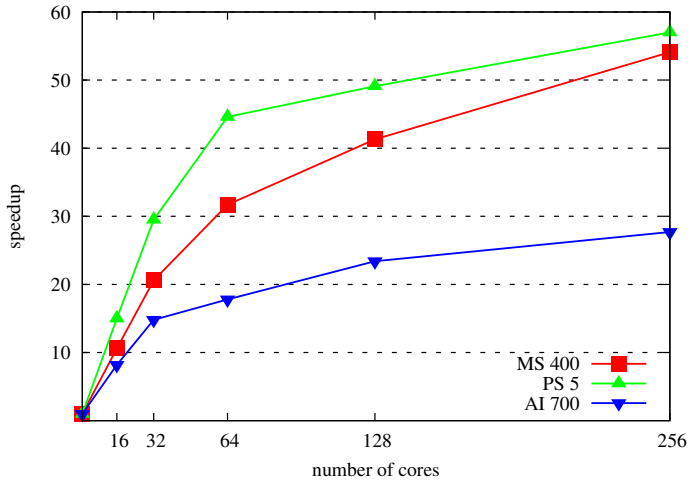

Fig. 4 Speedups for benchmark CSP programs on the HA8000 parallel machine, from [28]

scales very well (linearly) up to about 8,000 cores, on the JUGENE supercomputer. Figure 5 shows the performance results of the parallel adaptive search method on instances 21, 22 and 23 of CAP, in the form of runtime speedup for a given number of cores.

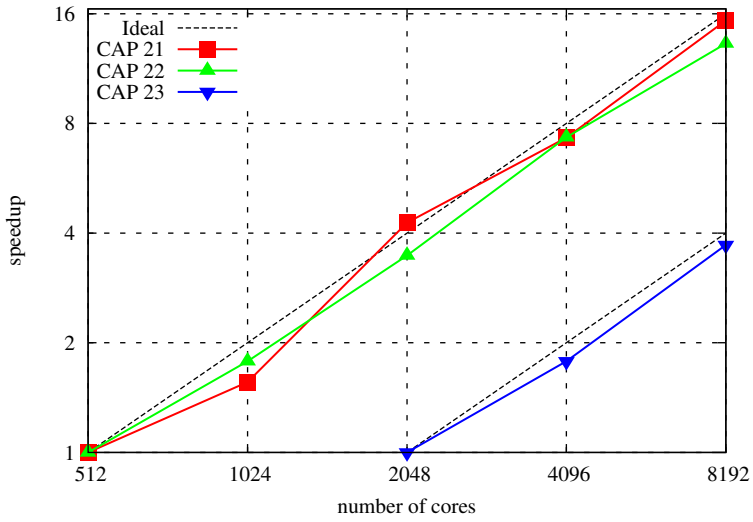

Fig. 5 Speedups for the Costas Array Problem on the JUGENE supercomputer, from [29]

This can be explained by the fact that the runtime distribution of the adaptive search metaheuristic on the CAP problem exhibits a nearly pure (non-shifted) exponential distribution; see [40] for details of experimental results. The authors also experimented with a limited form of cooperation among search processes (exchanging only solution costs between processes and performing restarts), but the results were not markedly different from the independent multiple-walk strategy.

It turns out that most independent multiple-walk procedures start off with good speedups attributable to parallel execution. However, this characteristic appears to hit a problem-dependent hard limit, which may be attributed to the lack of entropy (diversity) across the different runs which are being performed in parallel, and thereby bounds the usefulness of such a strategy, at larger scale.

Although there are stand-out exceptions, this diminishing returns situation becomes especially obvious when attempting to scale beyond a few dozen cores. This has prompted research into exploring more sophisticated parallel methods that can compensate for this performance drop, namely those that rely on some form of cooperation among worker threads, as discussed in section 6.

6 Cooperative Multiple-Walk Approaches

To overcome the limitations of the independent multiple-walk strategy, it is natural to consider a paradigm based on *cooperation*. This is the case of *Cooperative Multiple-walk* methods, which add a communication mechanism to the independent search strategy, in order to share or exchange information among solver instances during the search process. However, designing an efficient cooperative method is a very complex task, 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? [112]. The work presented in [76] studies these questions, and concludes that no one cooperative configuration may efficiently tackle all problems. Indeed, most cooperative choices are problem-dependent (and even instance-dependent).

According to the literature [117, 118, 19, 109], an *efficient* cooperative method should consider four essential functionalities: flexibility, adaptability, performance and scalability. Flexibility refers to the capability of a given method to tackle different problems, using different methods and providing hybrid behavior. Adaptability is related to the ability of a given method to adjust its cooperative behavior. In addition, a method has a good performance if it can obtain a high-quality solution in a short execution time. Finally, scalability refers to the ability of a given method to efficiently use a significant number of processing units (cores).

In this chapter, we analyze several approaches using the cooperative multiple-walk strategy. We identify three different kinds of algorithms: metaheuristic parallelization, agents-based and general frameworks.

6.1 Metaheuristic Parallelization Approaches

We first analyze cooperative methods based on metaheuristic parallelization. One of the oldest cooperative approaches was proposed in 1993 by Hogg and Williams [60]. The basic idea is to create multiple solver entities (metaheuristics) that share partial configurations (hints) through a centralized memory (blackboard). Each entity reports to the blackboard a hint at each step (based on its current state) with a given probability p. When the entity is at an appropriate decision point, it reads a hint from the blackboard with probability p. If p is set to zero, the algorithm behaves like independent search. The implementation of the method is dedicated to solving the graph coloring problem using two different heuristics: the Berlaz algorithm and heuristic repair. The experimental evaluation is performed using 10 agents, solving graphs with 100 nodes and comparing the performance of the independent and the cooperative approaches. The cooperative version presents better performance than the independent version in terms of the execution time, however the parallel scalability is not evaluated in this work.

In 1998, Aiex et al. proposed a cooperative parallel tabu search for solving the circuit partitioning problem [3]. This method implements a master-slave model composed of search processes (slaves) which implement different combinations of the initial solution algorithm and move attribute for a specialized tabu search metaheuristic. Periodically, the search processes exchange information (elite solutions) with the master node, which maintains a centralized shared memory for elite configurations. The parallel procedure is implemented using two different parallel programming languages, PVM (based on message passing) and Linda (based on the shared-memory model). The authors test both implementations on a set of problem instances from the ISCAS benchmark on an IBM SP-2 machine with 16 processors. Only 10 processors are used in the experimental evaluation, using one master and nine search processes. The implementation improves the solution quality for all problem instances with respect to the sequential version of the algorithm, the PVM version being 20% faster than the Linda implementation. This work does not present any evaluation of the performance in terms of the execution time and the parallel scalability.

Gendreau et al. [50], in 1999, proposed another master-slave scheme to parallelize the tabu search algorithm for solving the dynamic vehicle-routing and dispatching problem. The master entity manages an adaptive memory which is fed by a set of tabu search instances (slaves). The adaptive memory is used to create new initial solutions for slave processes. The authors present a prototype implementation, which runs on a network of 17 SUN UltraSparc workstations. The proposed method is compared with other heuristic approaches and obtains a better solution quality than its competitors. An evaluation of the parallel scalability is performed using up to 16 processors, showing that the solution quality is improved by increasing the number of processors involved.

Two similar methods are proposed to implement cooperation on the GRASP (Greedy Randomized Adaptive Search Procedure) and the Path Relinking metaheuristic [2, 102]. A distributed cooperation mechanism was proposed by Aiex et

al. in 2003 [2], which creates several search processes. Each process sends the best overall configuration to the other processes when the cost is improved. Each process maintains a local elite pool which possibly contains configurations from all processes. This pool is used as input for the Path Relinking phase. An experimental evaluation is carried out solving standard job shop scheduling test problems from Beasley's OR-Library. The experiments are done on an SGI Challenge computer composed of 28 R10000 MIPS processors, using 1, 2, 4, 8 and 16 processors. The prototype is coded in Fortran using the MPI library. The cooperative strategy obtains almost linear speedups, improving on the independent strategy; which, as expected, shows only a sub-linear speedup.

Ribeiro and Rosseti, in 2007, proposed another parallel cooperative approach also using GRASP and Path Relinking [102]. This method takes advantage of the multi-start behavior of the GRASP metaheuristic to implement a multiple-walks parallelization. In addition, a master-slave cooperative strategy is implemented. Slave processes send the best configurations to a master process, which maintains a centralized pool of elite solutions. Then the master can send back a new configuration to slave nodes upon request. A prototype implementation of this approach is developed using C and the MPI specification. The experiments are carried out on a cluster of 32 Pentium II 400MHz processors solving the randomly generated instances of the 2-path network design problem. The cooperative strategy presents smaller execution times and scales better than the independent implementation, obtaining almost linear speedups and reporting a maximum speedup of 17.6 using 32 cores. Although the two previous methods present fair parallel performances, the functionality is attached to GRASP behavior and to the problem nature, thus limiting its flexibility.

A cooperative parallel approach that uses the rollout algorithm for solving the Sequential Ordering Problem was proposed in 2003 by Guerriero and Mancini [57]. This approach presents a master-slave topology in which slaves are executed in parallel, running an instance of the rollout algorithm. Slave processes periodically send the best configurations found to the master, which maintains a centralized pool of configurations. The master restarts slaves with adjusted parameters using a new initial point from the pool. The cooperative mechanism can adapt its behavior by selecting the best parameters for the base algorithm. However, this cooperative approach is strongly linked to the rollout algorithm, limiting the possibility to use this technique with other metaheuristics. The parallel version of the algorithm was implemented in C++ using the MPI library. The experiments run on a cluster of nine nodes with two Pentium 1 GHz processors, solving 14 instances of the Sequential Ordering Problem (taken from the TSPLIB). The cooperative approach obtains a good solution quality for the given set of problems. The scalability of the algorithm is evaluated using 1, 2, 4 and 8 slaves (cores). The algorithm improves either the solution quality or the execution time used to find the best solution when increasing the number of slaves. However, the authors report that the rollout-like approach obtains a higher computational time to find good solutions compared with other state-of-the-art approaches.

The 2004 work by Crainic et al. [34] presents a master-slave cooperative method to solve the *p*-median problem based on the Variable Neighborhood Search (VNS) metaheuristic. The master process implements a central memory to maintain the best overall solution. The master also sends the initial configuration to slaves. The slave processes (VNS processes) perform the search and notify the master when improving the overall solution. A slave process asks the master for a new search point if it cannot improve its current configuration. An MPI implementation of this approach run on a 64-processor SUN enterprise machine with 400 MHz clock. The experiments use 1, 5, 10 and 15 processors, solving a set of problems from the TSPLIB benchmark. This strategy obtains significant gains in terms of execution time, maintaining a good solution quality. However the cooperation mechanism is strongly linked to the behavior of the VNS metaheuristic and to the problem model. The principles of this approach include avoiding using parameters, which is convenient for the user but not for the adaptability of the system.

In 2012, Cordeau and Maischberger [33] proposed a parallel iterated tabu search algorithm to solve vehicle-routing problems. The basic idea is to execute in parallel several iterated tabu search solver instances using different sets of parameters. The algorithm implements a communication mechanism to share the most promising configurations found in the search process. Each process can apply a crossover operator to the received configurations (with a given probability), in order to combine information of two different received configurations. The algorithm is implemented in C++ using the MPI libraries for the parallel version. The experiments run on a cluster composed of 128 nodes, each with a 3 GHz dual Intel Xeon CPU E5472 (i.e., four cores per node). This strategy is tested solving different variants of the vehicle-routing problem, using up to 80 cores, and obtaining good performances in terms of the solution quality (allowing the identification of new best known solutions for a large set of problems).

A cooperative approach based on the execution of multiple instances of the adaptive search solver was presented in 2013 by Machado et al. [77]. A single master solver instance sends every k iterations its current configuration to the other solver instances. Since this information is stored in a shared-memory structure, all the solver instances (threads) running on the node benefit from this communication. Each solver instance decides whether it adopts the received configuration or continues its current search process. This cooperative scheme was implemented using the GPI (Global Address Space Programming Interface) API for parallel applications running on clusters. The experiments are conducted on a cluster system with 155 nodes; each node includes a dual Intel Xeon 5148LV (i.e., four cores per node). This strategy is evaluated solving two constraint satisfaction problems from the CSPLib: *all-interval* and *magic-square*; and one hard real-life problem: the Costas Array Problem (CAP). The cooperative strategy presents no gain compared to the independent strategy, when solving the CAP. For the CSPLib problems the cooperative approach presents a better speedup than the independent strategy. The parallel scalability is evaluated using up to 512 cores; however the obtained speedups are sublinear for both cooperative and independent approaches. More recently, in 2015, Caniou et al. presented a similar approach in [29], which uses the same base algo-

rithm (adaptive search) and the same set of CSPLib problems. The authors propose a new cooperative approach in which only single integer values are exchanged between entities (as opposed to complex data types such as vectors of variables, i.e., a configuration). The receiver entity uses this information to decide whether it is convenient to develop a restart procedure. This strategy is evaluated on the Helios cluster of the GRID'5000 platform, using up to 128 cores. The results of the experimentation cannot show an improvement in the performance using this cooperative approach.

6.2 Agent-Based Approaches

Agent-based modeling is a powerful strategy that facilitates the implementation of cooperative approaches. In early work, in 1998, Talukdar et al. propose a multi-agent-based cooperative methodology to combine solving strategies [111]. The A-Team (asynchronous teams) framework allows agents to cooperate through a shared memory containing a population of configurations. Agents can create, modify or delete configurations from the shared memory. Furthermore, they can obtain elite configurations from the shared memory, which have probably been created by another agent, in order to cooperate and make the initial set of configurations evolve. The A-Team framework provides a good level of flexibility, because agents can implement different algorithms, and this method can be applied to different problems. The referenced paper does not report any experimental evaluation, however this approach has been used as the basis for many agent-based cooperative solvers.

In 2004, Milano and Roli presented a multi-agent metaheuristic architecture (called MAGMA) that can describe cooperative search or hybrid metaheuristics [85]. This architecture is based on a multi-level organization in which components (agents) are classified according to their capabilities. Low-level agents describe the basic functionality of metaheuristics. A top layer manages integration and cooperation of different solvers. Agents in the top layer can store partial or complete configurations and promote changes in lower layers in response to the gathered information. This approach provides a theoretical description that can be easily adapted to tackle different problems and to use different metaheuristics, thus providing fair flexibility and adaptability. Similarly to the A-Team strategy, MAGMA is considered as a generic framework; the referenced paper only provides an experimental evaluation in the appendix, where a guided-restart iterated local search algorithm is conceived as a combination of existing components in the MAGMA framework.

A multi-agent architecture was proposed in 2006 by Bachelet and Talbi for solving large-scale instances of the Quadratic Assignment Problem [110]. This method, called COSEARCH, is composed of a set of agents that perform specific tasks: search agent, intensifying agent and diversifying agent. COSEARCH implements as the main search agent a tabu search heuristic; for the diversifying agent, it uses a genetic algorithm; and for the intensifying agent, a kick operator is used. These agents share information through an adaptive memory that stores information about

the already visited areas of the search space and about the intrinsic nature of the elite solutions already found (initial and elite configurations). This strategy is evaluated solving a set of problem instances from the QAPLib benchmark. The experiments run on a heterogeneous parallel platform composed of around 150 workstations, using a significant number of cores. The results show COSEARCH presents better performance than a basic parallel multi-start strategy, in terms of execution time and solution quality.

The 2007 work by Aydin [11] proposed a study of different cooperative topologies for agent-based metaheuristics. This work tested three different schemes: A-Team, a multiple-island model and variable neighborhood search. The job shop scheduling problem is used to develop the experimental evaluation, which only considers the solution quality. All the schemes are developed using DREAM software [10] which is a Java-based framework that implements the distributed subpopulation model for evolutionary algorithms by using multi-agent technology. The main objective in this experimentation is to reveal more details about each strategy.

In 2009, Cadenas et al. presented a cooperative parallel hybrid strategy that uses machine learning techniques [26]. The system is composed of two different types of agents: metaheuristic and coordinator. Multiple instances of different metaheuristics are run in parallel by metaheuristic agents, which, simultaneously, share information through a blackboard data structure. One coordinator agent is used to analyze the information in the blackboard and to adapt the metaheuristic agents' behavior. The coordinator agent incorporates knowledge from an offline machine learning process. This knowledge helps the coordinator to guide the search and to adapt the behavior of the system to different situations. The authors also proposed a Java implementation of this strategy using tabu search, simulated annealing and genetic algorithms for the metaheuristic agent. This implementation is used to solve different instances of the knapsack problem. The experiments run on an Intel core2 Quad 1.66 GHz. The parallel version of the algorithm, which consists in a parallel execution of each metaheuristic, presents better performance than the non-cooperative approach. However, no comparison with state-of-the-art methods was carried out, and the evaluation does not include a parallel scalability analysis.

A Coalition-Based Metaheuristic (CBM) was presented in 2010 by Meignan et al. [81]. This approach is based on the agent metaheuristic framework and the hyperheuristic approach. The system architecture is composed of agents that implement a complete set of capabilities that make them suitable to perform different roles during the execution (strategist, guide, intensifier and diversifier). Agents exchange information in a decentralized and asynchronous manner. Agents use reinforcement learning and mimetism to adjust their behaviors. The authors present an implementation of the CBM in Java, running on a 3 GHz Pentium 4 processor. This implementation is tested solving the capacitated vehicle-routing problem and it shows competitive results in terms of both solution quality and execution time, using up to 20 parallel agents.

More recently, in 2014, Barbucha proposed another agent-based cooperative approach for population learning algorithms (called CPLA) [16]. This approach is based on the A-Team framework and on the population learning algorithm. The

basic idea is to make a population of individuals (configurations) evolve using a process that is divided into stages. At each stage, the population is improved using dedicated algorithms and different topologies. After each stage some elite individuals are promoted to the next stage. Agents have communication capabilities and, according to the stage, can share information with other agents (through a shared elite pool). Furthermore, multiple A-Teams can be run in parallel and exchange information through a migration manager agent. An implementation of the CPLA was developed using JADE (Java Agent Development Framework) [17]. The experiments run on the HOLK cluster built of 256 Intel Itanium 2 Dual Core processors solving the vehicle-routing problem with time windows. The results show CPLA has good performance in terms of solution quality and execution time, being competitive with state-of-the-art methods. No parallel scalability is analyzed in the referenced paper.

In 2016, Martin et al. proposed another agent-based cooperative approach [79]. In this method agents implement different metaheuristics to perform the search process. Agents asynchronously exchange partial configurations; which are analyzed by machine learning techniques in order to identify patterns and to adapt the agent behavior. The experimental evaluation runs on a Linux cluster composed of eight nodes, solving three different combinatorial optimization problems: the permutation flow-shop scheduling, the capacitated vehicle-routing and the nurse-rostering problems. The results show good performance in terms of solution quality, using up to 16 cores. The referenced paper does not present information about execution times or parallel scalability.

6.3 Framework Approaches

In this last group we analyze cooperative methods that propose a general framework. These methods generally offer high flexibility because they can tackle different problems using different metaheuristic solvers.

Cahon, Melab and Talbi in 2004 proposed an open-source framework for parallel and distributed design of hybrid metaheuristics, ParadisEO [27]. This framework provides different hybridization mechanisms for metaheuristics including population-based and single-solution methods. ParadisEO separates the modeling of the metaheuristic formulation from the problem to be solved, using a modular architecture that allows code and design reuse. For instance, ParadisEO-MO [64] is the module dedicated to the design, analysis and implementation of local search algorithms and the ParadisEO-PEO module provides a set of classes to design and implement parallel and distributed metaheuristics. ParadisEO-PEO supports different levels of parallel metaheuristics, from neighborhood decomposition (single-walk) to independent and cooperative multiple-walk. Cooperation is implemented following the island model (from population-based methods), in which the solver instances can share information based on a migration model. ParadisEO has been successfully experimented with in a wide range of problems; for instance in [108], the ParadisEO

framework is used to solve the multi-objective constrained combinatorial optimization model for a problem in radio network design.

A cooperative parallel hyper-heuristic framework was proposed in 2010 by Ouelhadj and Petrovic [94]. This framework is composed of multiple heuristic agents and one cooperative hyper-heuristic agent. Heuristic agents implement low-level heuristics performing a local search procedure. The best configuration found by the heuristic agents is sent to the cooperative hyper-heuristic agent which maintains a pool of elite configurations. This pool also stores information about low-level heuristics and the objective function. Additionally, the cooperative hyper-heuristic agent decides which low-level heuristic the heuristic agents will run and also provides them with elite configurations from the pool to diversify the search. This method clearly provides high flexibility, because it can be adapted to different problems or metaheuristics. Additionally some parameters were defined to adapt the cooperative mechanism. A prototype implementation to solve the flow shop scheduling problem is presented using C# and multi-thread libraries. The experiments run on an Intel Pentium M 1500 MHz processor obtaining good performance in terms of solution quality, however this cooperative approach does not outperform the state-of-the-art methods for the flow shop scheduling problem.

In 2014, Munera et al. [91] presented a Cooperative Parallel Local Search Framework (CPLS). This framework is both problem- and metaheuristic-independent and allows the programmer to tune the search process through an extensive set of parameters. The basic component of CPLS is an *explorer*, which executes an LS solver instance and runs on a physical core (see Figure 6). Several explorers are grouped into *teams*. Inside a team the explorers intensify the search, sharing the most promising solutions via an *elite pool*. The teams also communicate with one another to promote search diversification; for this a measure of the distance between teams is used to detect when two teams are exploring the same region (in which case a corrective action is taken to force one team to explore another region). Thus intra-team communication is used for intensification while inter-team communication ensures diversification.

The concepts and entities involved are all subject to parametric control (e.g., trade-off between intensification and diversification, elite pool size, communication interval, distance, corrective action, etc.). An implementation of CPLS (available as an open source library) in the X10 parallel programming language [104] has been used to solve different hard Combinatorial Optimization Problems [93], providing (super-)linear speedups up to 128 cores.

7 Parallelism at Work

In this section we discuss the efficacy of parallel local search methods on two hard problems, both of which have several real-world application instances: the Stable Matching (SM) and Quadratic Assignment (QAP) Problems.

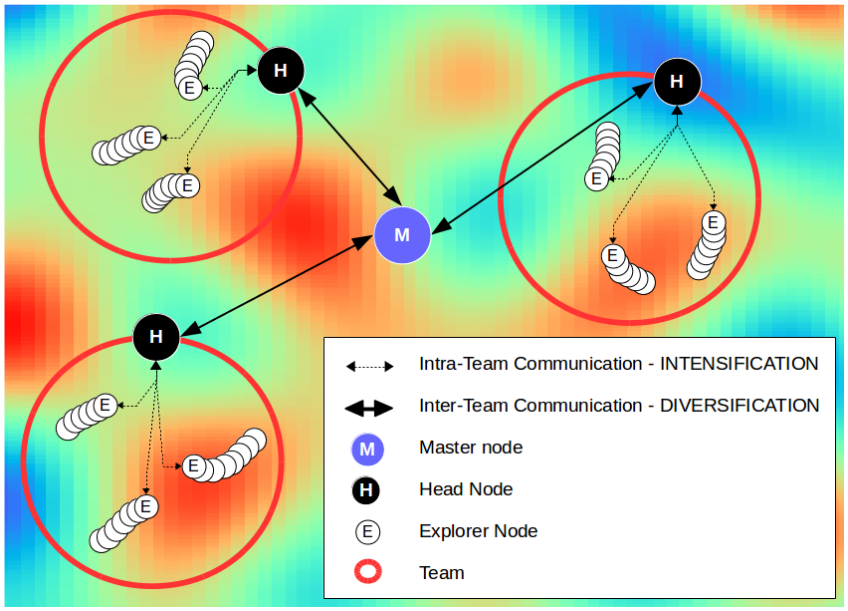

Fig. 6 CPLS framework

7.1 Stable Matching Problem

The Stable Matching problem was introduced by Gale and Shapley in their seminal 1962 paper [46]. The SM problem can be stated as follows: given a set of n men and a set of n women, each of whom have ranked all members of the other set in a strict order of preference, find a *matching* (a one-to-one correspondence between the men and the women) such that there is no man-woman pair where both prefer each other than their assigned partner. This criterion is called *stability* and is a desirable property since it ensures, according to stated preferences, that there is no man-woman pair for which both have incentive to elope – such a pair is called a *blocking pair*. Gale and Shapley proved that such a stable matching always exists and proposed an $O(n^2)$ algorithm (called GS in what follows) to find one.

However, requiring *each* member to rank *all* members of the opposite sex in a *strict* order is unfeasible for many real-life, large-scale applications. A natural variant of SM is the *Stable Matching with Ties and Incomplete Lists* (SMTI) problem [67, 78]. In SMTI, the preference lists may include ties (to express indifference among several partners) and may be incomplete (to express that some partners are unacceptable). A stable matching always exists for SMTI and can be easily obtained by arbitrarily breaking the ties and applying the GS algorithm. However, with the introduction of ties and incompleteness in the preference lists, the stable matching for an instance of SMTI may have different sizes. It is thus desirable to find the stable matching of *maximal* size (that is, with the smallest number of singles).

This optimization problem has been shown to be NP-hard, even for very restricted cases [67, 78]. This problem has attracted a lot of research in recent years since it is at the heart of a wide variety of important real-life applications. Indeed, matching problems can be found in several settings, such as car sharing or bipartite market sharing, job markets and social networks. Many of these applications involve very large sets, thereby ruling out the use of complete methods. SMTI has been shown to be an APX-hard problem [58] and most recent research focuses on designing efficient *approximation algorithms*, i.e., algorithms running in polynomial time yet able to guarantee solutions within a constant factor of the optimum [66, 68]. SMTI cannot be approximated within a factor of 21/19 and probably not within a factor of 4/3 either [59]. Currently, the best known algorithms are 3/2-approximations [80, 69, 96] or heuristic-based specific solutions. These algorithms produce a single solution for a given problem instance, even though it is often useful to provide multiple optimal or quasi-optimal solutions.

In [93], the authors proposed AS-SMTI, a local search procedure for SMTI based on adaptive search in the CPLS framework briefly described in Section 6.3. The sequential version displays significant improvement in performance or solution quality w.r.t. the state-of-the-art exact and approximate sequential algorithms, and the independent multi-walk parallel version exhibits a significant speedup with an increasing number of cores. Moreover, the cooperative parallel version achieves super-linear speedup on average, consistently behaving very well on hard instances.

The parallel experiments were carried out on a cluster of 16 machines, each with four 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) and the experiment involved up to 128 cores (four nodes and 32 cores per node). Figure 7 presents log-log graphs of the speedup using independent walks (IW in red) and cooperative walks (CW in green) on 10 very hard and large instances (size $n = 1\,000$). The independent version reaches a quasi-linear speedup (91.5 for 128 cores) while the cooperative version gets super-linear speedups (492 with 128 cores).

In [92] the same authors propose an extension of their algorithm to tackle one important and hard variant of the Stable Matching problem: the Hospital/Resident problem, which is NP-hard. This problem consists of a set of n_1 residents who apply for k positions distributed among n_2 hospitals. The preference list of a resident consists of the ordered list of *acceptable* hospitals. The preference list of a hospital contains the ordered list of residents who apply to it. In the most general case, preference lists are allowed to contain ties (to express indifference) and can be incomplete (residents only apply to a subset of the hospitals and hospitals rank their corresponding candidates). In addition, each hospital has a *capacity*, which indicates the maximum number of positions it offers. The problem consists in finding a (maximum size) stable matching between residents and hospitals (thus satisfying the preference lists) that complies with the capacities (each resident being assigned to at most one hospital and the number of residents assigned to any hospital not exceeding its capacity). The HRT problem is important in the medical domain and there are national programs in various countries, the best-known ones being the National Resident Matching Program (NRMP) in the USA, the Canadian Resident Match-

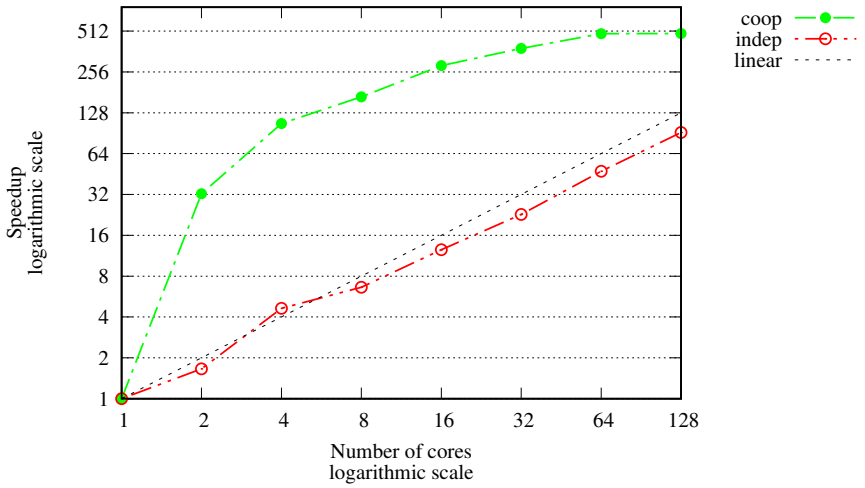

Fig. 7 Speedups obtained with AS-SMTI on hard instances of SMTI problems (size = 1 000)

ing Service (CARMS), the Scottish Foundation Allocation Scheme (SFAS) and the Japan Residency Matching Program (JRMP). As might be expected, such programs involve very large data sets. The HRT problem also has several other application domains, e.g., assignment of applicants to positions in job markets.

The resulting cooperative parallel solver, while much simpler and more general, displays performance which is comparable to the best known specific solvers for HRT, including those which assume domain restrictions (e.g., having ties on one side only).

7.2 The Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) was introduced in 1957 by Koopmans and Beckmann [71] as a model for a facilities location problem. This problem consists in assigning a set of n facilities to a set of n specific locations so as to minimize 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 and data center network topology, to cite but a few [32, 18]. This problem is known to be NP-hard and finding effective algorithms to solve it has attracted a lot of attention for many years.

Since the mid-1980s several metaheuristics have been successfully applied to the QAP: tabu search, simulated annealing, genetic algorithms, GRASP and ant-colonies [18]. For solving the hardest instances, the current trend is to resort to hybrid procedures, in order to benefit from the strengths of different classes of heuris-

tics. Such is the case of hybrid genetic algorithms for the Quadratic Assignment Problem (a.k.a memetic algorithms) [42]. The price to pay for this improvement is a significant increase in the complexity of the resulting solver code.

An alternative approach for constructing hybrid search methods has been presented in [89, 88], based on cooperative parallelism. The authors show additional benefits of the intra/inter-team cooperation mechanisms in order to provide hybridization behaviors. To this end, CPLS was configured with explorers running instances of *different* metaheuristics inside a team. Hybridization is obtained thanks to the collaboration between explorers through the *elite pool*. It turns out that the intra-team communication mechanism, implemented to intensify the search within a team, now also becomes a mechanism to exchange information between explorers running different metaheuristics. The whole system behaves like a hybrid solver, benefiting from cross-fertilization, which stems from the inherent diversity of the search strategies. The basic idea of running in parallel different metaheuristics that exchange elite solutions has been mentioned [7, 111] but from a general and strictly theoretical point of view. This technique may also be viewed as a *portfolio* approach [55] augmented with cooperation.

Following this line, the authors propose a parallel hybrid solver (called ParEOTS) to tackle the Quadratic Assignment Problem (QAP), combining two different metaheuristics: Taillard's Robust Tabu Search [107] and an original Extremal Optimization method [90]. This parallel hybrid solver performs very well on QAPLIB, the standard benchmark library used to assess QAP solvers [25]. For instance, linear speedups up to 128 cores can be achieved, see Figure 8.

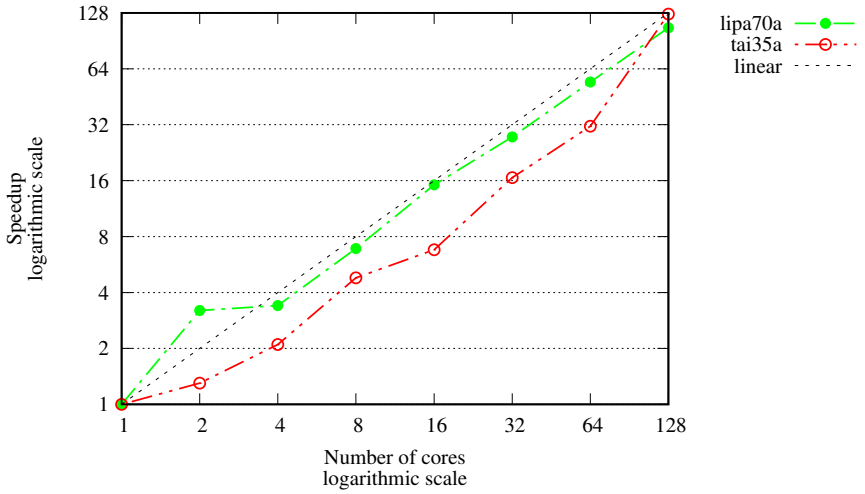

Fig. 8 Speedups obtained with ParEOTS on two QAPLIB instances

ParEOTS has been tested on the 33 hardest problems of QAPLIB, using 128 cores. The solver was set to stop when reaching the *Best Known Solutions* (BKS,

i.e., best known optimum) as recorded in the QAPLIB archive. A (comparatively short) timeout of 5 minutes was used to limit the execution in case the BKS is not reached. Each instance was solved 10 times (results are averaged). Table 1 shows the performance of ParEOTS. For each problem, the table includes the current BKS (which was sometimes the optimum), the number of times the BKS was reached by the solver (#BKS), the Average Percentage Deviation (APD), which is the average of the 10 relative deviation *percentages* computed as follows: $100 \times \frac{F(sol)-BKS}{BKS}$, and the average execution time (either shown as a decimal number representing seconds or in a human-readable form as mm:ss). Even with a very short timeout, ParEOTS provided solutions of high quality. It reached the best known solution (BKS) for all but four QAPLIB instances. When the BKS was not reached, the obtained solution was nevertheless very close (less than 0.22% off, on average).

	BKS	#BKS	APD	time
els19	17212548	**10**	0.000	0.0
kra30a	88900	**10**	0.000	0.0
sko56	34458	**10**	0.000	1.5
sko64	48498	**10**	0.000	1.7
sko72	66256	**10**	0.000	8.7
sko81	90998	**10**	0.000	0:24
sko90	115534	**10**	0.000	1:32
sko100a	152002	**10**	0.000	1:09
sko100b	153890	**10**	0.000	0:45
sko100c	147862	**10**	0.000	0:56
sko100d	149576	**10**	0.000	1:03
sko100e	149150	**10**	0.000	0:47
sko100f	149036	**10**	0.000	0:57
tai40a	3139370	**10**	0.000	1:26
tai50a	4938796	**3**	0.077	4:24
tai60a	7205962	**3**	0.146	4:15
tai80a	13499184	0	0.364	5:00
tai100a	21052466	0	0.298	5:00
tai20b	122455319	**10**	0.000	0.0
tai25b	344355646	**10**	0.000	0.0
tai30b	637117113	**10**	0.000	0.1
tai35b	283315445	**10**	0.000	0.3
tai40b	637250948	**10**	0.000	0.1
tai50b	458821517	**10**	0.000	2.6
tai60b	608215054	**10**	0.000	4.6
tai80b	818415043	**10**	0.000	0:53
tai100b	1185996137	**10**	0.000	1:11
tai150b	498896643	0	0.061	5:00
tai64c	1855928	**10**	0.000	0.0
tai256c	44759294	0	0.178	5:00
tho40	240516	**10**	0.000	0.5
tho150	8133398	**1**	0.007	4:51
wil100	273038	**10**	0.000	1:37

Table 1 ParEOTS on the hardest instances of QAPLIB (128 cores)

This solver was also tested on even harder QAP instances from Palubeckis [95] and Drezner [41], which were designed with a known optimum but were specifically ill-conditioned in order to be difficult for many metaheuristic-based methods. Recently Carvalho & Rahmann proposed new instances, with unknown optimum, that turn out to be extremely difficult to solve [39]. For the former two classes of problems (called `palu`*XX* and `dre`*XX*) the solver was configured to reach the optimum (within a timeout of 5 minutes). For the latter (called `cr-bl`*XX* and `cr-ci`*XX*) it was configured to stop as soon as the BKS was improved (with a timeout of 6 hours). ParEOTS was able to improve the quality of several solutions. Table 2 summarizes the new solutions discovered by ParEOTS for these hard problems, using 128 cores.

	OPT	previous	ParEOTS		
		BKS	#OPT	new BKS	time
palu30	271092	272080	**10**	**271092**	0.1
palu40	837900	840308	**10**	**837900**	4.0
palu50	1840356	1846876	**10**	**1840356**	0:17
palu60	2967464	2978216	**10**	**2967464**	1:07
palu70	5815290	5831954	**10**	**5815290**	2:07
palu80	6597966	6618290	**10**	**6597966**	1:56
palu100	15008994	15047406	**1**	**15008994**	5:00
palu150	58352664	58468204	0	**58414888**	5:00
palu200	75405684	75543960	0	**75498892**	5:00
dre90	1838	1959	**9**	**1838**	2:47
dre110	2264	2479	**6**	**2264**	3:43
dre132	2744	3023	**1**	**2744**	4:54
cr-bl81	-	7536	-	**7532**	48:41
cr-bl100	-	9272	-	**9264**	41:33
cr-bl121	-	11412	-	**11400**	1:05:10
cr-bl144	-	13472	-	**13452**	5:32:03
cr-ci144	-	795009899	-	**794811636**	2:29:27

Table 2 new solutions found by ParEOTS on other hard problems (128 cores)

It becomes clear from these examples that cooperative parallel hybridization for different metaheuristics can attain very competitive results and, in some cases, sometimes achieves a clear improvement.

8 Conclusion

In this chapter we have tried to present a survey of parallel local search methods over the last 20 years. Although local search methods have been pioneered since the late 1950s, parallelism has only been investigated in the context of local search methods since the 1990s, when multiprocessors started to become more widely available, and this endeavor continued until the present with experiments on massively parallel

supercomputers. Local search exhibits some natural opportunities for parallelism, which may be easily derived from the basic features of the search methods such as the selection of a new candidate solution within a neighborhood or the choice of an initial (random) starting solution. This observation prompted the adoption of some basic parallel schemes, such as single-walk and independent multi-walk methods, which can be effective on small-scale multiprocessor machines (e.g., with a few tens of cores). However, in order to achieve better performance on massively parallel machines, more complex schemes have to be devised, for instance cooperative multi-walks in which concurrent processes exchange information about their current search and communicate so as to guide the search towards promising areas of the search space. If information exchange and cooperation can be implemented efficiently and become effective enough to actually lead processes to parts of the search space where optimal or quasi-optimal solutions are, one may assert that cooperative strategies are instrumental in tapping the performance potential held in massively parallel computer architectures.

Encouraging results have already been achieved, e.g., super-linear speedups have been demonstrated on a few hard optimization problems, but more work is needed to develop general and efficient frameworks. Key issues to be investigated, especially in the context of the massively parallel machines with tens or hundreds of thousands of cores that are now available, are the flexibility and dynamicity of the system architecture, the scale and frequency of the communication between processes, and the nature of the information that should be exchanged.

Most, if not all, solvers that are mentioned in this text require non-trivial parameter tuning in order to attain their optimum performance. This task has been clearly identified and is the object of a significant and continued research effort, often resorting to different problem-solving techniques, such as machine learning.

Lastly, the hybrid nature of modern parallel multiprocessors poses several challenges concerning their effective use, as a significant portion of the available compute power stems from nonstandard architectures, such as GPUs or other accelerators. Making use of these multiple forms of parallelism is a high-stakes challenge, but one for which local search techniques could be a very good fit.

References

1. Emile Aarts and Jan K Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.
2. Renata Aiex, S. Binato, and Mauricio Resende. Parallel GRASP with Path-Relinking for Job Shop Scheduling. *Parallel Computing*, 29:393–430, 2003.
3. Renata Aiex, Simone Martins, Celso Ribeiro, and Noemi De R. Rodriguez. Cooperative Multi-thread Parallel Tabu Search with an Application to Circuit Partitioning. *Lecture Notes in Computer Science Volume 1457*, 1457:310–331, 1998.
4. Renata Aiex, Mauricio Resende, and Celso Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8(3):343–373, 2002.
5. Renata Aiex, Mauricio Resende, and Celso Ribeiro. TTT plots: a Perl program to create time-to-target plots. *Optimization Letters*, 1:355–366, 2007.

6. Enrique Alba. Special Issue on New Advances on Parallel Meta-Heuristics for Complex Problems. *Journal of Heuristics*, 10(3):239–380, 2004.
7. Enrique Alba. *Parallel Metaheuristics: a New Class of Algorithms*. Wiley-Interscience, 2005.
8. Gene M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *AFIPS Spring Joint Computer Conference, 1967. AFIPS '67 (Spring). Proceedings of the*, 30:483–485, 1967.
9. Alejandro Arbelaez and Philippe Codognet. A GPU Implementation of Parallel Constraint-Based Local Search. In *2014 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, volume 1, pages 648–655, Turin, Italy, 2014.
10. M. G. Arenas, Pierre Collet, A. E. Eiben, Márk Jelasity, J. J. Merelo, Ben Paechter, Mike Preuß, and Marc Schoenauer. A Framework for Distributed Evolutionary Algorithms. In *Parallel Problem Solving from Nature PPSN VII*, pages 665–675. Springer US, 2002.
11. Mehmet E. Aydin. Metaheuristic Agent Teams for Job Shop Scheduling Problems. *Holonic and Multi-Agent Systems for Manufacturing*, 4659:185–194, 2007.
12. László Babai. Monte-carlo algorithms in graph isomorphism testing. Research Report D.M.S. No. 79-10, Université de Montréal, 1979.
13. Per Bak. *How Nature Works: The Science of Self-organized Criticality*. Copernicus (Springer), 1st edition, 1996.
14. Per Bak and Kim Sneppen. Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters*, 71(24):4083–4086, 1993.
15. Per Bak, Chao Tang, and Kurt Wiesenfeld. Self-organized criticality: An explanation of the 1/f noise. *Physical Review Letters*, 59(4):381–384, 1987.
16. Dariusz Barbucha. A Cooperative Population Learning Algorithm for Vehicle Routing Problem with Time Windows. *Neurocomputing*, 146:210–229, 2014.
17. Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.
18. Ravi Kumar Bhati and Akhtar Rasool. Quadratic Assignment Problem and its Relevance to the Real World: A Survey. *International Journal of Computer Applications*, 96(9):42–47, 2014.
19. Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
20. Stefan Boettcher. Extremal Optimization. In Alexander K. Hartmann and Heiko Rieger, editors, *New Optimization Algorithms to Physics*, chapter 11, pages 227–251. Wiley-VCH Verlag, Berlin, 2004.
21. Stefan Boettcher and Allon Percus. Nature's way of optimizing. *Artificial Intelligence*, 119(12):275–286, 2000.
22. Stefan Boettcher and Allon Percus. Extremal Optimization: an Evolutionary Local-Search Algorithm. In *Computational Modeling and Problem Solving in the Networked World*, volume 21. Springer US, 2003.
23. A. Bortfeldt, H. Gehring, and D. Mack. A Parallel Tabu Search Algorithm for Solving the Container Loading Problem. *Parallel Computing*, 29(5 SPEC.):641–662, 2003.
24. Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A Survey on Optimization Metaheuristics. *Information Sciences*, 237(February):82–117, 2013.
25. Rainer E. Burkard, S. Karisch, and F. Rendl. QAPLIB - a Quadratic Assignment Problem Library. *European Journal of Operational Research*, 55(1):115–119, 1991.
26. J. M. Cadenas, M. C. Garrido, and E. Muñoz. Using Machine Learning in a Cooperative Hybrid Parallel Strategy of Metaheuristics. *Information Sciences*, 179(19):3255–3267, 2009.
27. S. Cahon, N. Melab, and E. G. Talbi. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.
28. Yves Caniou, Philippe Codognet, Daniel Diaz, and Salvador Abreu. Experiments in parallel constraint-based local search. In *EvoCOP'11, 11th European Conference on Evolutionary Computation in Combinatorial Optimisation*, volume 6622 of *Lecture Notes in Computer Science*, Torino, Italy, 2011. Springer Verlag.

29. Yves Caniou, Philippe Codognet, Florian Richoux, Daniel Diaz, and Salvador Abreu. Large-scale Parallelism for Constraint-Based Local Search: the Costas Array Case Study. *Constraints*, 20(1):30–56, 2015.
30. Yves Caniou, Daniel Diaz, Florian Richoux, Philippe Codognet, and Salvador Abreu. Performance Analysis of Parallel Constraint-Based Local Search. In *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, PPoPP '12, New York, NY, USA, 2012. ACM. poster paper.
31. Philippe Codognet and Daniel Diaz. Yet Another Local Search Method for Constraint Solving. In Kathleen Steinhöfel, editor, *Stochastic Algorithms: Foundations and Applications*, pages 342–344. Springer Berlin Heidelberg, London, 2001.
32. Clayton Warren Commander. A survey of the quadratic assignment problem, with applications. *Morehead Electronic Journal of Applicable Mathematics*, 4:MATH–2005–01, 2005.
33. Jean-François F Cordeau and Mirko Maischberger. A Parallel Iterated Tabu Search Heuristic for Vehicle Routing Problems. *Computers and Operations Research*, 39(9):2033–2050, 2012.
34. Teodor Crainic, Michel Gendreau, Pierre Hansen, and Nenad Mladenovic. Cooperative Parallel Variable Neighborhood Search for the p-Median. *Journal of Heuristics*, 10(3):293–314, 2004.
35. Teodor Crainic and Michel Toulouse. Special Issue on Parallel Meta-Heuristics. *Journal of Heuristics*, 8(3):247–388, 2002.
36. G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.
37. Leonardo Dagum and Ramesh Menon. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.
38. H.A. David and H.N. Nagaraja. *Order Statistics*. Wiley series in probability and mathematical statistics. Probability and mathematical statistics. John Wiley, 2003.
39. Sérgio A de Carvalho Jr. and Sven Rahmann. Microarray layout as a quadratic assignment problem. In *German Conference on Bioinformatics (GCB)*, volume 83, pages 11–20, Tübingen, Germany, 2006.
40. Daniel Diaz, Florian Richoux, Philippe Codognet, Yves Caniou, and Salvador Abreu. Constraint-based Local Search for the Costas Array Problem. In *LION 6, Learning and Intelligent OptimizatioN Conference*, Paris, France, 2012. Springer LNCS.
41. Zvi Drezner. The Extended Concentric Tabu for the Quadratic Assignment Problem. *European Journal of Operational Research*, 160(2):416–422, 2005.
42. Zvi Drezner. Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. *Computers & Operations Research*, 35(3):717–736, 2008.
43. Huub M. M. Eikelder, Bas J. M. Aarts, Marco G. A. Verhoeven, and Emile H. L. Aarts. Sequential and Parallel Local Search Algorithms for Job Shop Scheduling. In *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 359–371. Springer US, Boston, MA, 1999.
44. Merrill M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.
45. Edgar Gabriel and al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, 2004.
46. D Gale and L Shapley. College Admissions and the Stability of Marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.
47. Bruno-Laurent Garcia, Jean-Yves Potvin, and Jean-Marc Rousseau. A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints. *Computers & Operations Research*, 21(9):1025–1033, 1994.
48. F. García-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. The Parallel Variable Neighborhood Search for the p -Median Problem. *Journal of Heuristics*, 8(3):375–388, 2002.
49. Frédéric Gardi and Karim Nouioua. Local search for mixed-integer nonlinear optimization: A methodology and an application. In *Evolutionary Computation in Combinatorial*

Optimization - 11th European Conference, EvoCOP 2011, Torino, Italy, April 27-29, 2011. Proceedings, pages 167–178, 2011.
50. M. Gendreau, F. Guertin, J.-Y. Potvin, and E. Taillard. Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching. *Transportation Science*, 33(4):381–390, 1999.
51. Ian Gent and Toby Walsh. CSPLib: A Benchmark Library for Constraints. Technical report, Berlin, Heidelberg, 1999.
52. Fred Glover. Tabu Search–Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
53. Fred Glover. Tabu Search–Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
54. Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, jul 1997.
55. Carla Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.
56. Teofilo Gonzalez, editor. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall / CRC, 2007.
57. F. Guerriero and M. Mancini. A Cooperative Parallel Rollout Algorithm for the Sequential Ordering Problem. *Parallel Computing*, 29:663–677, 2003.
58. Magnus Halldorsson, Robert Irving, Kazuo Iwama, David Manlove, Shuichi Miyazaki, Yasufumi Morita, and Sandy Scott. Approximability Results for Stable Marriage Problems with Ties. *Theoretical Computer Science*, 306(1-5):431–447, 2003.
59. Magnus Halldorsson, Kazuo Iwama, Shuichi Miyazaki, and Hiroki Yanagisawa. Improved Approximation of the Stable Marriage Problem. *ACM Transactions on Algorithms*, 3(3):266–277, 2007.
60. Tad Hogg and Colin P Williams. Solving the Really Hard Problems with Cooperative Search. In *AAAI Conference on Artificial Intelligence (AAAI-93)*, pages 231–236, 1993.
61. Holger Hoos and Thomas Stützle. Evaluating Las Vegas algorithms: Pitfalls and remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, UAI'98*, pages 238–245. Morgan Kaufmann, 1998.
62. Holger Hoos and Thomas Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for sat. *Artificial Intelligence*, 112(1-2):213–232, 1999.
63. Holger Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.
64. J. Humeau, A. Liefooghe, E. G. Talbi, and S. Verel. ParadisEO-MO: From Fitness Landscape Analysis to Efficient Local Search Algorithms. Technical report, INRIA, 2013.
65. T. Ibaraki, K. Nonobe, and M. Yagiura, editors. *Metaheuristics: Progress as Real Problem Solvers*. Springer Verlag, 2005.
66. Robert Irving and David Manlove. Approximation Algorithms for Hard Variants of the Stable Marriage and Hospitals/Residents Problems. *Journal of Combinatorial Optimization*, 16(3):279–292, 2008.
67. Kazuo Iwama, David Manlove, Shuichi Miyazaki, and Yasufumi Morita. Stable Marriage with Incomplete Lists and Ties. In *In Proceedings of ICALP '99: the 26th International Colloquium on Automata, Languages and Programming*, number ii, pages 443–452. Springer-Verlag, 1999.
68. Zoltán Király. Approximation of Maximum Stable Marriage. Technical report, Egervary Research Group, Budapest, Hungary, 2011.
69. Zoltán Király. Linear Time Local Approximation Algorithm for Maximum Stable Marriage. *Algorithms*, 6(3):471—-484, aug 2013.
70. S Kirkpatrick, CD Gelatt Jr, and MP Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.
71. Tjalling C Koopmans and Martin Beckmann. Assignment Problems and the Location of Economic Activities. *Econometrica*, 25(1):53–76, 1957.
72. S Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.
73. Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated Local Search. In *Handbook of Metaheuristics*, pages 320–353. Kluwer Academic Publishers, Boston, 2003.

74. Thé Van Luong, Nouredine Melab, and El-Ghazali Talbi. Local Search Algorithms on Graphics Processing Unit. A Case Study: The Permutation Perceptron Problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 264–275. LNCS 6022, Springer Verlag, 2010.
75. Thé Van Luong, Nouredine Melab, and El-Ghazali Talbi. GPU Computing for Parallel Local Search Metaheuristics. *IEEE Transactions on Computers*, 62(1):173–185, 2013.
76. Rui Machado, Salvador Abreu, and Daniel Diaz. Parallel Local Search: Experiments with a PGAS-based programming model. In *12th International Colloquium on Implementation of Constraint and Logic Programming Systems*, pages 1–17, Budapest, Hungary, 2012.
77. Rui Machado, Salvador Abreu, and Daniel Diaz. Parallel Performance of Declarative Programming Using a PGAS Model. In Kostis Sagonas and Gopal Gupta, editors, *Practical Aspects of Declarative Languages, PADL'2013*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2013.
78. David Manlove, Robert Irving, Kazuo Iwama, Shuichi Miyazaki, and Yasufumi Morita. Hard Variants of Stable Marriage. *Theoretical Computer Science*, 276(1-2):261–279, apr 2002.
79. Simon Martin, Djamila Ouelhadj, Patrick Beullens, Ender Ozcan, Angel A. Juan, and Edmund K. Burke. A Multi-Agent Based Cooperative Approach to Scheduling and Routing. *European Journal of Operational Research*, In press:1–26, mar 2016.
80. Eric McDermid. A 3/2-Approximation Algorithm for General Stable Marriage. In *International Colloquium on Automata, Languages and Programming, ICALP'2009*, pages 689–700, Rhodes, Greece, 2009.
81. David Meignan, Abderrafiaa Koukam, and Jean Charles Créput. Coalition-based metaheuristic: A self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879, 2010.
82. Nouredine Melab, Thé Van Luong, Karima Boufaras, and El-Ghazali Talbi. ParadisEO-MO-GPU: A Framework for Parallel GPU-Based Local Search Metaheuristics. In *15th annual conference on Genetic and evolutionary computation conference GECCO '13*, pages 1189–1196, Amsterdam, The Netherlands, 2013.
83. Laurent Michel, Andrew See, and Pascal Van Hentenryck. Distributed constraint-based local search. In Frédéric Benhamou, editor, *CP'06, 12th Int. Conf. on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 344–358. Springer Verlag, 2006.
84. Laurent Michel, Andrew See, and Pascal Van Hentenryck. Parallel and Distributed Local Search in Comet. *Computers and Operations Research*, 36:2357–2375, 2009.
85. Michela Milano and Andrea Roli. MAGMA: A Multiagent Architecture for Metaheuristics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(2):925–941, 2004.
86. Nenad Mladenovic and Pierre Hansen. Variable Neighborhood Search. *Computers & Operations Research*, 24(11):1097–1100, 1997.
87. Hiroyuki Mori and Yoshihiro Ogita. A Parallel Tabu Search Based Method for Reconfigurations of Distribution Systems. In *2000 Power Engineering Society Summer Meeting (Cat. No.00CH37134)*, volume 1, pages 73–78. IEEE, 2000.
88. Danny Munera. *Solving Hard Combinatorial Optimization Problems using Cooperative Parallel Metaheuristics*. PhD Thesis, University Paris 1 Pantheon-Sorbonne, 2016.
89. Danny Munera, Daniel Diaz, and Salvador Abreu. Hybridization as Cooperative Parallelism for the Quadratic Assignment Problem. In *10th International Workshop, HM 2016*, volume 9668 of *Lecture Notes in Computer Science*, pages 47–61, Plymouth, UK, 2016. Springer International Publishing.
90. Danny Munera, Daniel Diaz, and Salvador Abreu. Solving the Quadratic Assignment Problem with Cooperative Parallel Extremal Optimization. In *The 16th European Conference on Evolutionary Computation in Combinatorial Optimisation*, Porto, 2016.
91. Danny Munera, Daniel Diaz, Salvador Abreu, and Philippe Codognet. A Parametric Framework for Cooperative Parallel Local Search. In Christian Blum and Gabriela Ochoa, editors, *European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP)*, volume 8600 of *Lecture Notes in Computer Science*, pages 13–24, Granada, Spain, 2014. Springer.

92. Danny Munera, Daniel Diaz, Salvador Abreu, Francesca Rossi, Vijay Saraswat, and Philippe Codognet. A Local Search Algorithm for SMTI and its extension to HRT Problems. In *3rd International Workshop on Matching Under Preferences*, Glasgow, UK, 2015.
93. Danny Munera, Daniel Diaz, Salvador Abreu, Francesca Rossi, Vijay Saraswat, and Philippe Codognet. Solving Hard Stable Matching Problems via Local Search and Cooperative Parallelization. In *AAAI*, Austin, TX, USA, 2015.
94. Djamila Ouelhadj and Sanja Petrovic. A Cooperative Hyper-heuristic Search Framework. *Journal of Heuristics*, 16(6):835–857, 2010.
95. Gintaras Palubeckis. An Algorithm for Construction of Test Cases for the Quadratic Assignment Problem. *Informatica, Lith. Acad. Sci.*, 11(3):281–296, 2000.
96. Katarzyna Paluch. Faster and Simpler Approximation of Stable Matchings. *Algorithms*, 7(2):176–187, nov 2014.
97. Panos M. Pardalos, Leonidas S. Pitsoulis, Thelma D. Mavridou, and Mauricio G. C. Resende. Parallel search for combinatorial optimization: Genetic algorithms, simulated annealing, tabu search and GRASP. In *Parallel Algorithms for Irregularly Structured Problems (IRREGULAR)*, pages 317–331, 1995.
98. J. Antonio Parejo, Antonio Ruiz-Cortés, Sebastián Lozano, and Pablo Fernandez. Metaheuristic Optimization Frameworks: a Survey and Benchmarking. *Soft Computing*, 16(3):527–561, 2012.
99. International Exascale Software Project. Exascale roadmap 1.0. Technical report, 2009. http://www.exascale.org/iesp/IESP:Documents.
100. César Rego and Catherine Roucairol. A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem. In *Meta-Heuristics*, pages 661–675. Springer US, Boston, MA, 1996.
101. Gerhard Reinelt. TSPLIB–A traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.
102. Celso Ribeiro and Isabel Rosseti. Efficient Parallel Cooperative Implementations of GRASP Heuristics. *Parallel Computing*, 33(1):21–35, 2007.
103. Celso Ribeiro, Isabel Rosseti, and Reinaldo Vallejos. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, 54:405–429, 2012.
104. Vijay Saraswat, Bard Bloom, Igor Peshansky, Olivier Tardieu, and David Grove. X10 Language Specification - Version 2.3. Technical report, IBM Research, 2012.
105. Kenneth Sörensen and Fred Glover. Metaheuristics. In *Encyclopedia of Operations Research and Management Science*, pages 960–970. Springer US, Boston, MA, 2013.
106. Kenneth Sörensen, Marc Sevaux, and Fred Glover. A History of Metaheuristics. In Rafael Marti, Panos Pardalos, and Mauricio Resende, editors, *Handbook of Heuristics*. Springer US, Boston, MA, 2016.
107. Éric Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel computing*, 17(4-5):443–455, 1991.
108. E. G. Talbi, S. Cahon, and N. Melab. Designing Cellular Networks Using a Parallel Hybrid Metaheuristic on the Computational Grid. *Computer Communications*, 30(4):698–713, 2007.
109. El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. Wiley, 2009.
110. El-Ghazali Talbi and Vincent Bachelet. COSEARCH: A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms*, 5(1):5–22, 2006.
111. Sarosh Talukdar, Lars Baerentzen, Andrew Gove, and Pedro De Souza. Asynchronous Teams : Cooperation Schemes for Autonomous Agents. *Journal of Heuristics*, 4:295–321, 1998.
112. Michel Toulouse, Teodor Crainic, and Michel Gendreau. Communication Issues in Designing Cooperative Multi-Thread Parallel Searches. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory&Applications*, pages 501–522. Kluwer Academic Publishers, Norwell, MA., 1995.
113. Charlotte Truchet, Alejandro Arbelaez, Florian Richoux, and Philippe Codognet. Estimating parallel runtimes for randomized algorithms in constraint solving. *J. Heuristics*, 22(4):613–648, 2016.

114. Charlotte Truchet, Florian Richoux, and Philippe Codognet. Prediction of Parallel Speed-ups for Las Vegas Algorithms. In Jack Dongarra and Yves Robert, editors, *Proceedings of ICPP-2013, 42nd International Conference on Parallel Processing*. IEEE Press, October 2013.
115. Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. The MIT Press, aug 2005.
116. Marcus Verhoeven. *Parallel Local Search*. PhD thesis, University of Eindhoven, Eindhoven, Netherlands, 1996.
117. Marcus Verhoeven and Emile Aarts. Parallel Local Search. *Journal of Heuristics*, 1(1):43–65, 1995.
118. Stefan Voß. Meta-heuristics: The State of the Art. In Alexander Nareyek, editor, *Local Search for Planning and Scheduling*, pages 1–23. Springer Berlin Heidelberg, 2001.
119. M. Yazdani, M. Amiri, and M. Zandieh. Flexible Job-Shop Scheduling with Parallel Variable Neighborhood Search Algorithm. *Expert Systems with Applications*, 37(1):678–687, 2010.