# Virtual Savant for the Heterogeneous Computing Scheduling Problem

Renzo Massobrio
Universidad de la República, Uruguay
Universidad de Cádiz, Spain
Email: renzom@fing.edu.uy

Bernabé Dorronsoro
Universidad de Cádiz, Spain
Email: bernabe.dorronsoro@uca.es

Sergio Nesmachnow
Universidad de la República, Uruguay
Email: sergion@fing.edu.uy

***Abstract*—A key issue when using distributed computing environments is finding a planning strategy to execute tasks in order to use the computational resources efficiently. This article presents the application of Virtual Savant to solve the heterogeneous computing scheduling problem, a widely-studied problem with several real-world applications. Virtual Savant is a novel method that uses machine learning techniques to automatically generate programs that can be executed in parallel to solve a given problem. Experimental analysis is performed on a set of problem instances generated following methodologies from the related literature. Results show that Virtual Savant is able to outperform MinMin, a well-known heuristic for the studied problem, by up to 15% while showing good scalability properties when increasing the number of computing resources and the dimension of the problem instances.**

## I. Introduction

Scheduling is one of the most important stages in the operation of a High Performance Computing (HPC) infrastructure. The goal of the scheduling problem is to optimally assign the computing resources by satisfying some efficiency criteria, usually related to the total execution time or resource utilization. Traditional scheduling problems are $\mathcal{NP}$-hard [1], thus classic exact methods are only useful for solving problem instances of reduced size. When dealing with large-dimension computing environments, metaheuristic techniques emerge as promising methods for solving scheduling problems. These methods allow computing accurate solutions in reasonable execution times, which usually satisfy the efficiency requirements for real-life scenarios.

In the context of HPC, finding accurate solutions has an important effect on the performance of the system from the point of view of both the user, because it implies a high quality of service (QoS), and the provider, thanks to an efficient use of the available resources. Computing the schedule of all arriving tasks directly impacts on the response time offered to users. We define the response time of the system as the time since the user submits his/her job until its execution is started. For this reason, current schedulers are simple heuristics that can offer acceptable solutions in short computation times (generally less than a second). In contrast, metaheuristics offer more accurate solutions to the problem, but at the cost of considerably longer response times. Therefore, there is a need for new methodologies that can quickly find high quality schedules.

We propose in this work the application of the Virtual Savant (VS) paradigm [2] to find accurate solutions for the heterogeneous computing scheduling problem (HCSP) in short computation times.

The VS paradigm is inspired by the savant syndrome, a rare condition in which a person demonstrates mnemonic or computing abilities far superior to what would be considered normal. As an example, some patients with savant syndrome (*savants*) are able to enumerate and identify huge prime numbers without the underlying knowledge of what a prime number is. Evidence suggests that patients with savant syndrome use pattern recognition in order to efficiently solve problems [3], [4], [5].

Analogously, VS aims at generating a program that can run in parallel by using machine learning techniques to predict the results computed by a reference algorithm that solves a given problem [2], [6]. VS receives as input both a set of problem instances and the results computed by a reference algorithm, which is used to train a machine learning classifier. Once the training phase is completed, VS can run in parallel to solve new, unknown, and even larger problem instances.

Therefore, the VS paradigm allows automatically generating programs by learning the behavior of a given optimization algorithm in order to generate a completely different program that reproduces it. The automatically generated program is lightweight and massively parallel. We use VS in this work to learn how to generate accurate solutions from a specialized heuristic.

VS showed promising preliminary results for HCSP [2], [7], [8] and knapsack problems [9], [10]. In this article, we make a deeper study on the performance of VS on HCSP, which proposes scheduling a set of independent tasks on a number of heterogeneous computing resources [11]. HCSP is an $\mathcal{NP}$-hard combinatorial optimization problem with real applications in modern computing facilities.

The main contribution of this article is the design and implementation of the parallel VS for HCSP. Despite its parallel design, this is the first time that VS is evaluated in a parallel environment. The new design has been carefully analyzed, studying its accuracy and its parallel capabilities and performance on a server equipped with an Intel® Xeon Phi™ 7250 processor. VS was trained in this work from solutions of MinMin, a well-known heuristic for this prob-

lem. In the comparison of VS against MinMin, our proposal outperforms the latter by up to 15% in terms of accuracy of solutions, while showing good scalability properties when increasing both the number of computing elements and the problem dimension.

The remainder of the article is organized as follows. Section II briefly describes the HCSP. Section III surveys some relevant proposals addressing the HCSP in the literature. Section IV presents the well known MinMin heuristic for HCSP, while Section V details the application of VS to the considered problem. Section VI presents the experimental evaluation of the proposed solution, and Section VII gives our main conclusions and lines of future work.

## II. Heterogeneous Computing Scheduling Problem

A heterogeneous computing system is a coordinated set of processing elements, often called resources, processors or simply machines, interconnected by a network. The heterogeneous quality refers to the variable computational capabilities of the resources (e.g., CPU processing power). A crucial problem for heterogeneous computing systems consists in finding a suitable task-to-machine assignment that optimizes some metric usually related to efficiency, resource utilization, economic profit, or quality of service.

Consider a heterogeneous computing system composed of several machines and a set of tasks with variable computational requirements to be executed in the system. A task is an atomic workload unit, i.e., it cannot be split into smaller chunks nor interrupted after it starts its execution. The execution time of any individual task vary from one machine to another. Assuming that the execution time of each task on each machine is known, HCSP proposes finding a task-to-machine assignment that optimizes some quality metric. The proposed scheduler focuses on optimizing the *makespan*, a well-known optimization criterion which is defined as the period of time between the start of the first task and the completion of the last task. Makespan is a measure of the productivity (throughput) of a computing system.

The HCSP mathematical formulation considers the following elements:

- A set of tasks $T = \{t_1, \ldots, t_n\}$ to be scheduled.
- A set of heterogeneous machines $M = \{m_1, \ldots, m_m\}$.
- A function $ET : T \times M \rightarrow \mathcal{R}^+$ where $ET(t_i, m_j)$ indicates the execution time of task $t_i$ on machine $m_j$.

The goal of the HCSP is to find an assignment function $f : T \rightarrow M$ that minimizes the makespan, defined by Eq. 1.

$$\text{makespan} = \max_{m_j \in M} \sum_{t_i \in T, f(t_i) = m_j} ET(t_i, m_j) \qquad (1)$$

The proposed model does not account for dependencies among tasks: the problem formulation assumes that all tasks can be independently executed, without considering the execution order. Even though it is a simplified version of the more general scheduling problem that accounts for task dependencies, the independent task model is specially important in distributed computing environments. Independent-task applications frequently appear in many lines of scientific research as well as in shared infrastructures, where different users submit tasks to be executed. Thus, the relevance of the HCSP version faced in this work is justified due to its significance in realistic distributed computing environments.

## III. Related work

In the last twenty years, research on the HCSP has been increasing due to the popularity of modern parallel and distributed computing systems. HCSP is an $\mathcal{NP}$-hard combinatorial optimization problem [12]. Therefore, heuristic and metaheuristics approaches are good candidates to address the HCSP, specially in real-life scenarios were schedulers are expected to provide accurate solutions in short execution times. A large number of heuristics [13], [14] and metaheuristics [15], [16], [17], [18], [19], [20] have been proposed for efficiently solving the HCSP. Among other metaheuristic techniques, evolutionary algorithms (EAs) have been widely applied for solving scheduling problems in heterogeneous computing systems.

Braun et al. presented a systematic comparison of 11 scheduling heuristics for the HCSP, including an EA and an hybrid algorithm which combined an EA with Simulated Annealing [13]. The proposed algorithms were seeded during the population initialization in order to significantly improve the search. Both methods were able to obtain the best known makespan values at that time for the studied HCSP scenarios.

Duran and Xhafa studied a steady-state genetic algorithm (GA) and the Struggle GA (SGA) for the HCSP [21]. Both EAs outperformed the previous best makespan results in more than half of the instances studied. A memetic algorithm (MA) proposed in 2007 by Xhafa included subordinate local search methods to find high-quality solutions in short execution times [22]. Later, the structured population of a cellular MA (cMA) was used to control the tradeoff between the exploitation and exploration of the HCSP solution space [23]. Using a seeded population initialization and three local search methods, cMA outperformed previous GA results for half of the instances from Braun et al.

Recent works explored applying the use of machine learning methods for estimating the performance of applications on different heterogeneous resources [24], for performance estimation and resource selection [25], and for predicting the speedup of CPU/GPU kernels [26]. However, to the best of our knowledge, no other approaches for automatically designing schedulers using machine learning techniques exist, rather than ours [27], [28], [2], [6].

## IV. The MinMin algorithm

MinMin is one of the most widely used methods for solving the HCSP [29]. MinMin is a two-phase greedy scheduler that greedily picks the task that can be completed the soonest. Starting from a set $U$ of all unmapped tasks, MinMin determines the machine that provides the Minimum Completion

Time (MCT) for each task in $U$, and assigns the task with the minimum overall MCT to its best machine. The mapped task is removed from $U$, and the process is repeated until all tasks are mapped. MinMin does not consider a single task at a time but all the unmapped tasks sorted by MCT. The availability status of the machines is updated accordingly after each assignment.

The idea behind Minmin is to select at every time that task from the unscheduled ones that is causing the minimum increase in the overall makespan value. Therefore, the algorithm is focusing on balancing the load among all the processors. This procedure generally leads to more balanced schedules and better makespan values than other heuristics, since tasks that can be completed the earliest are assigned first to the corresponding machine [13].

## V. Virtual Savant for HCSP

Virtual Savant is a novel paradigm to automatically generate programs that solve optimization problems in a parallel fashion [18]. The paradigm is inspired by the Savant syndrome, a rare condition in which a person with significant mental disabilities has certain abilities far in excess of what would be considered normal [3]. People with this condition (*savants*) usually excel at one specific skill such as art, memory, rapid calculation, or musical abilities. The methods used by savants to solve problems are not fully understood due to the difficulties in communicating with them. The main hypothesis states that savants learn through pattern recognition [4]. This allows them solving a given problem without understanding the underlying principles (e.g., being able to enumerate prime numbers without understanding what a prime number is).

In analogy to the Savant syndrome, VS is trained using machine learning techniques to automatically learn how to solve an optimization problem from a set of observations, which are usually obtained from a reference algorithm. Once the training is completed, VS can emulate the reference algorithm to solve new, unknown, and even larger problem instances, without the need of any further training. The VS paradigm consists of two phases: *classification*, where results for unknown problem instances are predicted, and *improvement*, where predicted results are further improved using specific search procedures.

The VS implementation for the HCSP uses Support Vector Machines (SVMs) for the classification phase. SVMs are trained using MinMin as the reference algorithm. Each task is considered individually during the training phase of VS. Therefore, each feature vector holds the execution time of one task on each machine and the classification label is the machine assigned to that task by the MinMin heuristic. This way, one single MinMin solution provides as many observations as the number of tasks the problem instance has. The LIBSVM framework with a Radial Basis Function kernel was used [30]. A specific fork of the LIBSVM package was designed to improve training times on manycore architectures [31]. Figure 1 outlines the training scheme for VS to solve the HCSP.

The complete model of VS to solve the HCSP is presented in Figure 2. Once the learning process is completed, VS uses

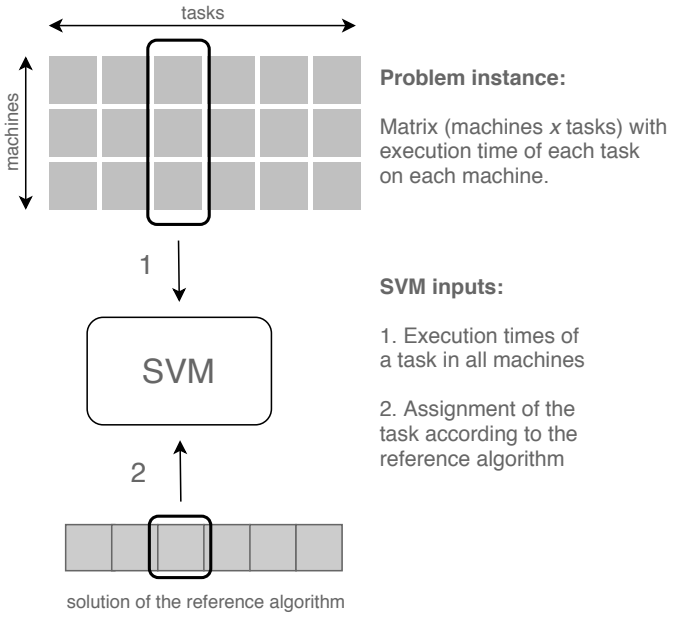



Fig. 1: Training scheme of VS for the HCSP

multiple instances of the trained SVM in parallel to predict the assignment of each task. These assignments are made independently for each task, providing VS with a high degree of parallelism. The output of the classification phase is a matrix that holds, for each task, its assignment probability for each machine. Because tasks are independently assigned, VS can be scaled to problem instances with any number of tasks, without requiring any additional training process. However, since the length of the training vectors depends on the number of machines, different SVMs must be trained in order to solve problem instances with varying number of machines. This does not impose a huge limitation on the proposed scheme since, in a real environment, the number of machines has usually low variability when compared to the number of tasks to schedule.

The improvement phase takes the resulting matrix of probabilities of the prediction phase, randomly generates feasible solutions considering the probability of assigning each task to each machine, and performs a simple local search heuristic over each generated solution. The local search operator considered in this work iteratively moves a randomly-chosen task from the most loaded machine (i.e., the one with the highest completion time) to a machine selected among the $N$ least loaded ones. The selected machine is the one with the smallest completion time after the task is moved. For the experimental analysis $N$ was set to 50% of the total number of machines. When all local searches are completed, the overall best solution found is returned.

## VI. Experimental analysis

This section reports the evaluation of the proposed VS for the HCSP.

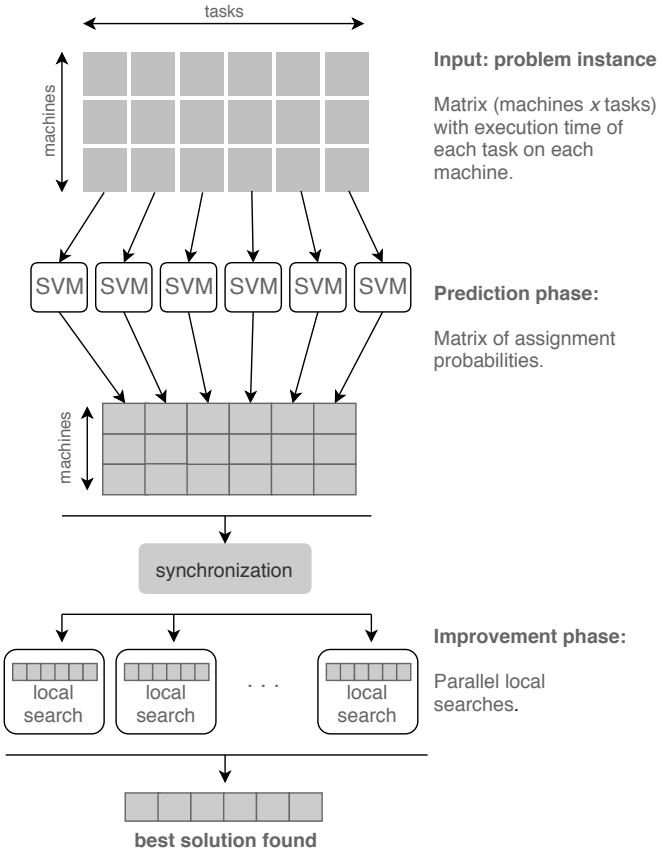



Fig. 2: Virtual Savant for the HCSP

### A. Methodology

*Execution platform*. Experiments were performed on a server with Intel®Xeon Phi™ 7250 processor (68 cores and 64GB RAM). The server was used exclusively during the experiments to accurately measure execution times.

*Problem instances*. A set of HCSP instances were generated according to the methodology described by Braun et al. [13]. Tasks were considered as highly heterogeneous and machines were considered consistent (i.e., a machine cannot be faster than another for a given task and slower for another task). High (*hi*) and low (*lo*) machine heterogeneity configurations were considered in the experimental evaluation. Three problem dimensions were studied (tasks×machines): 128×4, 512×16, and 1024×16, and 30 different problem instances were generated for each combination of problem dimension and machine heterogeneity, totaling 180 problem instances. The notation used to describe each problem instance is: dimension (tasks×machines), machine heterogeneity (*hi*/*lo*), and instance number ($i \in [1 \dots 30]$).

### B. Scalability using parallel processing elements

Tables I and II report the makespan and execution time (in seconds) of VS when varying the number of threads used for the classification phase and for the improvement phase. Results correspond to 30 independent executions performed with each number of threads. Results correspond to instance *512×16_hi_1* and are representative of the trends obtained for other problem instances. Boxplots in Fig. 3 show a graphical summary of the results.

| # *threads* | *makespan* | | | |
|---|---|---|---|---|
| | best | mean | median | std |
| 1 | 1707.78 | 1722.62 | 1720.03 | 10.44 |
| 20 | 1698.94 | 1705.72 | 1705.40 | 3.61 |
| 40 | 1697.67 | 1704.99 | 1705.20 | 3.05 |
| 60 | 1691.55 | 1702.30 | 1702.80 | 3.09 |
| 80 | 1695.61 | 1703.38 | 1704.59 | 2.94 |
| 100 | 1696.52 | 1702.02 | 1702.32 | 3.06 |
| 120 | 1697.44 | 1701.73 | 1701.74 | 2.14 |
| 140 | 1695.41 | 1700.78 | 1701.16 | 2.71 |
| 160 | 1694.68 | 1701.38 | 1701.37 | 2.31 |
| 180 | 1693.32 | 1699.99 | 1700.78 | 3.02 |
| 200 | 1693.81 | 1700.43 | 1700.96 | 2.40 |
| 220 | 1693.30 | 1700.25 | 1700.78 | 2.44 |
| 240 | 1692.38 | 1698.78 | 1699.01 | 2.65 |
| 260 | 1690.91 | 1698.65 | 1699.07 | 2.77 |

TABLE I: VS performance on instance *512×16_hi_1* varying the number of threads

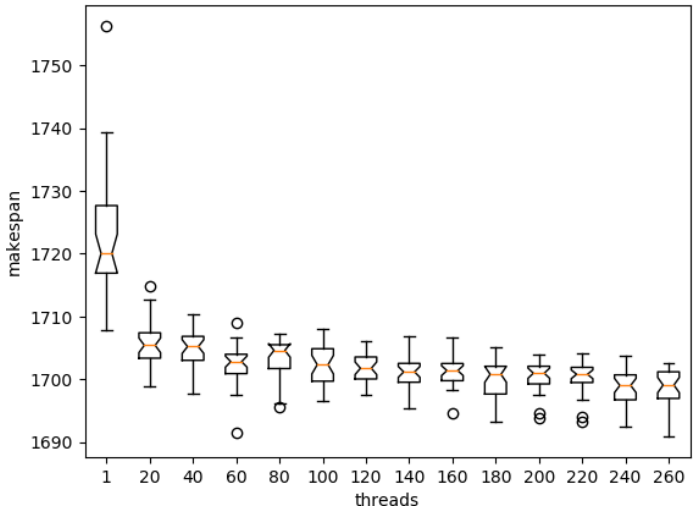
| # *threads* | *execution time* (s) | | | |
|---|---|---|---|---|
| | best | mean | median | std |
| 1 | 6.19 | 6.55 | 6.52 | 0.17 |
| 20 | 6.30 | 6.62 | 6.62 | 0.21 |
| 40 | 6.33 | 6.65 | 6.67 | 0.18 |
| 60 | 6.39 | 6.71 | 6.65 | 0.25 |
| 80 | 6.39 | 6.79 | 6.72 | 0.52 |
| 100 | 6.32 | 6.71 | 6.71 | 0.21 |
| 120 | 6.43 | 6.73 | 6.71 | 0.23 |
| 140 | 6.44 | 6.78 | 6.73 | 0.25 |
| 160 | 6.44 | 6.78 | 6.77 | 0.20 |
| 180 | 6.41 | 6.73 | 6.74 | 0.16 |
| 200 | 6.45 | 6.86 | 6.78 | 0.34 |
| 220 | 6.44 | 6.82 | 6.77 | 0.19 |
| 240 | 6.49 | 6.85 | 6.77 | 0.27 |
| 260 | 6.62 | 6.90 | 6.86 | 0.21 |

TABLE II: VS execution time on instance *512×16_hi_1* varying the number of threads

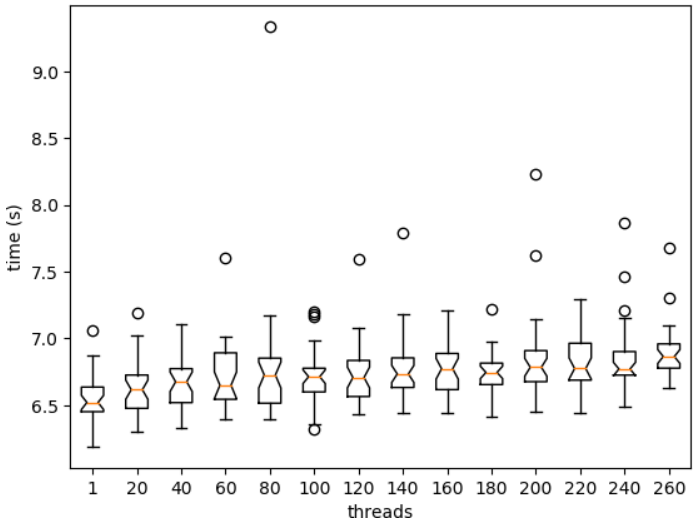
Results show that increasing the number of threads allows computing better results in terms of makespan, since more local searches are performed (1.4% average improvement over the sequential version when using 260 threads). Regarding execution times, the average increase is only 6% when varying from 1 to 260 threads. These results confirm the good scalability properties of VS when increasing the number of computing elements.

We show in Fig. 4 how the results and conclusions obtained for instance *512×16_hi_1* hold in the case of a smaller instance with low heterogeneity, namely *128×4_lo_1*. When comparing Figs. 3 and 4 we can see that the trend on the evolution of makespan of solutions when increasing the number of threads is similar in both cases. In the same way, a similar performance is obtained when considering execution times.

For the rest of the experiments presented in this work, we set the number of threads to 260, because it allows computing

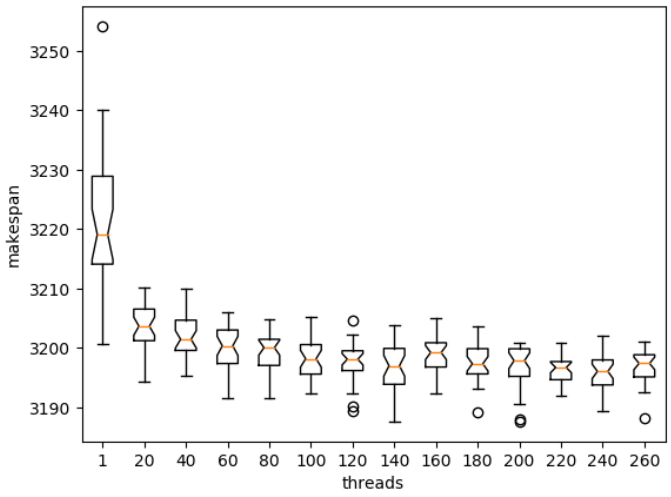



(a) makespan

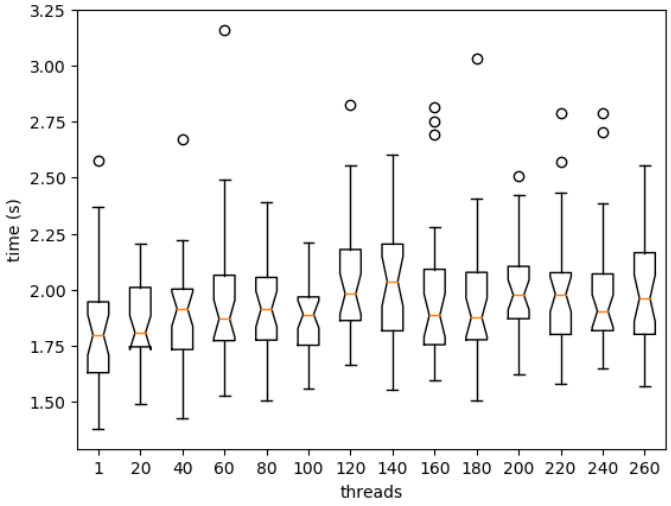



(b) execution time

Fig. 3: VS performance on instance *512×16_hi_1* varying the number of threads




(a) makespan




(b) execution time

Fig. 4: VS performance on instance *128×4_lo_1* varying the number of threads

the best results without a negative impact on the execution time.

### C. Comparison against MinMin

We present in this section a comparison of VS against MinMin. Please note that it is not the purpose of this work to provide a state-of-the-art scheduler but to analyze how accurately VS is able to learn the behavior of MinMin. As future work, we can target the use of VS to provide state-of-the art results. In that case, we would need to implement some more sophisticated heuristic for the improvement phase, as well as probably learning from other more accurate algorithms.

Table III reports the ratio between the makespan achieved by VS and MinMin, defined as the makespan of the solution of VS over the makespan of the MinMin solution. The numeric results reported include the best, mean, median, and standard deviation of the ratios achieved on each problem instance type (30 independent executions of 30 problem instances for each type). Ratios lower than 1.0 indicate an improvement of VS over MinMin. Additionally, the table also reports in the last column the number of times when VS outperformed MinMin, for every instance type. The largest instances used during the training phase of VS were of size 512×16. In order to study the scalability of VS in the problem dimension we executed VS on larger instances (1024×16) than those used for the training phase. Fig. 5 graphically summarizes the results.

VS is able to outperform MinMin on every execution of each problem instance with the same dimensions used for training. The best overall improvement considering all executions was achieved on instances of type *128×4_hi* with a 15% improvement over MinMin. On average, the best improvement

| instance type | *makespan*(*VS*)/*makespan*(*MinMin*) best | mean | median | std | #improvement |
|---|---|---|---|---|---|
| *128×4_hi* | 0.85 | 0.94 | 0.95 | 0.02 | 900/900 |
| *128×4_lo* | 0.93 | 0.96 | 0.96 | 0.01 | 900/900 |
| *512×16_hi* | 0.89 | 0.93 | 0.93 | 0.02 | 900/900 |
| *512×16_lo* | 0.92 | 0.94 | 0.94 | 0.01 | 900/900 |
| *1024×16_hi* | 0.94 | 0.98 | 0.98 | 0.01 | 853/900 |
| *1024×16_lo* | 0.95 | 0.97 | 0.97 | 0.01 | 900/900 |

TABLE III: Makespan comparison of VS against MinMin

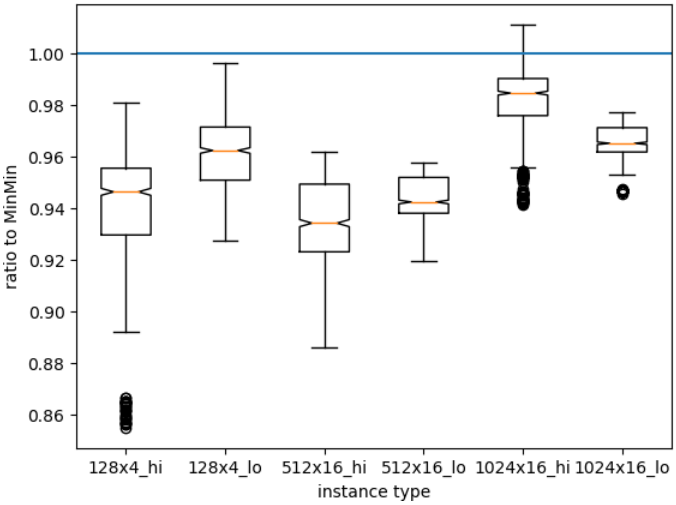



Fig. 5: Ratio of makespan: VS over MinMin. Values below 1.0 represent the cases when VS outperforms MinMin.

was achieved on instances *512×16_hi* with a 7% improvement over MinMin. Additionally, results show that VS is able to accurately solve problem instances of larger dimensions than those used during the training phase. For instances of size 1024×16 (i.e., double the size of the instances used for training) VS is still able to outperform MinMin in most cases, by up to 6% improvement.

## VII. Conclusions and Future Work

This article presented the application of the Virtual Savant paradigm to solve the heterogeneous computing scheduling problem in a massively parallel platform. Virtual Savant uses machine learning techniques to learn from a set of problem instances solved by a reference algorithm in order to generate a new program that can solve the same optimization problem in a massively parallel fashion. The results computed with the proposed solution were compared to those computed by the well-known MinMin heuristic, the algorithm used by Virtual Savant to learn from. Experimental results show that Virtual Savant is able to outperform MinMin in most problem instances, achieving up to 15% of improvement in terms of makespan. Additionally, the proposed solution shows excellent scalability properties when increasing both the computational resources and the problem dimensions.

The main lines of future work include using other machine learning methods for the classification phase, using more complex local searches and metaheuristics for the improvement phase, and applying the Virtual Savant paradigm to other related optimization problems.

## Acknowledgements

The work of Renzo Massobrio and Sergio Nesmachnow is partly supported by ANII and PEDECIBA, Uruguay. The work of Renzo Massobrio is partly supported by Fundación Carolina, Spain. Bernabé Dorronsoro acknowledges the Spanish MINECO and ERDF for the support provided under contracts TIN2014-60844-R (the SAVANT project) and RYC-2013-13355.

## References

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[2] F. Pinel, B. Dorronsoro, and P. Bouvry, "The virtual savant: Automatic generation of parallel solvers," *Information Sciences*, vol. 432, pp. 411–430, 2018.

[3] D. Treffert, "The savant syndrome: an extraordinary condition. a synopsis: past, present, future," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 364, no. 1522, pp. 1351–1357, 2009.

[4] ——, *Extraordinary People: Understanding Savant Syndrome*. iUniverse, 2006.

[5] L. Bouvet, S. Donnadieu, S. Valdois, C. Caron, M. Dawson, and L. Mottron, "Veridical mapping in savant abilities, absolute pitch, and synesthesia: an autism case study," *Frontiers in Psychology*, vol. 5, p. 106, 2014. [Online]. Available: https://www.frontiersin.org/article/10.3389/fpsyg.2014.00106

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

[7] B. Dorronsoro and F. Pinel, "Combining machine learning and genetic algorithms to solve the independent tasks scheduling problem," in *The 3rd IEEE International Conference on Cybernetics (CYBCONF-2017)*, 2017, pp. 1–8.

[8] F. Pinel and B. Dorronsoro, "Savant: Automatic generation of a parallel scheduling heuristic for map-reduce," *The International Journal of Hybrid Intelligent Systems*, vol. 11, no. 4, pp. 287–302, 2014.

[9] R. Massobrio, B. Dorronsoro, F. Palomo-Lozano, S. Nesmachnow, and F. Pinel, "Generación automática de programas: Savant Virtual para el problema de la mochila," in *XI Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, 2016, pp. 1–10.

[10] R. Massobrio, B. Dorronsoro, S. Nesmachnow, and F. Palomo-Lozano, "Automatic program generation: Virtual savant for the knapsack problem," in *International Workshop on Optimization and Learning*, 2018, pp. 1–2.

[11] R. Freund and H. Siegel, "Guest editor's introduction: Heterogeneous processing," *Computer*, vol. 26, no. 6, pp. 13–17, 1993.

[12] E. Horowitz and S. Sahni, "Exact and approximate algorithms for scheduling nonidentical processors," *Journal ACM*, vol. 23, no. 2, pp. 317–327, Apr. 1976.

[13] T. Braun, H. Siegel, N. Beck, L. Blni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810 – 837, 2001.

[14] O. Ibarra and C. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal ACM*, vol. 24, no. 2, pp. 280–289, 1977.

[15] S. Nesmachnow, H. Cancela, and E. Alba, "Heterogeneous computing scheduling with evolutionary algorithms," *Soft Computing*, vol. 15, no. 4, pp. 685–701, 2010.

[16] ——, "A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling," *Applied Soft Computing*, vol. 12, no. 2, pp. 626–639, 2012.

[17] L. Wang, H. J. Siegel, V. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, vol. 47, no. 1, pp. 8–22, 1997.

[18] F. Pinel and B. Dorronsoro, "Savant: Automatic generation of a parallel scheduling heuristic for map-reduce," *International Journal of Hybrid Intelligent Systems*, vol. 11, no. 4, p. 287302, 2014.

[19] F. Pinel, B. Dorronsoro, and P. Bouvry, "A new parallel asynchronous cellular genetic algorithm for scheduling in grids," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–8.

[20] F. Xhafa, J. Carretero, B. Dorronsoro, and E. Alba, "A tabu search algorithm for scheduling independent jobs in computational grids," *Computing and informatics*, vol. 28, no. 2, pp. 237–250, 2012.

[21] B. Duran and F. Xhafa, "The effects of two replacement strategies on a genetic algorithm for scheduling jobs on computational grids," in *Proceedings of the 2006 ACM Symposium on Applied Computing*, ser. SAC '06. New York, NY, USA: ACM, 2006, pp. 960–961. [Online]. Available: http://doi.acm.org/10.1145/1141277.1141504

[22] F. Xhafa, *A Hybrid Evolutionary Heuristic for Job Scheduling on Computational Grids*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 269–311.

[23] F. Xhafa, E. Alba, B. Dorronsoro, and B. Duran, "Efficient batch job scheduling in grids using cellular memetic algorithms," *Journal of Mathematical Modelling and Algorithms*, vol. 7, no. 2, pp. 217–236, Jun 2008. [Online]. Available: https://doi.org/10.1007/s10852-008-9076-y

[24] D. Nemirovsky, T. Arkose, N. Markovic, M. Nemirovsky, O. Unsal, A. Cristal, and M. Valero, "A deep learning mapper (DLM) for scheduling on heterogeneous systems," in *Latin American High Performance Computing Conference*, 2017, pp. 3–20.

[25] G. Zhao, Z. Shen, and C. Miao, "ELM-based intelligent resource selection for grid scheduling," in *International Conference on Machine Learning and Applications*. IEEE, 2009.

[26] Y. Wen, Z. Wang, and M. Boyle, "Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms," in *International Conference on High Performance Computing*, 2014.

[27] B. Dorronsoro and F. Pinel, "Combining machine learning and genetic algorithms to solve the independent tasks scheduling problem," in *3rd IEEE International Conference on Cybernetics*, 2017, pp. 1–8.

[28] F. Pinel and B. Dorronsoro, "Savant: Automatic Generation of a Parallel Scheduling Heuristic for Map-Reduce," *International Journal of Hybrid Intelligent Systems*, vol. 11, no. 4, pp. 287–302, 2014.

[29] P. Luo, K. Lü, and Z. Shi, "A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 67, no. 6, pp. 695–714, 2007.

[30] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27, 2011.

[31] R. Massobrio, S. Nesmachnow, and B. Dorronsoro, "Support Vector Machine Acceleration for Intel Xeon Phi Manycore Processors," in *Latin America High Performance Computing Conference.*, 2017, pp. 1–14.