Learning for Optimization with Virtual Savant

Aprendizaje para la optimización con Savant Virtual

Renzo Massobrio [1]

Recibido: Marzo 2022 Aceptado: Marzo 2022

Summary. - Optimization problems arising in multiple fields of study demand efficient algorithms that can exploit modern parallel computing platforms. The remarkable development of machine learning offers an opportunity to incorporate learning into optimization algorithms to efficiently solve large and complex problems. This article explores Virtual Savant, a paradigm that combines machine learning and parallel computing to solve optimization problems. Virtual Savant is inspired in the Savant Syndrome, a mental condition where patients excel at a specific ability far above the average. In analogy to the Savant Syndrome, Virtual Savant extracts patterns from previously-solved instances to learn how to solve a given problem in a massively-parallel fashion. In this article, Virtual Savant is applied to three optimization problems related to software engineering, task scheduling, and public transportation. The efficacy of Virtual Savant is evaluated in different computing platforms and the experimental results are compared against exact and approximate solutions for both synthetic and realistic instances of the studied problems. Results show that Virtual Savant can find accurate solutions, effectively scale in the problem dimension, and take advantage of the availability of multiple computing resources.

Keywords: machine learning; optimization; next release problem; heterogeneous computing scheduling problem; bus synchronization problem.

Resumen. - *Los problemas de optimización que surgen en múltiples campos de estudio demandan algoritmos eficientes que puedan explotar las plataformas modernas de computación paralela. El notable desarrollo del aprendizaje automático ofrece la oportunidad de incorporar el aprendizaje en algoritmos de optimización para resolver problemas complejos y de grandes dimensiones de manera eficiente. Este artículo explora Savant Virtual, un paradigma que combina aprendizaje automático y computación paralela para resolver problemas de optimización. Savant Virtual está inspirado en el Síndrome de Savant, una condición mental en la que los pacientes se destacan en una habilidad específica muy por encima del promedio. En analogía con el Síndrome de Savant, Savant Virtual extrae patrones de instancias previamente resueltas para aprender a resolver un determinado problema de optimización de forma masivamente paralela. En este artículo, Savant Virtual se aplica a tres problemas de optimización relacionados con la ingeniería de software, la planificación de tareas y el transporte público. La eficacia de Savant Virtual se evalúa en diferentes plataformas informáticas y los resultados se comparan con soluciones exactas y aproximadas para instancias tanto sintéticas como realistas de los problemas estudiados. Los resultados muestran que Savant Virtual puede encontrar soluciones precisas, escalar eficazmente en la dimensión del problema y aprovechar la disponibilidad de múltiples recursos de cómputo.*

Palabras Clave: *aprendizaje automático; optimización; problema del próximo lanzamiento; planificación en sistemas de cómputo heterogéneos; problema de sincronización de autobuses.*

[1] Doctor en Ingeniería Informática, Universidad de Cádiz/Universidad de la República, renzom@fing.edu.uy, ORCID iD: https://orcid.org/0000-0002-0040-3681

1. Introduction. - The increasing complexity of optimization problems arising in different fields of study requires algorithms that demand large computing resources [1]. Simultaneously, parallel computing has become a key piece in scientific computing, as it provides the resources needed to solve complex real-world problems that cannot be addressed using classic sequential systems [2]. Consequently, widespread parallel architectures have led to an increase in the adoption of parallel algorithms that can take advantage of the availability of multiple computing resources.

Software developers need to implement parallel programs to take profit from current architectures. This requires highly-skilled programmers that can design parallel programs from scratch or redesign legacy sequential implementations to profit from modern parallel architectures. Thus, there is an increased interest in techniques that can automatically generate elastic programs that can fully exploit highly-parallel computer platforms and scale in the number of computing resources [3]. The current growing interest in machine learning techniques comes at hand to deal with this problem.

The fields of optimization and machine learning are closely related. However, the vast majority of research has explored one direction of this relationship, i.e., optimization applied to machine learning techniques (e.g., parameter optimization in machine learning models, feature selection problems) [4]. The inverse, i.e., applying machine learning to solve optimization problems, while explored [5,6], still has plenty of room for contribution.

This article deals with VS, a novel paradigm that takes advantage of machine learning and parallel computing to address complex optimization problems [7]. VS is inspired in the Savant Syndrome, a mental condition where patients excel at certain abilities far above the average. In analogy to the Savant Syndrome, VS uses machine learning to find patterns that allow solving the problem at hand. These patterns are learned from a set of previously-solved instances of the problem. Due to its design, VS can be executed in massively-parallel computing architectures, significantly reducing execution times and effectively scaling in the problem instance.

The remainder of this document is organized as follows. Section 2 outlines the VS paradigm. Then, Section 3 presents the application and experimental evaluation of VS when solving three optimization problems. Finally, Section 4 presents the main conclusions and lines of future work.

2. Learning for optimization: Virtual Savant. - The Savant Syndrome is a rare mental condition where patients with significant mental disabilities develop certain abilities far above what would be considered average [8]. Patients with Savant Syndrome—known as savants—usually excel at a single specific activity, generally related to memory, rapid calculation, or artistic abilities. The main hypotheses state that savants learn through pattern recognition [9,10], solving problems without understanding their underlying principles.

VS is a novel technique, inspired by the Savant Syndrome, that aims to learn how to solve a given optimization problem [7]. As an analogy to the Savant Syndrome, VS proposes using machine learning techniques to find patterns that allow solving the problem at hand. These patterns are learned from a set of problem instances previously solved by one (or several) reference algorithm(s) for the problem. VS does not require knowing the code of the reference algorithms it learns from, in the same way that real-life savants are unaware of the underlying principles related to their skill. The training of VS involves partitioning the problem instance, and like savants, VS can derive global solutions by combining smaller pieces.

VS is trained using previously-solved problem instances. Learning is solely based on the input (i.e., the problem instance) and the output (i.e., the solution) computed by one or several reference algorithms. Each solved problem instance yields as many training examples as variables in the

problem being solved. Once the training set is generated, a supervised machine learning model is trained over that set.

After training, VS can solve new—unseen—problem instances by following a two-phase process comprised of prediction and improvement. In the prediction phase, the trained classifier is used to predict a solution to a new, unseen, problem instance. The output of the prediction phase is a probability distribution $P(\hat{y}^i)$ for each of the i variables in the problem. The improvement phase involves generating multiple candidate solutions following those probability distributions $P(\hat{y}^i)$. Each of these candidate solutions are refined using search procedures and heuristics (e.g., local search, greedy heuristic). Corrective functions may be included in the improvement operator to ensure that the returned solution satisfies all problem constraints. Finally, all solutions are gathered, and the best overall solution is returned.

Thanks to its design, VS can be run in a massively-parallel fashion. In the prediction phase, predictions can be made in parallel by using multiple copies of the same trained classifier, since each element in the problem is learned independently. After label probabilities are computed for each variable, multiple candidate solutions can be built and improved in parallel. Thus, VS can take advantage of available computing resources to improve its search of the solution space, leading to better solutions. Figure I outlines the complete workflow of VS when running in parallel.

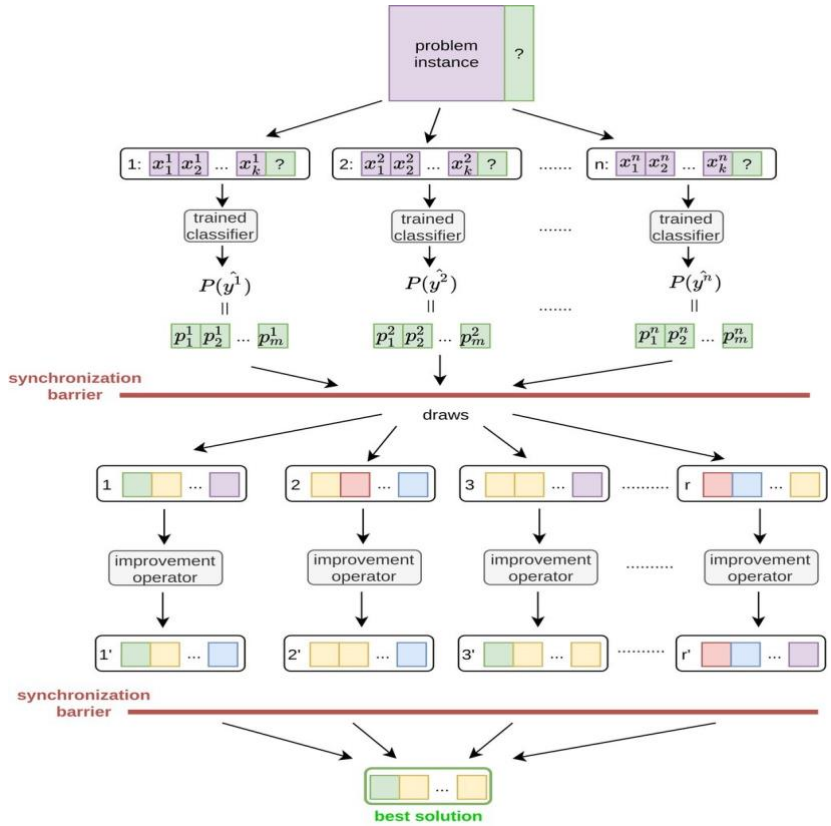

Figure I. Parallel VS workflow.

3. Applications of Virtual Savant. - This section describes the application and experimental analysis of VS over three optimization problems.

3.1. Next Release Problem. - The Next Release Problem (NRP) is a problem in Software Engineering that consists of selecting a subset of requirements or features to include in the next release of a software product, according to their expected revenues [11].

The goal is maximizing the total revenue without incurring in a total cost that exceeds the available budget.

At a lower level of abstraction, the NRP can be characterized as a specific variant of the 0/1 Knapsack Problem (0/1 KP), a classical NP-hard combinatorial optimization problem [12]. Equation 1 outlines the problem formulation, where decision variables $x_k \in \{0,1\}$ indicate whether the corresponding item is included (1) or not (0) in the knapsack. The knapsack capacity is analogous to the budget in the NRP formulation while items model the possible requirements to include in the next software release, each with an associated cost (i.e., the item's weight) and a given revenue (i.e., the item's profit).

$$argmax(\sum_{k=1}^{n} p_k x_k \mid \sum_{k=1}^{n} w_k x_k \leq C) \qquad \text{[Eq. 1]}$$

In the case of the NRP, a dataset of instances solved by an exact algorithm is used to train VS. The training vector corresponding to one requirement includes the following features: the cost of the requirement, the revenue the requirement renders, and the total budget (which is fixed for all requirements in a given problem instance). The classification label is a binary value, indicating whether the requirement is to be included (1) or not (0) in the next software release, according to the reference algorithm. The VS implementation for solving the NRP used SVMs as supervised machine learning classifiers. The training set was built using problem instances solved by the Nemhauser-Ullmann algorithm, which computes exact solutions for the NRP [13,14].

Two different proposals were implemented for the improvement phase. The first scheme applies a simple local search heuristic to each generated solution, which performs random modifications to the candidate solution to exclude or include requirements. The second improvement scheme consists of correction and improvement operators inspired by a popular greedy strategy that selects requirements to include or exclude based on their revenue/cost ratio.

A thorough study of the training and prediction phases of VS was performed, and five different variants for the improvement phase of VS were analyzed for the problem. Experimental evaluation was performed using a publicly-available benchmark of problem instances of varying size and correlation between the revenue and the cost of the requirements, which is a measure of instance difficulty. The Nemhauser-Ullmann algorithm, an exact method for the problem, was used as the reference algorithm for VS.

Firstly, a brief comparison of different feature configurations was performed, which showed no significant differences among the considered options. Secondly, a study on the training set size—required for accurate learning—was carried out. Smaller subsets of 10%, 15%, 25%, 50%, and 100% of the observations in dataset #1 in the benchmark were considered. Results showed that a training set built using 15% of the dataset was enough to make accurate predictions. Beyond that percentage, only marginal improvements were observed. Thirdly, model parameters were configured using cross-validation.

When considering only the efficacy of the prediction phase on unseen instances, VS was able to predict the exact solution with a median accuracy larger than 90% when grouping instances by their size and larger than 80% when grouping instances by their revenue/cost correlation. Detailed results are presented in boxplots in Figure II which show the accuracy achieved when grouping instances by size and correlation, respectively.

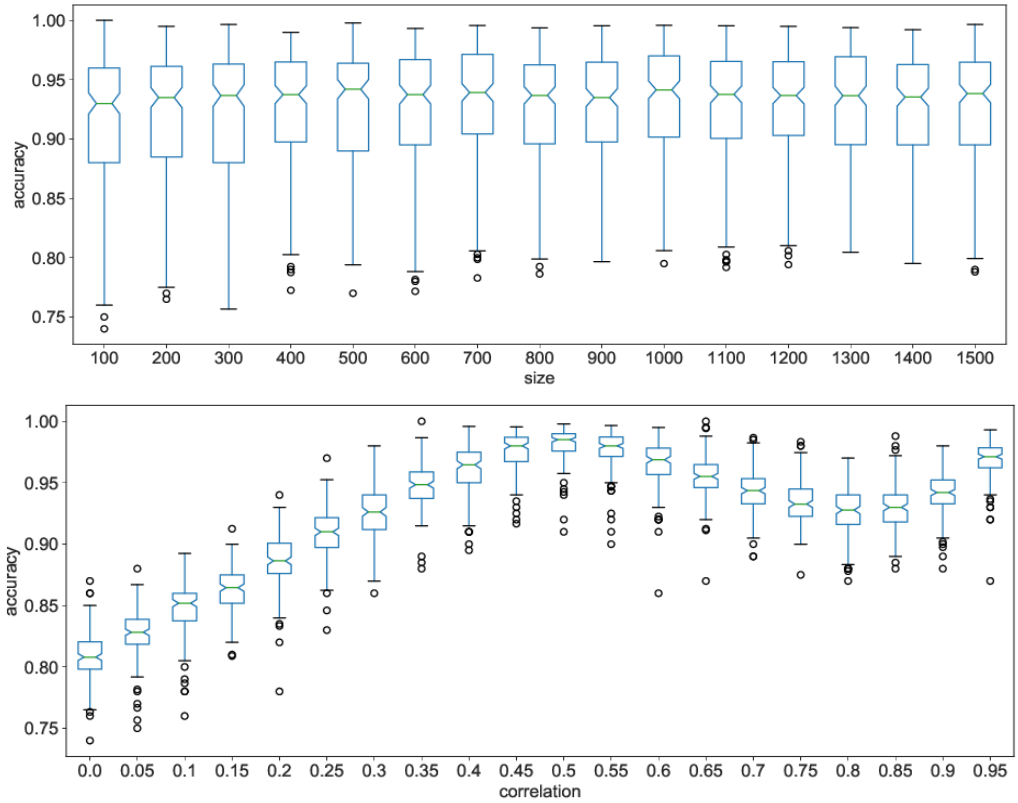

Figure II. SVM accuracy to predict the optimal solution with varying size/correlation.

The improvement phase in VS helps further refining the solutions generated in the prediction step. Experimental results showed that VS can compute highly-accurate solutions. Among the five improvement strategies devised, the simplest variant (a greedy mechanism that first corrects the solution and then improves it, based on the revenue/cost ratio of requirements) was the one that achieved the best results. The solutions computed by the VS version implementing this greedy mechanism were within 1% from the optima in all studied instances. Furthermore, VS was able to generate the optimal solution in many cases, and the computed solutions were within 0.2% (in median) of the known optima computed by the exact algorithm used as a reference. It was also observed that, interestingly, difficult instances for the reference algorithm were not necessarily difficult to solve for VS.

A detailed experimental evaluation of VS applied to the NRP can be found in [15].

3.2. Heterogeneous Computing Scheduling Problem. - The Heterogeneous Computing Scheduling Problem (HCSP) considers a heterogeneous computing system comprised of several resources (i.e., machines) and a set of tasks with variable computational requirements to be executed in the system. A task is defined as an atomic workload unit, i.e., it must be executed without interruptions and cannot be split into smaller chunks, which corresponds to a non-preemptive scheduling model. The execution time of any individual task varies from one machine to another and is assumed to be known beforehand, following a static scheduling approach. The HCSP proposes finding a task-to-machine assignment that optimizes some quality metric.

The scheduling problem addressed in this article focuses on optimizing the makespan, a well-known optimization criterion related to the productivity of a computing system. Makespan is defined as the time between the start of the first task (in the set of tasks to be executed) and the completion of the last task. Makespan is considered as a measure of productivity (i.e., throughput) of computing systems.

The mathematical formulation for the HCSP considers:

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

The HCSP proposes finding an assignment function $f : T \to M$ that minimizes the makespan, defined by Equation 2.

$$makespan = \max_{m_j \in M} \left\{ \sum_{t_i \in T, f(t_i) = m_j} ET(t_i, m_j) \right\} \quad \text{[Eq. 2]}$$

The VS implementation for the HCSP uses a custom SVM framework (xphi-LIBSVM [16]) for learning. SVMs are trained using MinMin as the reference algorithm, which is one of the most widely used methods for solving the HCSP [17]. MinMin is a two-phase greedy scheduler that greedily picks the task that can be completed the soonest. Each task in the instance 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 corresponds to the machine assigned to that task by the MinMin heuristic. Because tasks are independently assigned, VS can scale to problem instances with any number of tasks, without requiring any additional training process. A simple LS heuristic is applied over each candidate solution in VS, which 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 a subset of the least loaded ones.

The results computed by VS were compared to those computed by MinMin, the algorithm used by VS as a reference. Experimental results showed that VS outperformed MinMin in most of 180 problem instances, achieving up to 15% of improvement in terms of makespan. Detailed results are outlined in the boxplot in Figure III, which shows the ratio of makespan between VS and MinMin for different problem instances. Additionally, VS showed excellent scalability properties when increasing both the computational resources and the problem dimensions.

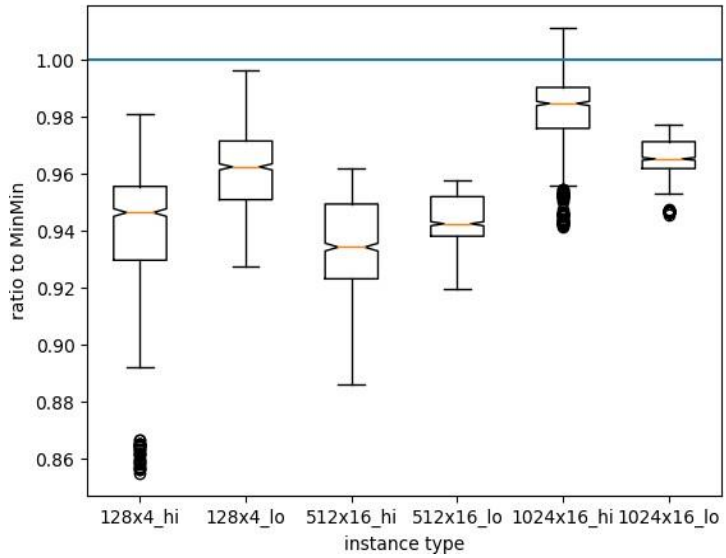

Figure III. Ratio of makespan: VS over MinMin for the HCSP.

Then, VS was evaluated on different computing platforms. Experimental evaluation over four different computing infrastructures showed that the massively-parallel design of VS allowed taking advantage of available computing resources to find accurate solutions for the HCSP. Increasing the number of parallel resources helped reducing the execution time of the prediction phase and did not increase the overall execution time, even though the computational demand of VS increases with the number of resources available. Besides, the makespan value (that evaluates the quality of the obtained results) generally decreased (i.e., improves) when increasing the number of threads.

Finally, VS was evaluated when solving very large problem instances—larger than those used during training—comprised of 32768 and 65536 tasks and 16 machines. Similar results to those obtained for the smaller instances were found, where the speedup increased with the number of cores, with a loss due to Hyper-Threading when the number of spawned threads exceeded the number of physical cores. However, speedup values were better for the largest instances studied. More details of VS applied to the HCSP can be found in [18].

3.3. Bus Synchronization Problem. - Public transportation planners often prefer network topologies comprised of few, short, and densely interconnected bus lines. This design is good from an operational point of view, because it allows operating higher bus frequencies with smaller vehicle fleets. However, it requires passengers to make more bus transfers instead of traveling directly from origin to destination. Passengers dislike transfers: studies have shown that the perceived time when waiting for a bus or when walking between bus stops can be up to 2.5 times larger than the real time spent [19]. Consequently, reducing waiting times for passengers when transferring between buses is a desirable goal from the point of view of citizens.

The Bus Synchronization Problem (BSP) consists in finding the headways of each bus line in a public transportation system, which allow maximizing the number of synchronized bus transfers.

A bus transfer is considered synchronized when the waiting time experienced at the transfer bus stop does not exceed a given threshold, defined according to the maximum time passengers are willing to wait for the transfer.

The VS implementation for the BSP uses Random Forests (RF), which are trained using the solutions computed by an Evolutionary Algorithm (EA) used as a reference [20]. The best solution found on each independent execution of the reference EA over each training instance is used to build the dataset of solved BSP instances. Two different classifiers are trained: one to predict the headway of the inbound line and another one for the headway of the outbound line.

The improvement operator consists of a simple LS that selects a bus line in each step and randomly changes its assigned headway according to a uniform distribution in the range of valid headways for the line. The change is accepted if the quality of the solution improves and is discarded otherwise. The quality of the solution is measured through a score function, which reflects the problem formulation, and is used by the reference EA as a fitness function. The score function accounts for the number of synchronized transfers and their corresponding demands.

The experimental analysis was performed using two sets of problem instances: one comprised of 130 synthetic instances and the other of 45 realistic instances modeling the public transportation network in Montevideo, Uruguay. VS was able to compute accurate solutions in both sets of problem instances. In the synthetic dataset, VS computed solutions within 1.2% of the reference EA in median when only considering the prediction phase and within 0.2% in median when including a 5000-step LS improvement operator. In the realistic instances from Montevideo, VS computed solutions 99.5% as good as the reference EA in median when considering only the prediction phase and outperformed the EA in eleven out of fifteen problem instances when adding a 5000-step LS improvement operator.

The BSP allowed studying the applicability of the VS paradigm to a real-world optimization problem and evaluating its effectiveness with respect to baseline solutions. A more complex problem decomposition than those applied for the NRP and HCSP was needed to solve the BSP. The applied problem decomposition involved training two separate machine learning classifiers. Additionally, the implementation was done using RF---in contrast with the two previous applications of VS which used SVM---showing the versatility and adaptability of VS. The experimental evaluation was performed over larger instances (in terms of the number of bus lines and synchronization nodes) than those considered in the training phase. The experimental results highlight the scalability properties of VS in terms of the problem dimension, which were also noted in the other problems addressed in this article.

The complete application of VS for the BSP is reported in [21].

4. Conclusions and future work. - This article explored Virtual Savant, a paradigm inspired by the Savant Syndrome that combines machine learning and parallel computing to solve complex optimization problems. Implementations of VS were developed and evaluated for solving three optimization problems: i) the NRP, a well-known problem from software engineering that was modeled as a 0/1 KP; ii) the HCSP, a classic task scheduling problem relevant in modern computing infrastructures; and iii) the BSP, an optimization problem related to public transportation networks.

Due to its flexible design, VS can use different machine learning algorithms for the training and prediction phases. In the studied problems two different classifiers were used: SVMs for the NRP and HCSP; and RF for the BSP. Similarly, the design of VS is flexible in terms of the algorithm(s) used as a reference. On the studied problems, both exact and approximate algorithms were used as a reference.

The scalability of VS was evaluated both in terms of the problem dimension and in the use of computational resources. Regarding the problem dimension, VS was evaluated over problem instances much larger than those seen during training. This is a very interesting feature of the design of VS, since it allows solving problem instances that may not be tractable for the algorithm used as a reference. The scalability in the use of computational resources was evaluated on the HCSP. For this problem, four different computing platforms were considered, including shared- and distributed-memory architectures. Results showed that the massivelyparallel design of VS allows efficiently using available computing resources.

The work presented in this article was intended to be a step forward towards bringing closer the machine learning and optimization research fields, but many lines of work remain to be addressed.

Regarding the training and prediction phase of VS, other machine learning algorithms need to be considered. One promising line of work is to incorporate ensemble learning to VS. These classifiers could even be trained using different optimization algorithms as a reference or over different sets of solved instances.

Regarding the improvement phase of VS, other operators should be considered and evaluated. Tailored improvement operators that incorporate problem-specific techniques could be easily included in VS. Additionally, another interesting line of work would be to use the pool of candidate solutions generated in the prediction phase of VS to initialize a population-based metaheuristic, e.g., an EA.

Other optimization problems, potentially with harder constraints or dependencies between the problem variables, should also be considered. Furthermore, addressing multiobjective optimization problems is an interesting line of future work. For this purpose, a domain decomposition approach could be implemented, using a linear combination of the objective functions and training a set of classifiers with different weights.

5. References

[1] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti, A., and Protasi, M. (2012). Complexityand approximation: Combinatorial optimization problems and their approximability properties. Springer Science & Business Media.

[2] Golub, G. H. and Ortega, J. M. (2014). Scientific computing: an introduction with parallel computing.Elsevier.

[3] Darte, A., Robert, Y., and Vivien, F. (2012). Scheduling and automatic parallelization. Springer Science& Business Media.

[4] Golub, G. H. and Ortega, J. M. (2014). Scientific computing: an introduction with parallel computing. Elsevier.

[5] Vlastelica, M., Paulus, A., Musil, V., Martius, G., and Rolıń ek, M. (2020). Differentiation of blackboxcombinatorial solvers. In International Conference on Learning Representations.

[6] Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer Networks. In Advances in Neural InformationProcessing Systems 28, pages 2692–2700.

[7] Pinel, F., Dorronsoro, B., Bouvry, P., and Khan, S. (2013). Savant: Automatic parallelization of ascheduling heuristic with machine learning. In World Congress on Nature and Biologically Inspired Computing, pages 52–57.

[8] Treffert, D. (2006). Extraordinary People: Understanding Savant Syndrome. Iuniverse.

[9] Pring, L. (2005). Savant talent. Developmental medicine and child neurology, 47(7):500–503.

[10] Heaton, P. and Wallace, G. L. (2004). Annotation: The savant syndrome. Journal of child psychologyand psychiatry, 45(5):899–911.

[11] Bagnall, A., Rayward, V., and Whittley, I. (2001). The next release problem. Information and SoftwareTechnology, 43(14):883–890.

[12] Kellerer, H., Pferschy, U., and Pisinger, D. (2004). Springer.

[13] Nemhauser, G. L. and Ullmann, Z. (1969). Discrete Dynamic Programming and Capital Allocation.Management Science, 15(9):494–505.

[14] Harman, M., Krinke, J., Medina-Bulo, I., Palomo, F., Ren, J., and Yoo, S. (2014). Exact scalablesensitivity analysis for the next release problem. ACM Transactions on Software Engineering and Methodology, 23(2):1–31.

[15] Massobrio, R., Nesmachnow, S., Palomo-Lozano, F., and Dorronsoro, B. (2021). Virtual savant as ageneric learning approach applied to the basic independent next release problem. Applied Soft Computing, 108:107374.

[16] Massobrio, R., Nesmachnow, S., and Dorronsoro, B. (2018c). Support Vector Machine Acceleration forIntel Xeon Phi Manycore Processors. In High Performance Computing Latin America Conference, pages 277–290.

[17] Luo, P., Lü, K., and Shi, Z. (2007). A revisit of fast greedy heuristics for mapping a class of independenttasks onto heterogeneous computing systems. Journal of Parallel and Distributed Computing, 67(6):695–714.

[18] de la Torre, J. C., Massobrio, R., Ruiz, P., Nesmachnow, S., and Dorronsoro, B. (2020). Parallel virtualsavant for the heterogeneous computing scheduling problem. Journal of Computational Science, 39:101048.

[19] International Transport Forum (2014). Valuing Convenience in Public Transport. OECD Publishing.

[20] Nesmachnow, S., Muraña, J., Goñi, G., Massobrio, R., and Tchernykh, A. (2020). EvolutionaryApproach for Bus Synchronization. In High Performance Computing Latin America Conference, pages 320–336.

[21] Massobrio, R., Nesmachnow, S., Muraña, J., and Dorronsoro, B. (2022). Learning to optimize timetables for efficient transfers in public transportation systems. Applied Soft Computing, 119:108616.