International Journal of Hybrid Intelligent Systems 11 (2014) 287–302
DOI 10.3233/HIS-140200
IOS Press

Savant: Automatic generation of a parallel scheduling heuristic for map-reduce

Frédéric Pinel[a,*] and Bernabé Dorronsoro[b]
[a]*University of Luxembourg, Luxembourg, Luxembourg*
[b]*University of Lille, Lille, France*

Abstract. This paper investigates the automatic generation of a Map-Reduce program, which implements a heuristic for an NP-complete problem with machine learning. The objective is to automatically design a new concurrent algorithm that finds solutions of comparable quality to the original heuristic. Our approach, called Savant, is inspired from the savant syndrome. Its concurrency model is based on Map-Reduce. The approach is evaluated with the well-known Min-Min heuristic. Experimental results on two problem sizes are promising, the produced algorithm is able to find solutions of comparable quality.

Keywords: Pattern recognition, parallelism and concurrency, conversion from sequential to parallel forms

1. Introduction

Parallel algorithms are becoming necessary in every aspect of computing with the widespread adoption of distributed systems composed of multicore processors. Up to now, parallelism was only required in specific cases, usually for performance improvements. The default computer is becoming a parallel machine [1]. This trend is a consequence of the evolution of computer processors, where physical limits are forcing chip designers to reduce the clock frequency of processors, and packaging more of them. Current computers now come with multiple processors, which are themselves multi-core. In addition, alternative parallel co-processors are common, such as graphics processing units (GPU). Mass markets are also favoring distributed systems such as clusters, assembled from off-the-shelf components in contrast to specialized parallel hardware [2]. Finally, recent Internet trends, such as cloud computing, the ubiquity of JavaScript virtual machines and mobile devices add another level in parallelism, by massively distributing computation across cloud servers and browsers. However, the parallelism provided by the hardware requires the design of concurrent algorithms to be fully exploited. Recent algorithms introduce concurrency as much as possible, but programming languages for concurrent program are still appearing [3–6], and manually designing a concurrent program remains a difficult task. Moreover a great number of existing programs need to be adapted to the parallel architectures.

In light of this trend, we are investigating a method to automatically parallelize existing algorithms. This is of course a challenging problem. As a first step, we limit this problem's scope in several ways.

- We relax a common constraint that the parallel version must implement the original algorithm (same algorithm but a different implementation). We are searching for a different algorithm that nevertheless performs the same function.
- We cast the problem as a supervised learning problem, and leverage the efficiency of modern machine learning techniques. This is inspired by the Savant syndrome (Section 3.2), which hints to a parallel machine performing seemingly sequential tasks. By analogy, we consider that the parallel version of the original algorithm must learn the behavior of the original one.
- We evaluate our proposed approach on a specific algorithm (Min-Min), a well-known heuris-

*Corresponding author: Frédéric Pinel, University of Luxembourg, Luxembourg, Luxembourg. E-mail: frederic.pinel@uni.lu.

tic for an NP-complete scheduling problem. This use case is motivated by three factors. First, we are familiar with this problem and its state-of-the-art parallel solvers, which is useful to assess the results obtained. Second, optimization problems highlight the key point of our approach: the design of different algorithms that solve the same problem. Indeed, solutions to optimization problems are evaluated with a fitness function, which score a solution regardless of how this solution was found. This helps abandon existing algorithms, as long as the solutions are of comparable quality, and the produced design is concurrent. Finally, solving combinatorial problems is computationally intensive, and parallel heuristics is an active research area that could benefit from automatic parallelization. The ultimate intention is to evaluate this approach on other algorithms and problems.

Our contribution is a method to automatically generate a parallel version of the chosen heuristic, under the Map-Reduce architecture [7]. The method takes an initial sequential program that implements the scheduling heuristic, and derives a Map-Reduce based program that can exploit significant parallelism (hundreds of cores) even for small problem instances. The concurrent design intentionally focuses on small problems, where the source of concurrency does not lie in the data or the complexity of the algorithm, as suggested by D. Hillis [8]. This paper extends our previous work [9]: the training configuration is refined, the algorithm's capability is improved, and the results are further analyzed to better understand its behavior.

Section 2 describes the problem and reviews previous work. Section 3 presents and motivates our approach, called Savant. Section 4 reports the experimental results of the Savant approach.

2. Problem statement

In this section, we present the automatic parallelization problem, and provide the necessary information on the use case for our evaluation. Section 2.1 reviews past and current automatic parallelization efforts. Section 2.2 states the scheduling problem and the heuristic used to evaluate the approach. Section 2.3 provides background information on the scheduling problem and it's known solvers. Section 2.4 describes the heuristic we have chosen for our evaluation.

2.1. Automatic parallelization

Parallelism was initially considered a part of the automatic build of executables from source code. This optimization step is usually approached by applying source-to-source transformations [10,11]. Transformations include loop-unrolling, data access patterns, and rely on careful inspection of data dependencies to extract concurrency from the source program. This approach to parallelization preserves the algorithm and most of the source code, by applying transformations that respect the semantics of the original program. The transformations are carefully defined, so as to guarantee identical behavior, and may even rely on formal reasoning [12]. Other authors apply AI techniques to identify the transformations and their order of application. Evolutionary algorithms and machine learning were applied in [13–15].

As mentioned, our focus is not to preserve most of the source program, nor even the algorithm, but to find new algorithms and code. Genetic Programming (GP) [16] is a method to achieve such a goal. Indeed, GP aims to automatically evolve a program that displays a set of properties. Parallelism can be one of them. A combined evolutionary and source-to-source transformation technique was presented in [17]. There is little detail presented however. In [18–20], the authors use GP to evolve a program in order to achieve parallelism. The programs found are evaluated both in terms of correctness (their purpose) and their degree of parallelism. Evolving the program allows for easier evaluation of the parallelism by executing the code. We find that the programs evolved are relatively simple ($O(100)$ assembly instructions), and require considerable effort to find (the stopping condition is the absence of progress in the last $10^6 - 10^8$ evolutions). GP is a general technique which comes with its drawbacks, such as the computational effort required. Also, defining parallelism as a fitness function is an elegant formulation of the problem, but is not reliable regarding the parallelism obtained. We believe more specific, thus efficient, approaches can be used. Finally, genetic algorithms can be used to evolve rules for computation, instead of a solution to a problem [21]. Therefore, such an approach could in principle be used for automatic parallelization but we have not found previous work.

2.2. The min-min heuristic for the independent tasks mapping problem

The proposed parallelization approach is applied to the Min-Min heuristic, described in Section 2.4, which

finds solutions to the independent tasks mapping problem.

This problem assumes a set of independent computing tasks, which can be processed by a set of heterogeneous resources, or machines. Each task can only be processed by a single machine (it cannot be split across machines). A problem instance is defined by an Estimated Time to Complete (ETC) matrix, which holds the duration of each task on every possible machine. The ETC is assumed given. The ETC is randomly generated according to the procedure in [22]. In this study, the tasks and machines are considered highly heterogeneous, and the machines are consistent (a machine cannot be slower than any other for a task, and faster for another). An example ETC is:

$$\begin{bmatrix} & t_1 & t_2 & t_3 & ... \\ m_1 & 12.3 & 17.8 & 45.7 & ... \\ m_2 & 15.9 & 18.3 & 73.0 & ... \end{bmatrix},$$

where task t_1 takes 12.3 units of time to execute on machine m_1.

The optimization problem is finding the solution that minimizes *makespan*. Makespan is the time when the last task finishes, across all machines. It is computed using the ETC matrix, by summing the task's ETC on their respective mapped machine. The makespan minimization problem is NP-complete [23]. A solution to the problem is represented as an array of integers, where $solution[t] = m$ means that task t is assigned to machine m.

2.3. Independent tasks mapping problem

The approach proposed in this paper is applied to the resolution of a combinatorial problem from the scheduling domain: the independent tasks mapping problem. The complexity of this problem confines candidate solver algorithms to heuristic and metaheuristic approaches, except for very small problems. This section reviews the most relevant past work related to this problem.

One of the most relevant heuristics applied to solve this problem are the list scheduling algorithms. Because we make use of them in this paper, they are introduced in a separate section (Section 2.4). Pinel et al. propose in [24] a heuristic that works in two phases by running first Min-Min and then improving its result with a local search heuristic. In addition to these heuristics, metaheuristics have been successfully applied to this problem too. Some examples are Genetic Algorithms [25,26], Ant Colony Optimization [27], and other hybrid algorithms [27,28].

Due to the complexity of the problem and the quick response needed, a number of parallel metaheuristics have been proposed in the literature. Pinel et al. designed a multi-threaded parallel cellular genetic algorithm for the problem in [29]. Nesmachnow et al. proposed a parallel CHC algorithm in [30] and test it on a cluster of 4 quad core servers. There are also several of works proposing parallel algorithms of the considered problem for the GPU. Solomon et al. [31] presented a Particle Swarm Optimization algorithm that provide high speedups (up to 37 times faster), but the results reported are worse than those of the compared heuristics. Nesmachnow [32] proposed GPU implementations of two scheduling heuristics, reporting a maximum speedup of about 5 with respect to the sequential version of the heuristic. Finally, Pinel et al. [33] proposed two parallel designs for GPU of Min-Min and a cGA, reporting speedups of up to 538 times faster with respect to the sequential Min-Min. Additionally, Pinel et al. [9] presented a novel framework, inspired in the Savant syndrome, that automatically learns and reproduce the behavior of a target algorithm. The resulting algorithm can be executed using map-reduce framework, so it can be run on large clusters, as well as on multi-core and GPU architectures. It was applied to learn the Min-Min heuristic, and it was even able to outperform it.

The scheduling problem is inherently a multi-objective problem, since there are several conflicting objectives to take into account, as those matching the interests of the service provider (i.e., makespan, cost) and the customer (especially those related to the QoS) [34]. We discuss next the main works in the literature proposing multi-objective evolutionary algorithms (MOEAs) for the problem of independent tasks scheduling. Before, we will briefly classify the main existing techniques to solve multi-objective problems (MOPs). We first may distinguish between *hierarchical* or *simultaneous* approaches.

The former class, also called *lexicographic ordering* in the specialized literature, lies in first establishing some priority on the considered criteria. Then, these criteria are optimized in the given priority order without worsening the value of the higher priority ones. The main drawback of this technique is that a pre-defined ordering of objectives is required. This is not always easy, and it bias the search and, therefore, the obtained result and the performance of the algorithm. The alternative to lexicographic ordering is to optimize the criteria at the same time. There are several approaches for that, being the main techniques based on function ag-

gregation, ϵ-constraint, and Pareto dominance. In function aggregation technique, the multi-criteria problem is transformed into a single-objective one by aggregating the criteria into one single weighted function. This method is commonly used in the literature because of its easy implementation, but it will not work when the Pareto front is concave, regardless of the weights used [35]. Additionally, the solution found by the algorithm is obviously biased by the weights used in the function aggregation. The ϵ-constraint method is based on the optimization of one of the criteria, considered as the primary one, and redefining the other objectives as constraints bound by some allowable levels ϵ_i. Hence, the problem is reduced to a kind of constrained single-objective optimization but, in practice, it will be difficult to set accurate values for the constraints that will still allow finding feasible solutions. In the Pareto dominance technique, all the criteria are tackled as independent functions, and the algorithm is optimizing all of them at the same time. For that, the concept of *dominance* is introduced, meaning that one solution *dominates* –is *dominated* by– another if it is better –worse– or equal for all the objectives and strictly better –worse– for at least one of them. Therefore, two solutions are called *non-dominated* if none dominates the other. The objective of the Pareto dominance technique is then to find the Pareto optimal set, composed by the best non-dominated solutions to the problem. In contrast to the previously mentioned methods, the output of this technique is a set of non-dominated solutions, and not only one single solution.

In 2006, Yu et al. [36] presented a genetic algorithm (GA) to solve the problem of resource allocation by optimizing the execution time and the cost of the schedule by using some ϵ-constraint technique. The primary objective was in this case to optimize the makespan with the constraint of a maximum given budget, which is specified by the users for workflow execution. Here, makespan is optimized to benefit the service provider, but some minimum QoS level is ensured with the constraint added, thus benefiting the users. One year later, Sweeney et al. studied a similar problem that was enhanced with two more constraints: a maximum number of total cycles to run the job and a maximum execution time. In that paper, they used both a GA and a simulated annealing (SA) algorithm to solve the problem [37].

One of the most popular techniques to solve multi-objective jobs scheduling problems is functions aggregation. Works like those of Xhafa et al. [28,38,39], Zhong et al. [40], and Kromer et al. [41] are examples of the use of techniques like cellular GAs, memetic algorithms (MA), differential evolution (DE), and tabu search (TS), among others, for the optimization of an aggregation function composed by the makespan and flowtime. The considered problem in these works was the scheduling of independent tasks where makespan and flowtime are the criteria to optimize. In [42], Abraham et al. compared this approach versus the Pareto dominance technique for this problem on a GA, SA, a fuzzy particle swarm optimization (PSO) algorithm (all of them optimizing the aggregated function), and MOEA, a multi-objective evolutionary algorithm based on Pareto dominance. As a result, they found that MOEA provided excellent results, outperforming the compared algorithms.

Jakob et al. [43] studied the optimization of the weighted sum of four objectives: two of them benefiting the provider, namely makespan and resource utilization, and two more focusing on the interests of the customer, time and cost of each application. Later, they extended their work by considering the reschedule of previously scheduled non-executed tasks [44].

Nesmachnow [45] studies the performance of several state-of-the-art multi-objective evolutionary algorithms, namely MOCell, MOCHC, NSGA-II, and SPEA2, on the problem of scheduling independent tasks on Grids minimizing the total makespan and flowtime. He concluded that MOCell was the most useful tool among the compared ones for the considered problem.

In [46], the same author proposes a new technique that is hybrid between aggregation and Pareto dominance for the same problem, but minimizing makespan and the weighted response ratio (i.e., the total response ratio of each task multiplied by its weight –priority–) this time. The technique consists of structuring the population into several subpopulations, each optimizing an aggregated function of the two objectives with different weights using CHC evolutionary algorithm. Then, Pareto dominance is used to build the Pareto front from solutions in the different islands.

There are some works dealing with MO algorithms for the robust scheduling problem, like [47], which is considering some kind of robustness by optimizing, together with the resource utilization, the resources reliability by assigning some static reliability values to every resource (i.e., a value meaning how reliable is the resource) in the problem definition; or [48], directly optimizing a robustness metric together with the makespan with four state-of-the-art MO algorithms (MOCell, NSGA-II, IBEA, and MOEA/D).

Finally, there is a number of papers focusing on multi-objective versions of the problem with energy efficiency considerations. In [49], the authors proposed a number of multi-objective heuristics to solve the problem of independent tasks scheduling in multi-core heterogeneous Grid computing systems, accounting for makespan and energy consumption minimization. In that work, the algorithms exploit the heterogeneity of resources to save energy in the computation of tasks. There are other works that make use of dynamic voltage scaling techniques to save energy, subject to all tasks are executed before their specified deadline [50]. Li et al. [51] presented an online dynamic power management strategy with multiple power saving states. Then, they proposed an energy-aware scheduling algorithm to reduce energy consumption.

2.4. List scheduling heuristics

The class of *list scheduling techniques* comprises a large set of deterministic static scheduling methods that work by assigning priorities to tasks based on a particular ad-hoc heuristic [52]. After that, the list of tasks is sorted in decreasing priority and each task is assigned to a processor, regarding the task priority and the processor availability. Algorithm 1 presents the general schema of a list scheduling method.

Algorithm 1 Schema of a list scheduling algorithm.

```
1: while tasks left to assign do
2:     determine the most suitable task according to
       the chosen criterion
3:     for each task to assign, each machine do
4:         evaluate criterion (task, machine)
5:     end for
6:     assign the selected task to the selected machine
7: end while
8: return task assignment
```

The first algorithms following the generic schema presented in Algorithm 1 were introduced in the pioneering work by Ibarra and Kim [53]. Later, many list scheduling techniques have been proposed in order to provide easy methods for tasks-to-processors scheduling. This class of methods has also often been employed in hybrid algorithms, with the purpose of improving the search of metaheuristic approaches for solving scheduling problems.

The simplest category of list scheduling heuristics applied to minimize the makespan metric uses a single criterion to perform the tasks-to-machine assignment [54]. Many popular heuristics fit into this category, including *Shortest Job to Fastest Resource* and *Longest Job to Fastest Resource*, which sort the tasks by increasing/decreasing execution time (ET) and assign them to the available resources, sorted by their decreasing computing capacity; *Opportunistic Load Balancing*, which sorts the set of tasks in an arbitrary order and assigns them to the next machine that is expected to be available, regardless of the ET for each task on that machine; *Minimum Execution Time*, which sorts the set of tasks in an arbitrary order and assigns them to the machine with lower ET for that task, regardless of the machine availability; and *Minimum Completion Time* (MCT), which sorts the set of tasks in an arbitrary order and assigns each task to the machine that can complete it earlier. More details about these simple list scheduling heuristics can be found in the works by Braun et al. [25] and Freund et al. [54].

Trying to overcome the inefficacy of these simple heuristics, other list scheduling methods have been proposed taking into account more complex and holistic criteria to perform the task mapping, and then reduce the makespan values [25]. Some of the most popular heuristics in this class include:

- *Min-Min*: greedily picks the task that can be completed the soonest. The method starts with a set U of all *unmapped* tasks, calculates the MCT for each task in U for each machine, and assigns the task with the minimum overall MCT to the best machine. The mapped task is removed from U, and the process is repeated until all tasks are mapped. Min-Min does not consider a single task at a time but all the unmapped tasks sorted by MCT, and the availability status of the machines is accordingly updated after every assignment. This procedure generally leads to more balanced schedules (i.e., better makespan values) than other heuristics, since more tasks are expected to be assigned to the machines that can complete them the earliest;
- *Max-Min*: is similar to Min-Min, but it assigns the task with the overall *maximum* MCT to the best machine. Therefore, larger tasks are allocated first in the most suitable machines and shorter tasks are mapped afterwards, trying to balance the load of all machines. In some scenarios, Max-Min can provide better schedules than Min-Min;
- *Sufferage*: identifies in each iteration the task that will *suffer* the most if it is not assigned to a certain host. The *sufferage value* is computed as the difference between the best MCT of the task and

its second-best MCT. Sufferage gives precedence to those tasks with high sufferage value, assigning them to the machines that can complete them at the earliest time. This approach could lead to find schedules with better makespan values than Min-Min and Max-Min.

The Min-Min, Max-Min, and Sufferage list scheduling heuristics follow a generic schema which applies two *phases* to perform the task-to-resource assignment: in the first phase, N pairs (task, machine) are selected considering a specific criterion, and then in the second phase one of the N pairs is selected regarding an overall comparison. Finally, there are some efforts to design new implementations of these algorithms with a lower complexity [55,56].

3. Approach

In this section, we present our approach to automatic parallelization. Our starting point is a generic parallel algorithm that satisfies our concurrency objectives, Section 3.1. We briefly present the Savant syndrome, the source of inspiration for our approach, in Section 3.2, and describe how we train the generic parallel algorithm to solve the optimization problem in Section 3.3.

3.1. The target parallel model

In Section 2.1, we mentioned that the previous genetic programming approaches considered parallelism as an objective, this yields uncertain parallelism results. Here, we address the parallelism problem by specifying a target parallel model. This approach will only produce algorithms that conform to this model, thus guaranteeing the degree of parallelism.

The chosen algorithmic model is a single iteration of a Map-Reduce application. Open source and free software frameworks exist for this model, on different architectures (GPU, multi-core, clusters), which makes it a practical choice. Moreover, theoretical works have found it equivalent to BSP and PRAM [57], both well-studied parallel models.

In this model, the input data is first processed independently by many mappers, whose results are then further processed independently by reducers. Independent processing means that the mappers and reducers do not communicate or otherwise synchronize, a key factor of parallel design. In addition to this qualitative definition of the parallel model, we seek a high number of mappers or reducers, regardless of the problem size. Also, the new algorithms must scale with respect to problem size and hardware resources.

3.2. Analogy with the savant syndrome

Our automatic parallelization question can be restated as: how to automatically design a scalable and massively parallel Map-Reduce algorithm, that finds solutions to the independent task mapping problem of comparable quality (the fitness function computes the makespan) to the Min-Min heuristic. We looked for previous occurrences where a massively parallel machine (composed of weak computing nodes, to ensure the reliance on parallel processing), was able to solve small, sequential problems in a short time. This question lead to the Savant syndrome [58–62].

People displaying symptoms of this syndrome can compute small sequential tasks, such as calendar computation (finding the day of the week for a given date), in a very short time (700 msec), using largely unknown methods. Their methods for calendar computation are considered unknown because experiments showed that the distribution of the response time does not match those of known algorithms. Also, Savants can perform other date computations with similar performance, while this is more time-consuming for a computer algorithms (and reported impossible with classical algorithms [60]). Although not fully understood, the Savants seem to be learning pattern-recognition rules from data, which are later applied in parallel to new input. This matches their ability to perform calendar computation while ignoring complicated details of calendars, and to enumerate prime numbers while ignoring what a prime number is, or even how to multiply and divide. The mental activities that some Savants (such as D. Tammet) describe incline us to believe that their pattern-recognition learning method is supervised.

Finally, Savants appear to rely on probabilities: their answers are not 100% correct, and although they are not very proficient in mathematics in general, they understand probabilities better than average. The learned pattern matching is also consistent with other studies, such as chess perception in players of different skill [63,64].

3.3. Application to automatic parallelization

In this section, we use the analogy of the Savant syndrome for the resolution of the independent task

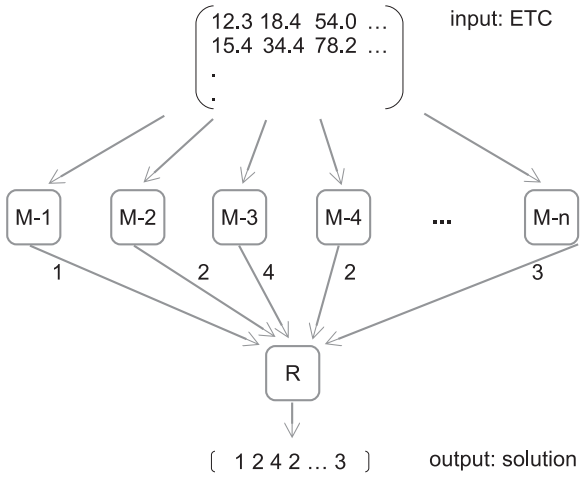

Fig. 1. Overview of the Savant parallel algorithm.

mapping problem. Figure 1 provides an overview of the algorithm. As mentioned in Section 2.2, a solution to the scheduling problem is an array of integers, one per task assignment (the "output" in Fig. 1). Typical instances involve many more tasks than machines, usually hundreds of tasks even for small problems. The concurrent design therefore opts for parallelism at the task level, and decomposes the task assignment into independent task assignment functions. The independent functions are the mappers in the Map-Reduce framework. The combinatorial nature of the problem suggests this approach could find poor solutions, therefore, an additional step can be included to improve the solutions found. This second step is the reducer in Map-Reduce. The next paragraphs further detail the Savant algorithm.

By analogy with the Savant syndrome, the task assignment mappers are multi-class classifiers. Each classifier attempts to correctly assign the task to a machine. Correctness means choosing the same machine assignment as Min-Min, because this is the algorithm we are parallelizing. The mappers' input is the ETC matrix. However, each mapper does not need all or the same ETC data (Section 4.1 provides more details). The classifiers result from supervised learning, in analogy to the Savant syndrome, and because it is well-suited to the parallelization problem (we are given an original algorithm or program to parallelize, which can generate as much training data as required). This efficiency is another advantage of the classification approach over genetic programming, which randomly proposes algorithms in hope of meeting the objectives. We use one mapper per task in the independent task mapping problem. This introduces a high number of mappers for a given problem instance, and scales linearly with the problem size (the number of tasks) and hardware resources (such as CPU cores). Also, the classifiers work completely independently of each other.

The reduce step collects the mappers' output for all tasks and assembles the final solution. A simple reducer can do nothing: just relay the classifiers' solution. However, the independent task mapping problem optimizes a fitness (minimizes makespan), and this fitness is not exploited so far. A reduce step could improve the solution provided by the mappers, by exploiting the fitness function. The reducer we propose is a random local search. It performs a fixed number of random swaps in the solution (swaps machine assignments between two randomly selected tasks) and updates the solution when a swap improves fitness. This reducer runs in constant time, because it only depends on the number of swaps chosen, and is not problem specific (beyond the fitness calculation).

As per Fig. 1, the input is an independent task mapping problem instance: an ETC matrix of size *tasks* $\times$ *machines*. The various M boxes represent the mappers, one for each task. Each box outputs a machine assignment. The R box is the reducer, in this diagram the simple reducer is shown, which collects the classifiers' results.

The classifiers are trained under supervised learning, an observation is an ETC and the Min-Min solution. The next section details the parameters for the training and evaluation of the approach.

4. Experimentation

This section presents the experimentation configuration and the results observed, the solutions and their quality.

4.1. Configuration

The ETC are randomly (but not uniformly) generated according to the procedure in [22]. The ETC columns and rows are further sorted as follows. The low-index tasks have smaller execution time than the high-index tasks. The low-index machines are faster than the high-index machines. The sorting criteria for tasks is the sum of all ETC values, which represents an approximative measure of a task's execution time. The machine sort is possible given the consistent property of the instances [22]. Sorting is considered necessary to

produce classifiers with common index values for tasks and machines, across ETC instances. The assumption is that pattern-recognition rules depend on the nature of the task and machines, but are invariant across instances. Two scheduling problem sizes are used in the experiments, 128 tasks to map across 4 machines (denoted 128×4), and 512 tasks to map across 16 machines (denoted 512×16). The intent is to observe the behavior of the Savant algorithm when the problem size increases.

The classifiers are multi-class SVMs (Support Vector Machines), available via libSVM [65]. We choose the following, recommended, default parameters. The kernel chosen is RBF. Cross-validation is used to select the parameter values. 600 ETC different instances are used for training with 128×4 instances. 1,000 ETC different instances are used for training with 512×16 instances, because of the higher number of classes. 100 ETC different, unseen, instances are used for the evaluation. The ETC input data is scaled.

An important consideration for training of the Savant algorithm is the selection of features for the SVMs. Choosing the entire ETC matrix values for classification yields poor prediction results, and requires longer training time. We have chosen, after investigation, to use a very simple rule: the features used for the classifier of a given task are the values of the ETC column of that task. In other words, a task classifier will rely only on the estimated execution times of that task, on every machine. So, for the 128×4 instances, each task classifier relies on only 4 features. This is a surprising result because the nature of both the combinatorial optimization problem (minimizing a global solution fitness) and the Min-Min heuristic (also relying on global knowledge) suggest that correctly assigning a task requires more knowledge than just the local task profile. Therefore we also investigated several other feature selection mechanisms, however, none improved the prediction scores of the simple rule. First, we extended the features for a task's classifier to the ETC values of similar tasks. Because the ETC is sorted, similar tasks are the neighboring tasks' ETC values. Extending the neighborhood by 1–4 columns of the ETC lead to worse predictions on unseen instances. Generalizing the exploration of features, we built a genetic algorithm [1] that searched for the appropriate features for each task classifier [66–68]. The motivation was to depart from simpler, more intuitive feature sets, at the expense of a long computerized search. The GA proposed very irregular and complex feature sets. Incidentally, the best feature selections included the task's ETC column. Although the search proposed sets which improved the prediction (by a 5–10% percent), the evaluation on completely unseen instances proved slightly worse. Most likely, the GA search suffered from overfitting, which we were not able to mitigate. Finally, the local search reducer of Section 3.3, when used, is run for 10,000 iterations, for both problem sizes.

[1] The GA was an implementation of a Cellular GA.

4.2. Savant's mapper solution similarity

In this section, we compare the solutions to the scheduling problem found by the Savant mapper algorithm, to the ones found by Min-Min. The reducer simply assembles the results from the individual mappers (task classifiers), the local search reducer is not applied. This comparison reflects the mapper's accuracy in predicting the Min-Min assignments for the evaluation (unseen) problem instances. The similarity score is the count of correct task-to-machine classifications across the 100 evaluation problem instances, for each task.

Figure 2 reports the similarity for 128×4 problems. It shows that the accuracy for the smaller and bigger tasks is lower than average. The average accuracy across tasks is approximately 82%, which is surprisingly high, because the Min-Min algorithm chooses machine assignments based on more information (the ETC values of all unassigned tasks, and the current machine completion times) than the 4 ETC values of each Savant mapper. When below average, the accuracy is still greater than 60%. The assignment errors for larger tasks is understandable because the Min-Min assignment for larger tasks occurs later in the algorithm's execution, and depends on the previous task assignments, information that the Savant's mapper does not have. The mismatch between the Savant's mapper and Min-Min for the smaller tasks is caused by approximation in the tasks' sorting, which determines the mapper model to apply for a task. The tasks are sorted by the sum of their ETC values (over the different machines), which is imprecise. However, errors for the smaller tasks is acceptable because they have little influence on the overall fitness function.

Figure 3 reports the similarity between the solutions found by the Savant's mapper and Min-Min, for 512×16 problems. The general curve of the plot is similar to that of the 128×4 problem size, where accuracy

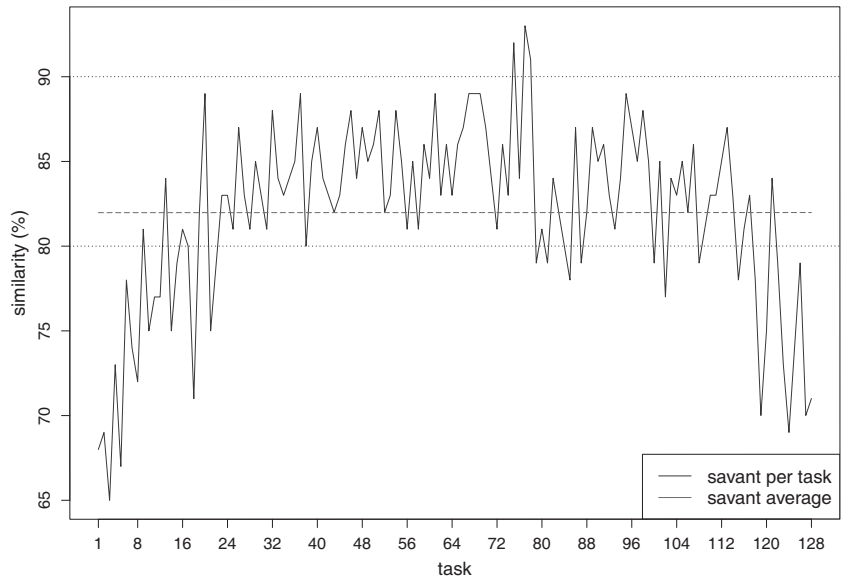

Fig. 2. Savant mapper solution similarity for 128 × 4 problems (without the local search reducer).

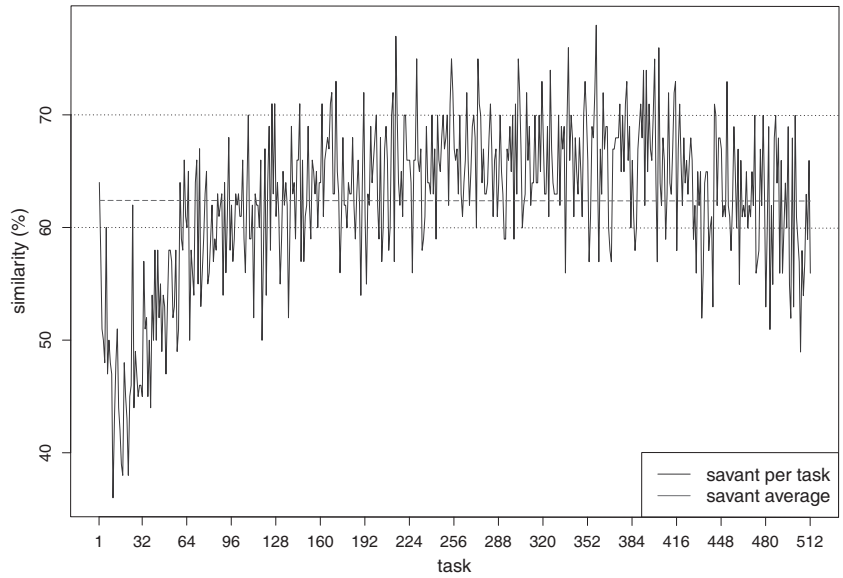

Fig. 3. Savant mapper solution similarity for 512 × 16 problems (without the local search reducer).

for small and large tasks are below average. The accuracy of the mappers is worse than for the smaller problems. The average accuracy increased from the 58% of the original study [9] to 63%. This confirms that the 16 machines to assign tasks to need more training observations than for the 4 machines of the smaller 128 × 4 problems. In this study, the task classifiers are trained with 1,000 observations, versus 600 previously. Overall, the accuracy of the classifiers, operating on only 16 factors (the ETC values for the task), is high.

4.3. Savant's mapper prediction accuracy

In this section, we report on the probability for each Savant's task mapper to correctly assign it's task to a machine, called prediction accuracy. A correct assignment means matching the Min-Min assignment. This information is available in the SVM implementation [65].

The prediction accuracy is different to the similarity plotted in Figs 2–3. The similarity of Section 4.2

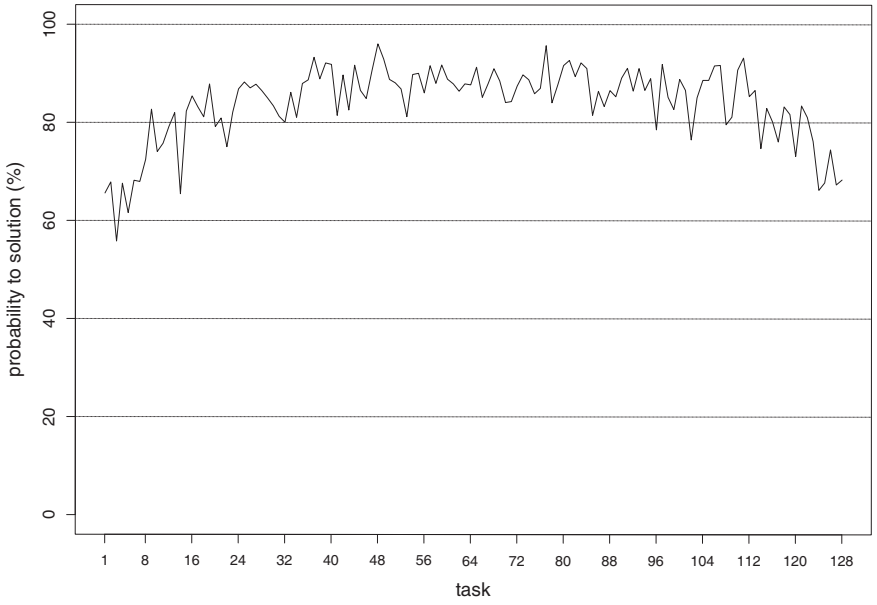

Fig. 4. Savant mapper prediction accuracy for 128×4 problems (without the local search reducer).

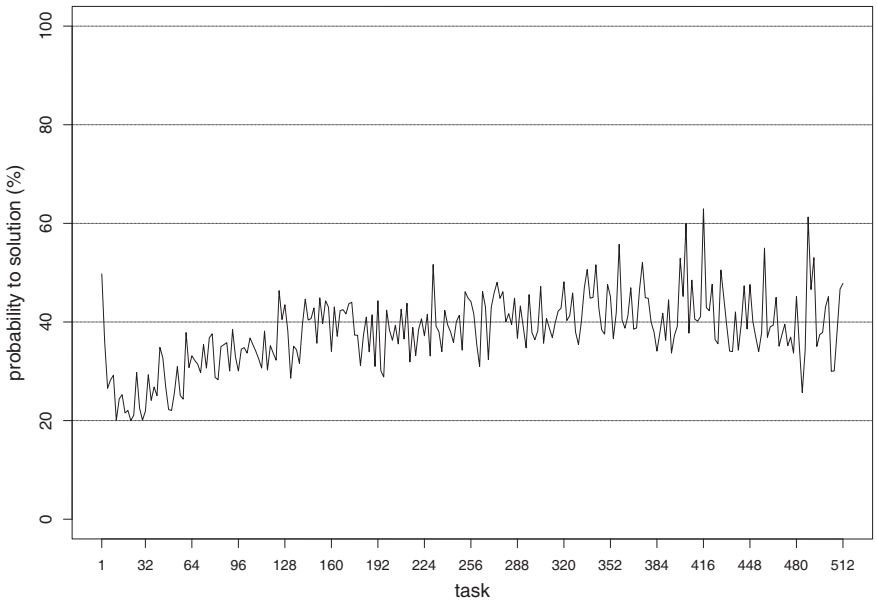

Fig. 5. Savant mapper prediction accuracy for 512×16 problems (without the local search reducer).

is based on the mapper's decision, which is the classifier's highest probability estimate. Here, we report on the probability to correctly assign each task, not always the highest probability. This is useful because in case of mis-assignment (when the highest probability points to a wrong assignment), we would like to know what was the mapper's probability for the correct assignment. This provides a broader accuracy measure of the mappers. Indeed, the probability estimates of a task for the various machine assignments vary greatly. Several assignments can have very close probabilities, reflecting an ambiguous choice, whereas some choices for machine assignments have very different probabilities, reflecting a strong preference.

For smaller problem instances, Fig. 4 shows that the prediction of the SVMs is highly accurate, better

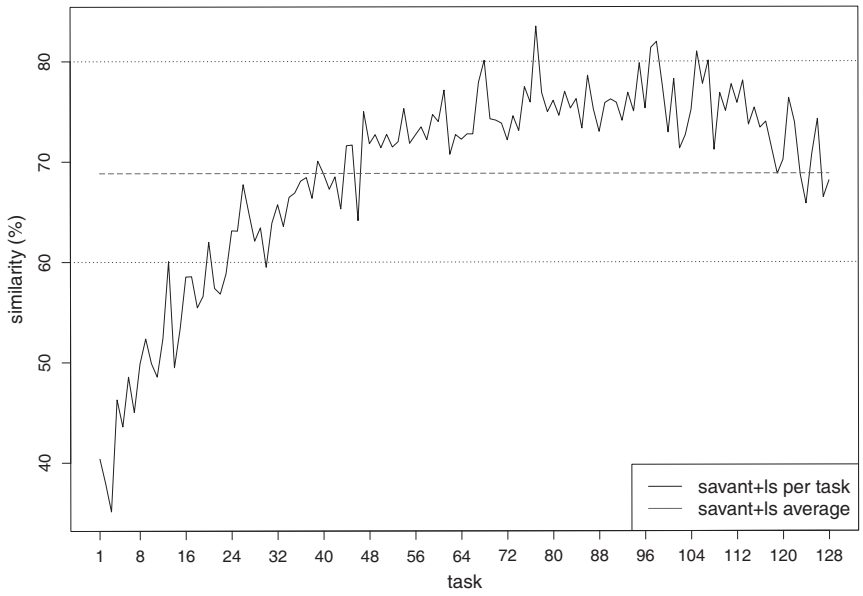

Fig. 6. Savant solution similarity for 128 × 4 problems (with the local search reducer).

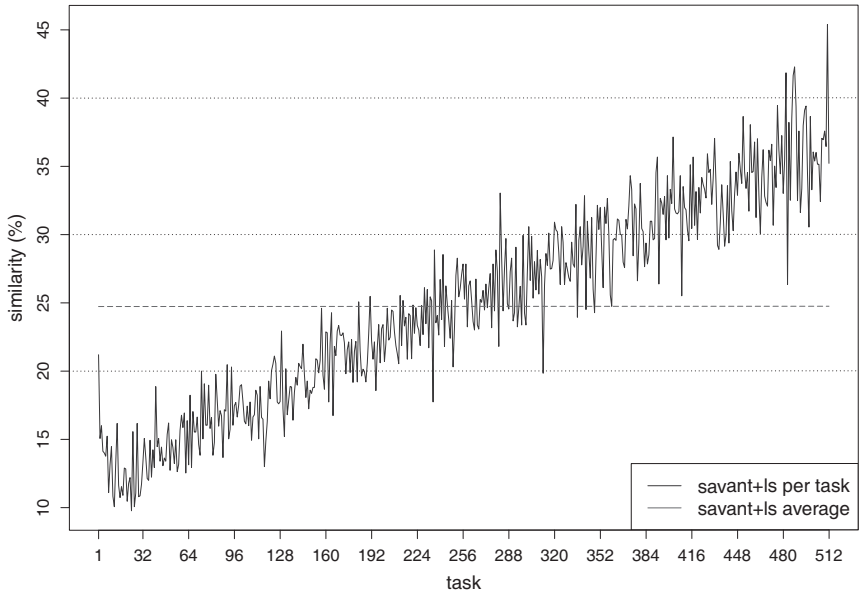

Fig. 7. Savant solution similarity for 512 × 16 problems (with the local search reducer).

than the similarity. This means that even in case of errors, the correct assignment had a high probability for the mapper. The distribution across tasks is similar to Fig. 2.

For larger problem instances, Fig. 5 shows much lower probabilities than the similarity of Fig. 3, because the probability of machine assignments is split across more machines. Also, the accuracy of assignments is less than for smaller problem instances. The distribution of the probabilities across tasks is different to Fig. 3, where the accuracy does not degrade with larger tasks.

4.4. Savant's reducer solution similarity

In the previous sections, we showed the accuracy of the Savant's mappers only. In this section, we show the effect of the reducer step, with the application of the

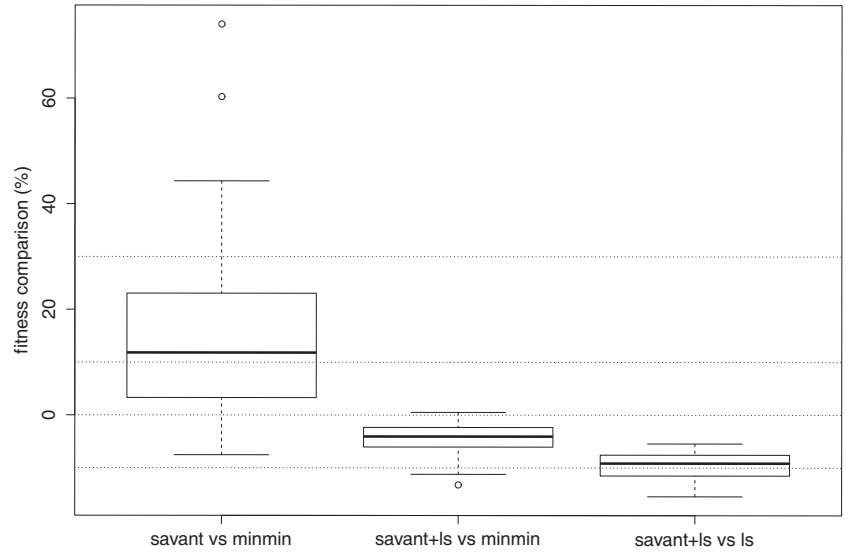

Fig. 8. Savant solution quality for 128 × 4 problems.

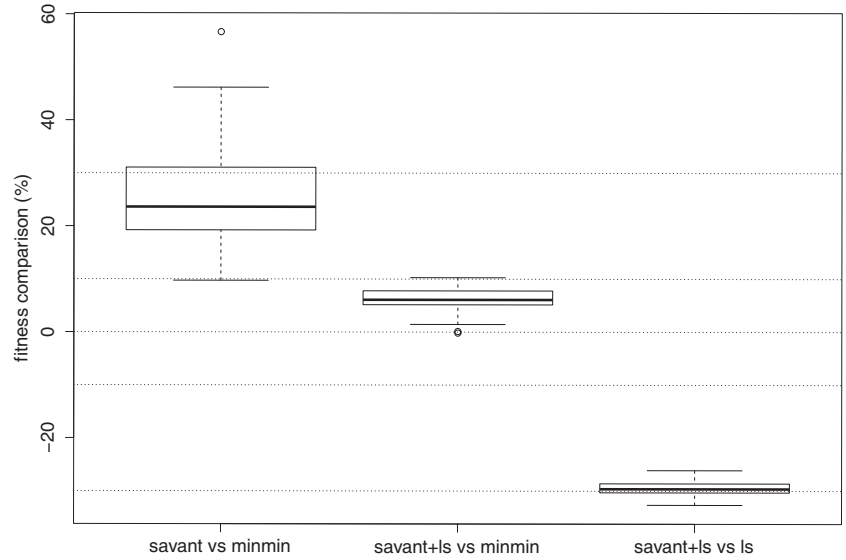

Fig. 9. Savant solution quality for 512 × 16 problems.

stochastic local search, on the similarity to the original Min-Min solutions. The motivation is to determine if the fitness-based solution search leads to solutions similar to Min-Min.

Figure 6 shows that the similarity to Min-Min for 128 × 4 problems is less than the mapper's. This means that the local search, although improving the quality of the solution, does not bring the mapper's solution closer to the Min-Min solutions.

Figure 7, for 512 × 16 problems, also shows lower similarity to Min-Min than the mapper. However, more clearly than in smaller problem instances, the distribution of similarity across tasks is different to the mapper's similarity, where similarity after local search increases with the task size. Better quality solutions seem to share the machine assignments of the larger tasks across algorithms.

4.5. Savant solution quality

In this section, we evaluate the quality of the solutions found by the Savant algorithm. This evaluation is

presented in the form of comparisons with other algorithms. Three comparisons are shown for each problem size; 128 $\times$ 4 and 512 $\times$ 16. The comparison results are plotted as boxplots.

The leftmost boxplot (labeled "savant vs minmin" in the figures) reports the differences in solution quality (fitness, expressed in %) between the Savant algorithm without the local search reducer and the original Min-Min algorithm, for the same 100 evaluation ETC instances. The middle boxplot (labeled "savant+ls vs minmin") shows the same comparison but with the local search reducer presented in Section 3.3. The local search in the reducer may be solely responsible for these results, because of its fitness-based stochastic search. To measure the impact of the fitness-based reducer, we add the rightmost boxplot (labeled "savant+ls vs ls") that compares the Savant with local search reducer with the same local search applied to random solutions. This exposes the influence of the mapper component of the Savant on the solution quality.

Figure 8 shows the boxplot results for the 128 $\times$ 4 problem instances. We note that the Savant algorithm using only the classifiers (mappers) produces good results. The solutions are only 10% worse, in median, than the Min-Min solutions, while 20% different. Applying the local search in the reduce step, Savant results are very good: the algorithm finds better solutions than Min-Min. The rightmost boxplot ("savant+ls vs ls") shows that the classifiers do play a role in the quality of the solutions found, because the random solutions with local search are much worse than the mappers with local search.

Figure 9 shows results for the larger 512 $\times$ 16 problem instances. The mapper only solutions are much worse than the Min-Min solutions, although the fitness is much improved upon our previous experiments [9] with less training observations. We conclude that the poor accuracy observed in Figs 3 and 5 impacts the quality of the solutions found. However, the Savant with the application of the local search reducer still produces good results. The rightmost boxplot shows that the Savant classifiers contribute significantly to the quality of the solutions found. In both problem sizes, we notice that the local search reducer significantly reduce the variance in the results. The same number of local search iterations was performed on both problem sizes, although the search space is much larger. This may explain the greater difference in solution quality for 512 $\times$ 16 problem instances.

5. Conclusions

This paper investigated the possibility of automatically parallelizing a heuristic for a combinatorial optimization problem. The parallel model is Map-Reduce. The long term goal is to find a parallelization method applicable to as many algorithms as possible. The approach presented, Savant, contrasts with previous work: we defined a generic parallel pattern-matching engine (suited to Map-Reduce) that learns the algorithm to parallelize. The parallel algorithm produced is completely different from the original sequential algorithm, yet achieves the same results. This approach is more natural on an optimization problem, because the quality of a solution is based only on the solution, and not on the process that created or found it. We consider the results presented promising. The Savant algorithm provides solutions of comparable quality to the original algorithm.

One could argue that the optimization problem instances addressed are not large, and the original algorithm (Min-Min) is fast on such problem sizes. However, our goal is to automatically design competitive parallel algorithms, rather than rival on execution speed, because of the on-going trend in computing architectures.

There is also room for improvement. The degraded solution similarity and quality when the problem size increases needs to be addressed. We plan to use more training samples when the number of classes increases, as is the case when problem size increases. A more fundamental improvement is the feature selection for the tasks SVMs. We used a simple rule, common to all task SVMs (that nevertheless achieves good results): select the task's ETC column. We are currently investigating the use of different feature selection rules for each task, given the tasks' differences. One alternative is to use different ETC columns for each task's classifier. Another alternative is to use elements of the ETC matrix, instead of columns. We plan to automatically discover such rules, so as to meet our goal of automatic parallelization. Another improvement is the re-design of the reducer step, into a concurrent version that is also generic.

Other future work will investigate if the Savant algorithm is suitable for different, more elaborate and thus time-consuming, algorithms for the same optimization problem. Also, we need to understand how this approach performs on different problems altogether.

Acknowledgments

We thank the anonymous reviewers for their comments and suggestions for improvement.

References

[1] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek et al., A view of the parallel computing landscape, *Communications of the ACM* **52**(10) (2009), 56–67.

[2] A.S. William Gropp, Ewing Lusk, *Using MPI*. MIT Press, 1999.

[3] J. Armstrong, The development of Erlang, in Proceedings of the second ACM SIGPLAN international conference on Functional programming, ser. ICFP '97. New York, NY, USA: ACM, 1997, pp. 196–203. [Online]. Available: http://doi.acm.org/10.1145/258948.258967.

[4] D. Callahan, B.L. Chamberlain and H.P. Zima, The cascade high productivity language, in *High-Level Parallel Programming Models and Supportive Environments, 2004. Proceedings. Ninth International Workshop on*, IEEE, 2004, pp. 52–60.

[5] Google, The go programming language, http://golang.org/.

[6] Mozilla, The rust programming language, http://www.rust-lang.org/.

[7] J. Dean and S. Ghemawat, Mapreduce: simplified data processing on large clusters, *Communications of the ACM* **51**(1) (2008), 107–113.

[8] D. Shasha and C. Lazere, *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*, 1st ed., Copernicus Books, 1998.

[9] F. Pinel, B. Dorronsoro, P. Bouvry and S.U. Khan, Savant: Automatic parallelization of a scheduling heuristic with machine learning, in: *Nature and Biologically Inspired Computing (NaBIC), 2013 World Congress on*, IEEE, 2013, pp. 52–57.

[10] C.D. Callahan, K.D. Cooper, R.T. Hood, K. Kennedy and L. Torczon, ParaScope: A parallel programming environment, *International Journal of High Performance Computing Applications* **2**(4) (1988), 84–99.

[11] F. Irigoin, P. Jouvelot and R. Triolet, Semantical interprocedural parallelization: An overview of the PIPS project, in: *Proceedings of the 5th international conference on Supercomputing*, ACM, 1991, pp. 244–251.

[12] X. Leroy, Formal certification of a compiler back-end or: Programming a compiler with a proof assistant, in *ACM SIGPLAN Notices* **41**(1) (2006), 42–54.

[13] D. Zhang and J.J. Tsai, Machine learning and software engineering, *Software Quality Journal* **11**(2) (2003), 87–119.

[14] M. Harman, The current state and future of search based software engineering, in: *2007 Future of Software Engineering*, IEEE Computer Society, 2007, pp. 342–357.

[15] C. Ryan, A.H. van Roermund and C.J.M. Verhoeven, *Automatic re-engineering of software using genetic programming*, Kluwer Academic, 2000.

[16] J.R. Koza, Genetic programming as a means for programming computers by natural selection, *Statistics and Computing* **4**(2) (1994), 87–112.

[17] P. Walsh and C. Ryan, "Paragen: a novel technique for the autoparallelisation of sequential programs using gp," in *Proceedings of the First Annual Conference on Genetic Programming*. MIT Press, 1996, pp. 406–409.

[18] S.M. Cheang, K.S. Leung and K.H. Lee, Genetic parallel programming: Design and implementation, *Evolutionary Computation* **14**(2) (2006), 129–156.

[19] K.S. Leung, K.H. Lee and S.M. Cheang, Evolving parallel machine programs for a multi-alu processor, in *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, vol. 2, IEEE, 2002, pp. 1703–1708.

[20] K. Thearling and T.S. Ray, Evolving parallel computation, *Complex Systems* **10**(3) (1996), 229.

[21] D.E. Goldberg, Genetic and evolutionary algorithms come of age, *Communications of the ACM* **37**(3) (1994), 113–119.

[22] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen and R.F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *J. Parallel Distrib Comput* **61** (June 2001), 810–837. [Online]. Available: http://portal.acm.org/citation.cfm?id=511973.511979.

[23] E. Horowitz and S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, *Journal of the ACM (JACM)* **23**(2) (1976), 317–327.

[24] F. Pinel, B. Dorronsoro, J. Pecero, P. Bouvry and S.U. Khan, A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational grids, *Cluster Computing, The Journal of Networks, Software Tools, and Applications* **16**(3) (2013), 421–433.

[25] T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys and B. Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *Journal of Parallel and Distributed Computing* **61**(6) (2001), 810–837.

[26] J. Carretero and F. Xhafa, Using genetic algorithms for scheduling jobs in large scale grid applications, *Journal of Technological and Economic Development – A Research Journal of Vilnius Gediminas Technical University* **12**(1) (2006), 11–17.

[27] G. Ritchie and J. Levine, A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, in: *Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG)*, 2004.

[28] F. Xhafa, E. Alba, B. Dorronsoro and B. Duran, Ecient batch job scheduling in grids using cellular memetic algorithms, *Journal of Mathematical Modelling and Algorithms* **7**(2) (2008), 217–236.

[29] F. Pinel, B. Dorronsoro and P. Bouvry, A new parallel asynchronous cellular genetic algorithm for scheduling in grids, in *Nature Inspired Distributed Computing (NIDISC) sessions of the International Parallel and Distributed Processing Symposium (IPDPS) Workshop*, 2010.

[30] S. Nesmachnow, H. Cancela and E. Alba, Heterogeneous computing scheduling with evolutionary algorithms, *Soft Computing* **15**(4) (2010), 685–701.

[31] S. Solomon, P. Thulasiraman and R. Thulasiram, Collaborative multi-swarm PSO for task matching using graphics processing units, in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 2011, pp. 1563–1570.

[32] S. Nesmachnow and M. Canabé, GPU implementations of scheduling heuristics for heterogeneous computing environments, in *Proceedings of the XVII Congreso Argentino de Ciencias de la Computacion*, 2011, pp. 1563–1570.

[33] F. Pinel, B. Dorronsoro and P. Bouvry, Solving very large instances of the scheduling of independent tasks problem on the GPU, *Journal of Parallel and Distributed Computing* **73**(1) (2013), 101–110.

[34] F. Xhafa and A. Abraham, Computational models and heuristic methods for grid scheduling problems, *Future Gener Comp Sy* **26** (2010), 608–621.

[35] I. Das and J.E. Dennis, A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems, *Structural and Multidisciplinary Optimization* **14**(1) (1997), 63–69.

[36] J. Yu and R. Buyya, A budget constrained scheduling of workflow applications on utility grids using genetic algorithms, in *HPDC*, 2006, pp. 1–10.

[37] J. Sweeney and S.P. Ahuja, Heuristic solutions to resource allocation in grid computing: A natural approach, *J Supercomput* **44** (2008), 179–198.

[38] F. Xhafa, J. Carretero, B. Dorronsoro and E. Alba, A tabu search algorithm for scheduling independent jobs in computational grids, Comput Inform, special issue on Intelligent Computational Methods and Models, vol. 28, 2009, pp. 1001–1014.

[39] F. Xhafa, J.A. Gonzalez, K.P. Dahal and A. Abraham, A GA(TS) hybrid algorithm for scheduling in computational grids, in *Hybrid Artificial Intelligence Systems*, ser. LNAI, vol. 5572, Springer, 2009, pp. 285–292.

[40] L. Zhong, Z. Long, J. Zhang and H. Song, An efficient memetic algorithm for job scheduling in computing grid, in *Information and Automation*, ser. CCIS, vol. 86. Springer, 2011, pp. 650–656.

[41] P. Krömer, V. Snásel, J. Platos, A. Abraham and H. Ezakian, Evolving schedules of independent tasks by differential evolution, in *Intelligent Networking, Collaborative Systems and Applications*, ser. SCI **329** (2011), 79–94.

[42] A. Abraham, H. Liu, C. Grosan and F. Xhafa, Metaheuristics for Scheduling in Distributed Computing Environments, ser. SCI. Springer, 2008, vol. 146, ch. Nature Inspired Metaheuristics for Grid Scheduling: Single and Multi-objective Optimization Approaches, pp. 247–272.

[43] W. Jakob, A. Quinte, K.-U. Stucky and W. Süb, Fast multiobjective scheduling of jobs to constrained resources using a hybrid evolutionary algorithm, in *Parallel Problem Solving from Nature (PPSN X)*, ser. LNCS, vol. 5199. Springer, 2008, pp. 1031–1040.

[44] W. Jakob, A. Quinte, K.-U. Stucky and W. Süb, Fast multiobjective rescheduling of grid jobs by heuristics and evolution," in *Parallel Processing and Applied Mathematics (PPAM)*, ser. LNCS, vol. 6068. Springer, 2010, pp. 21–30.

[45] S. Nesmachnow, Parallel multiobjective evolutionary algorithms for batch scheduling in heterogeneous computing and grid systems, *Computational Optimization and Applications* **55** (2013), 515–544.

[46] S. Nesmachnow and S. Iturriaga, Multiobjective scheduling on distributed heterogeneous computing and grid environments using a parallel micro-chc evolutionary algorithm, in *Int Confence on P2P, Parallel, Grid, Cloud and Internet Computing*, 2011, pp. 134–141.

[47] I. De Falco, A. Della Cioppa, D. Maisto, U. Scafuri and E. Tarantino, *A Multiobjective Extremal Optimization Algorithm for Efficient Mapping in Grids*, ser. Advances in Soft Computing, Springer-Verlag, 2009, vol. 58, pp. 367–377.

[48] B. Dorronsoro, P. Bouvry, J.A. Cañero, A.A. Maciejewski and H.J. Siegel, Multi-objective robust static mapping of independent tasks on grids, in IEEE Congress on Evolutionary Computation (CEC), part of the World Congress on Computational Intelligence (WCCI), 2010, pp. 3389–3396.

[49] S. Nesmachnow, B. Dorronsoro, J.E. Pecero and P. Bouvry, Energy-aware scheduling on multicore heterogeneous grid computing systems, *Journal of Grid Computing* **11**(4) (2013), 653–680.

[50] K. Kim, R. Buyya and J. Kim, Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters, in *Proc. of the 7th IEEE International Symposium on Cluster Computing and the Grid*, 2007, pp. 541–548.

[51] Y. Li, Y. Liu and D. Qian, A heuristic energy-aware scheduling algorithm for heterogeneous clusters, in *15th International Conference on Parallel and Distributed Systems (ICPADS)*, 2009, pp. 407–413.

[52] P. Luo, K. Lü and Z. Shi, A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems, *J Parallel Distrib Comput* **67**(6) (2007), 695–714.

[53] O. Ibarra and C. Kim, Heuristic algorithms for scheduling independent tasks on nonidentical processors, *Journal of the ACM* **24**(2) (1977), 280–289.

[54] R. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. Lima, F. Mirabile, L. Moore, B. Rust and H. Siegel, Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet, in *Proc. of the 7$^{\text{th}}$ Heterogeneous Computing Workshop*. Washington DC, USA: IEEE Computer Society, 1998, p. 3.

[55] E.K. Tabak, B.B. Cambazoglu and C. Aykanat, Improving the performance of independenttask assignment heuristics minmin, maxmin and sufferage, *IEEE Transactions on Parallel and Distributed Systems* **25**(5) (2014), 1244–1256.

[56] P. Ezzatti, M. Pedemonte and A. Martin, An efficient implementation of the min-min heuristic, *Computers and Operations Research* **40**(11) (2013), 2670–2676.

[57] M.F. Pace, Bsp vs mapreduce, *Procedia Computer Science* **9** (2012), 246–255.

[58] H. Welling, Prime number identification in idiots savants: Can they calculate them? *Journal of autism and developmental disorders* **24**(2) (1994), 199–207.

[59] L. Mottron, M. Dawson, I. Soulieres, B. Hubert and J. Burack, Enhanced perceptual functioning in autism: An update, and eight principles of autistic perception, *Journal of Autism and Developmental Disorders* **36**(1) (2006), 27–43.

[60] L. Mottron, K. Lemmens, L. Gagnon and X. Seron, Non-algorithmic access to calendar information in a calendar calculator with autism, *Journal of Autism and Developmental Disorders* **36**(2) (2006), 239–247.

[61] J.R. Hughes, A review of savant syndrome and its possible relationship to epilepsy, 2010.

[62] H. Darius, Savant syndrome-theories and empirical findings, Ph.D. dissertation, University of Skövde, 2007.

[63] W.G. Chase and H.A. Simon, Perception in chess, *Cognitive Psychology* **4**(1) (1973), 55–81. Available: http://www.sciencedirect.com/science/article/pii/0010028573900042.

[64] N. Charness, E.M. Reingold, M. Pomplun and D.M. Stampe, The perceptual aspect of skilled performance in chess: Evidence from eye movements, *Memory and Cognition* **29**(8) (2001), 1146–1152.

[65] C.-C. Chang and C.-J. Lin, LIBSVM: A library for support vector machines, *ACM Trans. Intell. Syst. Technol.* **2**(3) (May 2011), 27:1–27:27. [Online]. Available: http://doi.acm.org/10.1145/1961189.1961199.

[66] N.T. Siebel and G. Sommer, Evolutionary reinforcement learning of artificial neural networks, *International Journal of Hybrid Intelligent Systems* **4**(3) (2007), 171–183.
[67] A. Castaño, F. Fernández-Navarro, C. Hervás-Martínez, M. García and P.A. Gutiérrez, Classification by evolutionary generalised radial basis functions, *International Journal of Hybrid Intelligent Systems* **7**(4) (2010), 239–248.
[68] C.M. Mufassil Wahid, A. Shawkat Ali and K.S. Tickle, Hybrid feature selection through feature clustering for microarray gene expression data, *International Journal of Hybrid Intelligent Systems* **10**(4) (2013), 165–178.