Virtual Savant as a generic learning approach applied to the basic independent Next Release Problem

Renzo Massobrio[a,b], Sergio Nesmachnow[b], Francisco Palomo-Lozano[a], Bernabé Dorronosoro[a]

[a] *Universidad de Cádiz, España*
[b] *Universidad de la República, Uruguay*

Abstract

This article presents how Virtual Savant (VS) can be used to automatically learn, from an exact algorithm, how to solve the basic independent Next Release Problem in a quick and accurate way. This variant of the Next Release Problem (NRP) is in essence a 0/1 Knapsack Problem and VS is applied to solve the underlying optimization problem. VS is a generic problem-solving approach based on machine learning and heuristics, that works by mimicking how a reference program produces solutions to problem instances. Essentially, VS learns how to generate solutions to a given problem from a reference algorithm and a training set of instances, and it is able to efficiently solve new problem instances by using the acquired knowledge. In this paper, an exact optimizer is used as a reference algorithm. Hence, we are using VS to learn from optimal solutions, which helps to reduce the approximation error inherent to the learning process. We compare five versions of VS (differing in the heuristics they implement) on a large benchmark, composed of problems with different sizes and difficulties. For the best VS configuration, which also has the lowest computational complexity, computed solutions differ less than 1% from the optima in the worst case. Therefore, VS succeeds in learning how to solve the problem under study, and it does so in a highly efficient way, exempting the programmer from having a deep knowledge of the problem domain or highly specialized parallel programming skills.

Keywords: Basic Independent Next Release Problem, Knapsack Problem, Machine Learning, Optimization

[0]Corresponding author: Renzo Massobrio - renzom@fing.edu.uy

1. Introduction

Requirements Engineering (RE) can be described as the process of formulating, documenting, and maintaining a set of requirements during an engineering design process [1]. RE is an important discipline in many areas of engineering, and it is specially useful in software engineering (SE), where defining and analyzing software requirements is crucial to properly define a system [2]. Particularly in early stages, or during the inception of the new release of a product, requirements can be classified into two groups. One group comprised of core or mandatory requirements that cannot be dispensed with and will likely be present in any version of a product for technical, strategical, or policy compliance reasons. A second group, composed of optional requirements, usually reflecting different product features, which may be demanded by stakeholders. The selection of the optional requirements to implement is a key problem in RE.

The Next Release Problem (NRP) is a related problem in SE [3]. In essence, the NRP consists of selecting a subset of requirements or features to include in the *next release* of a software product, taking into account their expected *revenues* and other factors, as which requirements are demanded by the stakeholders (users or customers) and their perceived relevance. This selection problem is constrained in practice by the resources available for the development process, as requirements have implementation *costs* and they cannot exceed a given *budget*. There are also potential interactions between requirements, like dependencies or precedence constraints. NRP is a relevant problem when it comes to developing and maintaining modern complex software systems, and it is one of the most popular problems for which search-based RE and SE approaches have been applied. Search-based approaches often apply heuristic and metaheuristic search methods, e.g., evolutionary algorithms [4], for solving RE and SE problems formulated as optimization problems.

Among the different NRP variants proposed in the seminal work by Bagnall et al. [3], the basic independent NRP considers no dependencies between requirements, disjoint sets of requirements for different stakeholders and, unless otherwise stated, a single objective function to maximize. The objective function takes into account the revenues of the requirements and the preferences of the stakeholders for the next release of a software product. 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 basic independent NRP can be characterized as a specific variant of the 0/1 Knapsack Problem (KP), a classical $\mathcal{NP}$-hard combinatorial optimization problem [5].

This article proposes solving the basic independent NRP modeled as a 0/1 KP using Virtual Savant (VS), a novel optimization method that is able to automatically learn from programs that solve optimization problems. VS combines machine learning and parallel computing techniques to learn how to solve a given problem based on a reference algorithm, which is considered as a black-box [6, 7]. For this purpose, a machine learning classifier is trained using a set of problem instances solved by the reference algorithm. After the training phase is completed, VS is able to efficiently and accurately solve new, previously unknown, and even larger problem instances (i.e., with a larger number of variables) than the reference algorithm, applying parallel computing techniques. Three research questions are investigated here:

> *RQ 1* – Is VS able to solve basic independent NRP instances of varying size and difficulty?
>
> *RQ 2* – What is the contribution of each phase in VS to the overall computed results?
>
> *RQ 3* – How does VS scale when using multiple computing resources?

This work significantly extends our previous conference paper [8], where some preliminary results on the application of VS to the 0/1 KP were presented. The main objective of this work is not to provide some state-of-the-art algorithm for 0/1 KP, but to show that it is possible to learn how to solve the problem with high accuracy using machine learning techniques. The main contributions of this extended version are:

1. A thorough study of the training phase of VS.
2. Results for VS on basic independent NRP problem instances of varying size and difficulty.
3. A comparison of different alternatives for the improvement phase of VS on the problem instances under study.
4. A novel method to correct and improve solutions and its evaluation.
5. A complexity analysis.
6. A study on the parallel scalability of VS.

The remainder of this article is organized as follows. Section 2 presents a detailed account of VS, the target problem, the design and implementation of VS for the target problem, and a complexity analysis. The main published works related to the topics of this paper are summarized in Section 3. Then, the experiments performed and the results obtained are discussed in Section 4. Finally, we present our conclusions and main lines of future research in Section 5.

2. VS for the the basic independent NRP

This section reports on VS and how it can be applied to solve the basic independent NRP, modeled as a 0/1 KP.

2.1. An overview of VS

VS is a novel paradigm that aims to learn from a reference algorithm how to solve a given optimization problem in a massively parallel fashion [6, 7]. VS is inspired by the *savant syndrome*, an extremely rare mental condition in which patients with significant mental disabilities develop certain specific abilities far in excess of what would be considered normal [9].

Patients with savant syndrome (*savants*) often excel at a single specific activity, generally related to memory, rapid calculation, or artistic abilities. The underlying thought processes of savants are not yet fully understood by researchers. However, the main hypothesis is that savants learn through pattern recognition [10]. This mechanism allows them to solve problems without understanding their underlying principles. For instance, some patients are able to enumerate large prime numbers or discriminate between prime and non-prime numbers, without understanding what a prime number is.

In an analogy to the savant syndrome, VS proposes using machine learning techniques to find patterns that allow solving a given problem. The training is done in a supervised fashion, using a set of problem instances, and their solutions, built by some reference algorithm. Once the training phase is completed, VS is able to emulate the reference algorithm in order to solve previously unknown, maybe bigger, problem instances, without the need of any further retraining.

VS works in two steps: *prediction*, where a trained classifier is used to predict a solution to a given (unseen) problem instance, and *improvement*, where the predicted solution is further refined using search procedures and

heuristics to improve its quality. Each phase can be executed in a massively-parallel fashion, following a map-reduce approach. This allows for a reduction of execution times and a better exploration of the solution space: the higher the amount of available resources, the higher the number of independent processes that can be run in parallel.

2.2. The basic independent NRP as a 0/1 KP

The NRP deals with the selection of a subset of requirements or features to be included in the next release of a particular software product. Each requirement has an associated implementation cost and an expected revenue. The revenue is usually related to the preferences of the stakeholders, who may also demand that certain requirements are present. Requirements may have dependencies between them. The goal is to find which requirements maximize the total revenue, subject to a budget limit for their total cost. There are many variants that are often referred to as NRP problems in the literature. A taxonomy appears in the seminal work by Bagnall et al. [3].

The basic independent NRP is the simplest variant, and can be modeled by the 0/1 KP, a classic $\mathcal{NP}$-hard combinatorial optimization problem [5]. The importance of this variant of the NRP problem stems from its relative simplicity and from the fact that many other variants (though not every conceivable variant) can be reduced or transformed to it [3, 11, 12].

This problem can be mathematically formulated as a 0/1 KP as follows. Given a set of n items, each with a profit p_k and a weight w_k, and the capacity C of the knapsack, the 0/1 KP consists in finding a subset of items that maximizes the total profit, without exceeding the knapsack capacity. Eq. 1 shows the exact problem formulation, where decision variables $x_k \in \{0, 1\}$ indicate whether the corresponding item is included or not in the knapsack. The knapsack capacity is analogous to the budget in the basic independent NRP formulation, while items model the requirements to be considered for inclusion in the next software release, each with an associated cost (the weight of the item) and a given revenue (the profit of the item).

$$\arg\max \left\{ \sum_{k=1}^{n} p_k x_k \;\middle|\; \sum_{k=1}^{n} w_k x_k \leqslant C \right\} \tag{1}$$

As mentioned above, more elaborate NRP variants exist in the literature. For instance, dependencies among requirements are considered in the original general NRP formulation [3]. Many algorithms addressing these variants

work by transforming their instances into one or several instances of the basic independent NRP, thus effectively resorting to the simplest version to solve them under the hood. Consequently, the basic independent NRP is relevant even when considering more complex scenarios.

2.3. Design

Fig. 1 outlines the training process used in VS for the 0/1 KP. Each item in the problem instance is treated as a separate observation during the training phase of VS. Therefore the training vector for the machine learning classifier includes the weight and profit of a given item (arrow 1), as well as the capacity of the knapsack (arrow 2). The classification label is a binary value, indicating whether the item is included or not in the knapsack in the solution computed by the reference algorithm, as indicated by arrow 3 in the figure.

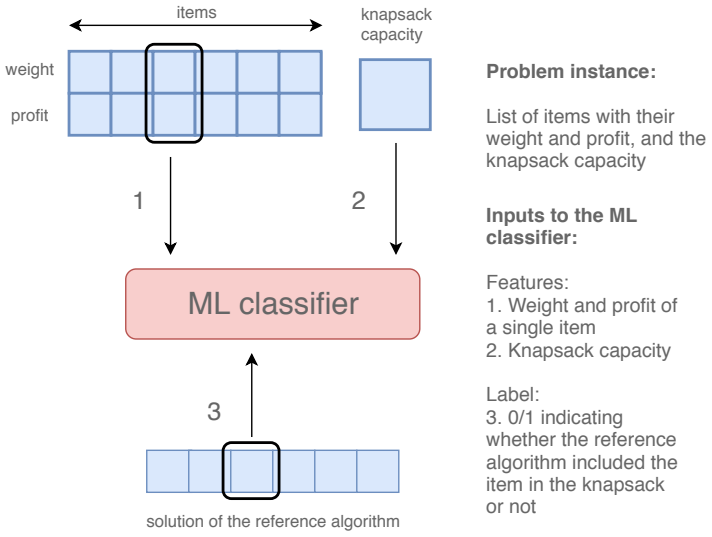

Figure 1: Outline of the training phase of VS.

Thanks to the learning scheme proposed, each problem instance contributes to the learning phase with as many observations as the number of items in the instance. This allows drastically reducing the number of reference solutions required in the learning process. Another key advantage of this approach is that the machine learning classifier can decide on the assignment of each variable without knowing the assignments of the others, so

the solution can be built in parallel by as many independent processes as the number of items in the problem instance. Each independent process is just a replication of the same machine learning classifier.

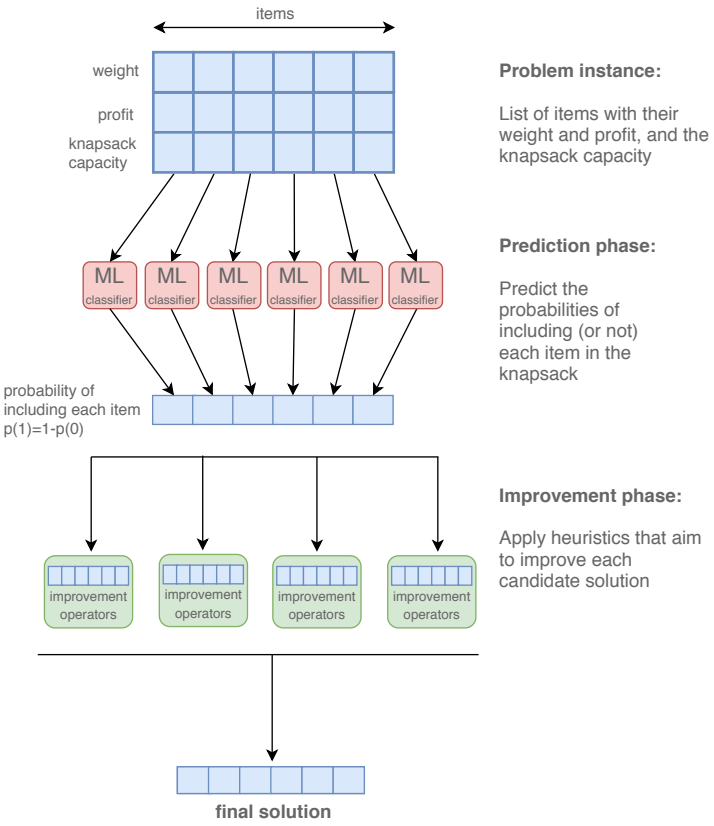

Figure 2: Outline of the prediction and improvement phases of VS to solve the given problem.

After training the machine learning classifier, VS is able to handle new, previously unknown, and even bigger instances than those used for learning. This process, outlined in Fig. 2, involves two phases: *prediction* and *improvement*. In the prediction phase, VS receives as input an unknown problem instance to solve. Since the machine learning classifier was trained considering each item individually, several copies of the same classifier can be spawned, forming a pool, splitting the new problem instance and making predictions for each variable in parallel using this pool. Potentially, each item in the problem instance can be handled by a different copy of the same

classifier. The output of each classifier is the probability of including the corresponding item in the knapsack. This architecture, where predictions are made independently for each item, provides VS with a high degree of parallelism. The results computed by each classifier in the pool are gathered to form a single vector that holds the probability of including each item in the knapsack.

During the improvement phase, the probability vector built after the prediction phase is used to generate different candidate solutions. There are several possibilities to build these solutions. In this work, we follow a generic method that performs random samples, guided by the probabilities of including each item, as reported by the machine learning classifier. Each generated solution is then subject to an *improvement operator*, which aims to modify the solution to achieve better results. This phase also plays an important role in specific problem instances where the prediction phase may perform poorly. The improvement phase is also subject to massive parallelism, e.g., one candidate solution can be generated and improved per computing resource available. It is worth noting that the generated solutions could be unfeasible, i.e., they may not satisfy the knapsack capacity constraint. Therefore, during the improvement phase it is necessary to include a *correction operator* to ensure the feasibility of the computed solution. In this work, several alternatives for improvement and correction operators are taken into account, which are described in the following section.

Algorithm 1 outlines a pseudo-code of the VS design applied to the 0/1 KP. The loop at lines 3 to 5 corresponds to the prediction phase, where the probability of including each item is predicted based on the weight and profit of each item and the knapsack capacity. The loop at lines 6 to 9 corresponds to the improvement phase of VS, where candidate solutions are generated based on the probabilities computed in the prediction phase and are refined using improvement operators. Finally, the best generated solution is returned (line 10). It is straightforward to parallelize both loops.

2.4. Implementation

The implementation details of each phase of the VS for the 0/1 KP follow.

2.4.1. Prediction phase

Our VS implementation for solving the 0/1 KP uses Support Vector Machines (SVMs) as supervised machine learning classifiers. The training set

Algorithm 1: VS applied to the 0/1 KP.

```
   input: instance
 1 probability_vector ← []
 2 candidate_solutions ← []
 3 foreach item in instance do
 4 |   probability_vector[item] ← predict(item.weight, item.profit,
   |     instance.knapsack_capacity)
 5 end
 6 for i ← 0 to candidate_solutions.size − 1 do
 7 |   candidate_solutions[i] ← generate_solution(probability_vector)
 8 |   improvement_operators(candidate_solutions[i])
 9 end
10 return best(candidate_solutions)
```

is composed of problem instances solved using the Nemhauser-Ullmann algorithm as a reference algorithm, which computes exact solutions for the 0/1 KP [13, 11]. We adopted the SVM implementation provided by the well-known LIBSVM framework [14]. A specific fork of this implementation was created to adapt it for its execution on many-core architectures so that it is suitable for the VS paradigm [15]. The Radial Basis Function (RBF) kernel was used to map vectors to a higher dimensional space.

2.4.2. Improvement phase

Two different proposals were implemented for the improvement phase, which are described next and are compared and evaluated in the experimental evaluation outlined in Section 4.

Local search and corrections. The first proposed scheme for the improvement phase is to apply a simple local search (LS) heuristic to each generated solution. This LS operator simply performs random modifications to the candidate solution to exclude or include items that are present or not in the knapsack, respectively. In each step of the LS, a randomly-chosen bit of the candidate solution is flipped, the new solution is evaluated using a score assignment function, and the LS continues from that solution if an improvement is found. Algorithm 2 describes the score assignment function used to guide the LS. The function considers a solution with total weight W, total profit P, and overweight $O = W - C$, where C is the knapsack capacity. W,

P, and O are scaled using the minimum and maximum weight and profit values in the problem instance. A constant $f > 0$ is used as a penalty factor for overweighted solutions, while $m \in (0, 1)$ is used to define the maximum overweight for a solution to be considered by the LS.

The score assignment function is devised to allow the LS to explore unfeasible solutions with a total weight that exceed the capacity constraint by up to a factor of m, but penalizing such solutions in order to guide the search towards feasible solutions.

Algorithm 2: Score assignment for solutions during the local search.

input: solution, instance
1 scale(W, P, O, C, instance)
2 **if** $O \leq 0$ **then return** P
3 **else if** $O \leq m \cdot C$ **then return** $P - f \cdot O$
4 **else return** $-O$

Since the solution returned by the LS operator might be unfeasible, it is necessary to include a correction operator to guarantee feasibility. Two different correction approaches were considered:

1. Correction by profit (Cprofit): iteratively removes the item with lowest profit until the total weight of the solution is not greater than the knapsack capacity.
2. Correction by weight (Cweight): iteratively searches for items with weights not lower than the overweight of the solution, removing the item with the lowest weight among them. If no item satisfies this condition, the item with the overall highest weight is removed.

Greedy correction and improvement. An alternative operator for the improvement phase, inspired by a popular greedy strategy for knapsack problems, was considered. First, while the total weight of the candidate solution exceeds the capacity of the knapsack, the item with the lowest profit/weight ratio is iteratively removed, in order to correct unfeasible solutions. Then, while the total weight of the candidate solution is within the knapsack capacity (i.e., there is still room to add items to the knapsack), the greedy improvement scheme iteratively adds the leftover items that fit, one by one, in descending order of profit/weight ratio.

2.5. Complexity analysis

The computational complexity of the VS algorithm when solving new problem instances depends on the complexity of each of its two phases (i.e., prediction and improvement), which are analyzed next.

The prediction phase involves solving the equations that define the SVM with RBF kernel for each item in the new problem instance. Making the prediction of whether an item should be included or not in the knapsack has a complexity in $\mathcal{O}(d \cdot k)$, where k is the number of support vectors in the trained model and d is the dimension of the input vector. Thus, the total complexity of this phase is in $\mathcal{O}(d \cdot k \cdot n)$, where d is the number of features, k is upper-bounded by the number of training samples[1], and n is the size of the problem instance being solved. In the experiments described in Section 4, $k = 8046$ and $d = 3$, since there are three features: item weight, item profit, and knapsack capacity.

The improvement phase involves applying a specific improvement operator, thus, its complexity depends on the specific improvement operator applied. In the case of the greedy approach (which achieves the overall best results, as reported in Section 4) the procedure consists of two steps. In the first step, the procedure iteratively removes the item with the lowest profit/weight ratio from the knapsack as long as the capacity is exceeded. In the second step, the operator iteratively inserts the item with the highest profit/weight ratio as long as the capacity is not exceeded. This is efficiently implemented by presorting the item list by density (i.e., the profit/weight ratio of each item) and then, in linear time, iterating over that sorted list twice: from left to right, to remove items, and from right to left, to include leftover items. Therefore, the complexity of the greedy improvement operator is dominated by the polylogarithmic presorting stage, and is in $\mathcal{O}(n \log n)$ in the worst case.

Overall, the complexity of VS is in $\mathcal{O}(d \cdot k \cdot n + n \log n)$ and its fixed-parameter complexity is in $\mathcal{O}(\max\{k \cdot n, n \log n\})$, as d can be assumed a fixed small value.

[1]The upper-bound is only reached in the extreme case that each sample is used as a support vector.

3. Related Work

This section presents an overview of previous research on the 0/1 KP and the basic independent NRP, as well as a brief survey on machine learning methods applied to solving optimization problems.

3.1. 0/1 KP and the basic independent NRP

0/1 KP is a widely studied problem in Operations Research. Dantzig [16] studied this problem and called it "the knapsack problem", recognizing its importance from an integer programming perspective. In 1969, Nemhauser and Ullmann [13] proposed an exact algorithm to solve 0/1 KP using dynamic programming. The algorithm was applied to solve a well-known problem in the field of management: allocating capital within constrained budgets. The basic capital allocation problem can be modeled as a 0/1 KP considering investment opportunities, or *projects*, as the items to select for inclusion in the knapsack and the available budget for investments as the knapsack capacity. Each project has a required investment and an expected return of investment, which are analogous to the weight and profit of items in the 0/1 KP formulation. Thus, the capital allocation problem consists in finding a portfolio of projects in which to invest within a given constrained budget.

A wealth of algorithms and techniques have been developed for the 0/1 KP. The books by Martello and Toth [17] and, more recently, Kellerer et al. [5] are authoritative sources on knapsack-related problems and algorithms, including both the underlying theory and a discussion of practical aspects.

Bagnall et al. [3] introduced NRP as an optimization problem of industrial importance in RE. Several versions of the problem are discussed, where the objective function aims to evaluate the satisfaction of the stakeholders and the goal is to maximize it subject to different constraints. Given the limitations of exact algorithms, approximation algorithms and heuristics were also used. The simplest version of NRP is the basic independent NRP used in our work. Bagnall et al. also discuss how some more general versions can be transformed into the basic independent NRP.

Harman et al. [11] studied the basic independent NRP modeled as a 0/1 KP and considered the impact of uncertainty in the estimation of requirements, presenting a sensitivity analysis tool to help stakeholders to deal with inaccuracies when estimating the costs of requirements in a project. The underlying idea is to identify those requirements for which a small deviation in the cost estimation leads to a high impact in the value of the optimal

solution. Once those problematic requirements are identified, a decision-maker could potentially assign more resources to the estimation of sensitive requirements. Uncertainty is tackled by producing several basic independent NRP instances for each original instance. An optimized implementation of the original algorithm from Nemhauser and Ullmann [13] was used to solve 0/1 KP instances. This optimized implementation is also used in our work as a reference algorithm during the training phase of VS.

Veerapen et al. [12] solved different variants of NRP, including single and bi-objective formulations, using integer linear programming (ILP). A set of transformations to simplify some variants of NRP is discussed. The optimization software CPLEX was used to solve the ILP instances and it was observed that the performance of this approach has improved significantly since Bagnall's seminal work. This improvement does not just stem from the dramatic increase in computing power, but also from steady advances in ILP solvers. In the approach proposed by Veerapen et al., the goal is to provide an exact optimization method capable of managing instances of reasonable size. Since the problem is $\mathcal{NP}$-hard, execution times can grow dramatically for large problem instances. In contrast, VS aims to providing a good approximation method that is fully scalable with fast and predictable execution times, once the training phase has been completed.

3.2. Machine learning for optimization problems

Few articles which apply machine learning techniques to solve combinatorial optimization problems were found in the literature review.

Vinyals et al. introduced Pointer Networks (*ptr-nets*), a model based on recurrent neural networks (RNNs) [18]. *Ptr-nets* are trained by observing solved instances of a problem and, similarly to the VS paradigm, can also deal with problem instances of varying size. The proposed approach is evaluated when solving three discrete combinatorial optimization problems: finding planar convex hulls, computing Delaunay triangulations, and solving the planar Travelling Salesman Problem (TSP). Experimental results show that the ptr-nets were able to find competitive results in problem instances larger than those seen during the training phase.

Later, Bello et al. [19] outperformed the results of Vinyals et al. when solving TSP by using reinforcement learning over the ptr-nets architecture, with negative tour length as a reward signal. Experimental evaluation was performed on TSP instances of up to 100 cities. To outline the applicability of the proposed approach to other optimization problems, the authors studied

the 0/1 KP. For this problem, the experimental evaluation was performed on instances of up to 200 items. The proposed approach was able to solve all studied problem instances to optimality. In our paper, we evaluate the proposed VS framework over instances of up to 1500 items with different correlations between the weight and profit of items.

More recently, Hu et al. [20] extended the model proposed by Vinyals et al. [18] and applied it to solve the three-dimensional bin packing problem, an optimization problem related to the 0/1 KP. A deep reinforcement learning approach is used to predict the sequence of items to pack in a bin. The specific empty spaces in which items are placed and the orientation of each item are computed by heuristic methods. The proposed approach outperformed a specific heuristic for the problem during the experimental analysis. Improvements of 5% on average over the baseline heuristic were achieved for the problem instances studied. Reinforcement learning is also used by Li and Malik to automatically generate optimizers for unconstrained continuous optimization problems [21]. The resulting algorithm is an iterative process that is guided by the gradient of the objective function in the last iterations.

Dai et al. proposed taking advantage of the fact that, when solving optimization problems, the structure tends to remain nearly unchanged, with problem instances only varying in the specific data [22]. The authors proposed a framework that combines reinforcement learning with graph embedding to solve optimization problems over graphs. The proposed approach constructs the solution in a greedy fashion and takes advantage of a graph embedding network called *structure2vec* to incorporate the graph structure into the learning process. Experimental results show that the proposed approach is useful for problems where the graph structure is important to compute the overall solution. Experimental results on real-world datasets are also outlined for the three studied graph problems.

Selsam et al. proposed *NeuroSAT*, a solver for the propositional satisfiability problem (SAT) based on Message Passing Neural Networks (MPNN) [23]. The proposed approach relies on training a MPNN using only satisfiability of the problem instance as a supervision bit. Experimental evaluation shows that the network is able to predict satisfiability after several iterations. A post-hoc procedure based on clustering is used to derive the boolean values of each variable based on the activations in the neural network. The experimental evaluation showed that NeuroSAT is able to solve larger instances than those used during training, albeit demanding a larger number of iterations and with a significant drop in efficiency. The proposed approach was applied

to solve multiple graph problems modeled as SAT instances to show the wide applicability of NeuroSAT. No execution time or performance metrics are reported either for the training or prediction experiments of NeuroSAT.

Two recent works have applied reinforcement learning to solve scheduling problems. Waschneck et al. applied Deep Q Network (DQN) to the Job Shop Scheduling problem [24]. Experimental analysis over a small factory simulation showed that the proposed approach achieves solutions of comparable quality to those computed by an expert but is not able to outperform classic dispatching heuristics. More recently, Wang et al. applied a similar approach to study the multi-objective workflow scheduling problem [25]. A DQN reinforcement learning model is used to schedule computing tasks in a cloud environment with the goals of minimizing task completion time and user's cost. The experimental evaluation over problem instances based on real data showed that the proposed approach is able to outperform several baseline heuristics and metaheuristics for the problem.

Some works in the literature have used machine learning to address the NRP. Araújo and Paixão [26] proposed an architecture that combined machine learning with an interactive genetic algorithm to solve the NRP. In their proposed approach, a machine learning model is used to replace the human interaction needed to model the requirements engineer preferences. The goal is to eventually replace the human interactions needed for the execution of the genetic algorithm. In their model, user preference is learned based on the inputs made in the first iterations of the interactive genetic algorithm. The machine learning model is only proposed in this article. The model implementation and experimental evaluation are reported in a more recent article [27]. The proposed approach differs significantly to the one proposed in our article. In their approach, the underlying optimization problem is solved using the genetic algorithm, i.e., machine learning is solely applied to predict the revenue of the items in the instance based on the users preference. In our approach, VS is used to learn how to solve the underlying optimization problem using an exact algorithm as reference.

The VS paradigm was introduced by Pinel and Dorronsoro, where it was applied to a task scheduling problem [6]. This work was later extended by evaluating different reference algorithms during the training phase and including different procedures in the improvement phase [28, 7]. Additionally, the parallel capabilities of the VS paradigm were evaluated in a many-core computing environment [29]. The application of VS to the 0/1 KP was initially studied by Massobrio et al. [8]. The main difference between VS

and the surveyed works is the massive parallel capabilities that VS offers. Because VS predicts each variable in the solution vector independently, it can efficiently use multiple computing resources and be deployed on massively-parallel computing architectures. This paper extends our previous work on VS by conducting deeper studies on the training phase and evaluating its performance, proposing new strategies for the improvement phase too.

4. Experimental analysis

This section reports the experimental analysis of the proposed VS for the basic independent NRP, modeled as a 0/1 KP. The problem instances used in the experiments are described and the results of each phase operating on VS (training, prediction, and improvement) are reported and discussed.

4.1. Problem instances

The proposed VS for the basic independent NRP was trained and evaluated using a standard benchmark of problem instances. These instances were solved to optimality using the Nemhauser-Ullmann algorithm.The benchmark is comprised of problem instances with different size and Pearson correlation between cost and revenue of requirements. Pearson correlation is a measure that is usually applied to characterize the difficulty of solving a problem instance [11]. The benchmark includes a total of 50 datasets, each with 300 instances with varying size and Pearson correlation. The benchmark is publicly available, and it can be downloaded from *The Next Release Problem* website at University of Cádiz (ucase.uca.es/nrp).

Within each dataset, instance sizes vary from 100 to 1500 requirements (stepsize: 100). For each instance size, Pearson correlation between cost and revenue of requirements varies from 0.0 to 0.95 (stepsize: 0.05). Figs. 3 and 4 show two examples of the relation between cost and revenue in instances with different Pearson correlation.

Out of the 50 datasets of the benchmark, dataset #1 was used for observations during the training phase of VS, datasets #2 through #5 were used to define the size of the training set and for feature selection, and datasets #6 to #15 were used for the evaluation of VS. Problem instances and experimental results are available at www.fing.edu.uy/~renzom/vs_nrp.tar.gz.

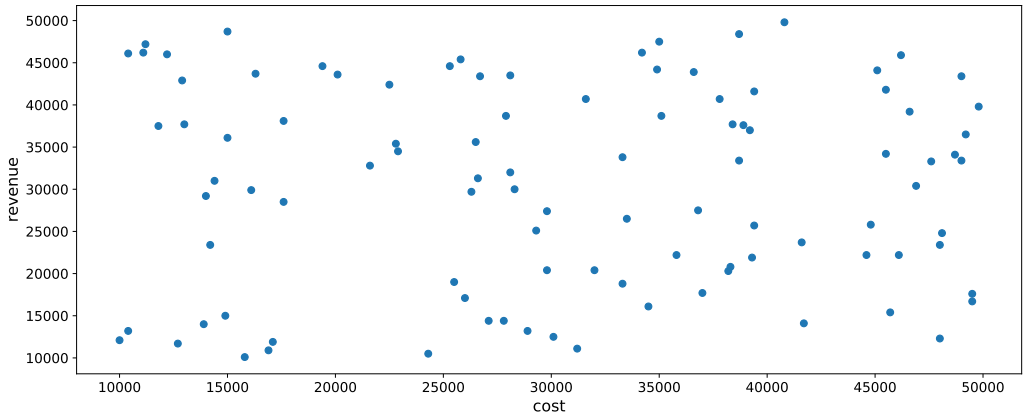

Figure 3: Relation between cost and revenue in sample basic independent NRP instance with Pearson correlation = 0.00.

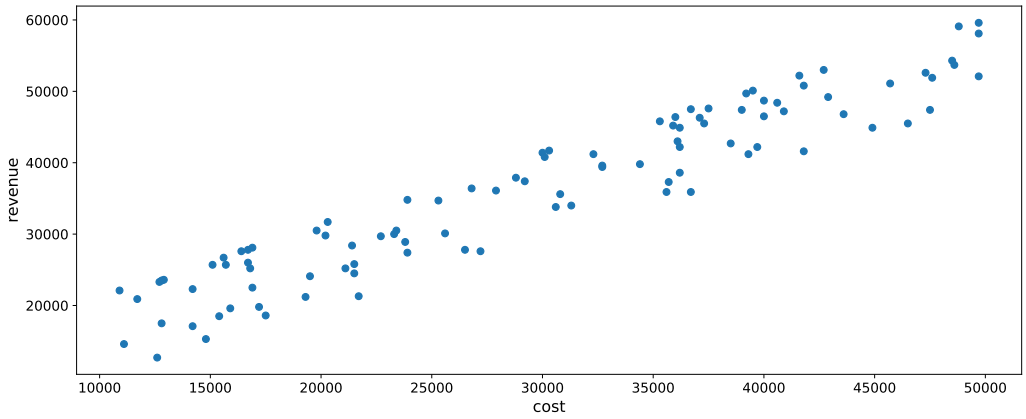

Figure 4: Relation between cost and revenue in sample basic independent NRP instance with Pearson correlation = 0.95.

4.2. SVM training

Training experiments were performed following an incremental approach, focused on studying different features of the problem, relations between them, and parameter values of the training method. The goal of the study was to determine the configuration of parameters and combination of features that allows achieving the best accuracy in the prediction phase of VS. A three-step analysis was performed, which follows.

First step: study of the input features. In the first step, three different feature configurations were evaluated for the input vectors of the SVM.

- *C1*: Item weight, item profit, and knapsack capacity (3 features).
- *C2*: Item weight, item profit, and ratio between knapsack capacity and the total number of items in the instance (3 features).
- *C3*: Item weight, item profit, knapsack capacity, and total number of items in the instance (4 features).

The average accuracy (i.e., the percentage of accurate predictions of the trained SVM compared to the optimal solution) for each dataset was evaluated using the three feature configurations. Results showed minor differences among the different feature configurations considered, with accuracy values between 89.4% and 89.7%. Consequently, configuration *C1* was chosen for the remainder of the experimental analysis due to its simplicity.

Second step: study of the number of training observations. A study of the number of observations used during the training phase of the SVM was performed. Table 1 shows the prediction accuracy when varying the number of observations of dataset 1 used during training. Training with 15% of dataset 1 results in a 31% improvement in prediction accuracy when compared to using only 10%. However, increasing the size of the training set beyond 15% of dataset 1 results in marginal accuracy improvements. Since training times increase drastically with larger training sets, a SVM trained with 15% of dataset 1 is used for the rest of the experimental evaluation.

Table 1: SVM accuracy for different training set sizes.

	Number of observations (% of the total)				
	252000 (100%)	126000 (50%)	63000 (25%)	37800 (15%)	25200 (10%)
dataset-2	89.6%	89.5%	89.4%	89.4%	58.0%
dataset-3	89.5%	89.4%	89.3%	89.3%	57.9%
dataset-4	89.6%	89.5%	89.3%	89.3%	58.0%
dataset-5	89.7%	89.6%	89.4%	89.4%	58.0%

Third step: parameters configuration. SVM and RBF kernel parameters were configured (parameters C and γ, respectively). For this purpose, cross-validation (CV) was performed over a set of 5000 observations randomly selected from dataset 1. A 5-fold CV was performed with the following candidate values: $C \in [2^{-5}, 2^{15}]$ and $\gamma \in [2^{-15}, 2^{3}]$ (step-size: 2^2). Results indicated that the best accuracy values are computed with $C = 8192$ and $\gamma = 0.5$. Average accuracy values before and after CV are reported in Table 2. An improvement of $\sim 1\%$ is achieved after the parameters configuration, in datasets #2 through #5.

Table 2: Average accuracy before and after parameter configuration using cross validation.

	Before CV	After CV
dataset-2	89.4%	90.4%
dataset-3	89.3%	90.5%
dataset-4	89.3%	90.5%
dataset-5	89.4%	90.5%

4.3. Prediction phase

Once the training phase was completed, VS was evaluated using the unseen problem instances from datasets #6 through #15. The experimental evaluation was performed using the best configuration of features, training size, and parameters, which are summarized in Table 3.

Table 3: Configuration for the experimental evaluation.

Parameter	Value
Feature vector	$\langle$item weight, item profit, knapsack capacity$\rangle$
Training set size	37800
C	8192
γ	0.5

Firstly, we focus the study on the prediction phase of VS. Boxplot in Fig. 5 shows the accuracy achieved by the SVM; problem instances are grouped by size. The accuracy of VS is defined as the percentage of variables that are correctly predicted, when comparing to the optimal solution provided by

the Nemhauser-Ullmann algorithm. Analogously, Fig. 6 shows the accuracy values achieved when grouping instances by the weight/profit Pearson correlation of their items. The notches in the boxes display the variability of the median between samples. If the notches of two boxes are not overlapped, then it means that there are statistical significant differences in the data with 95% confidence.

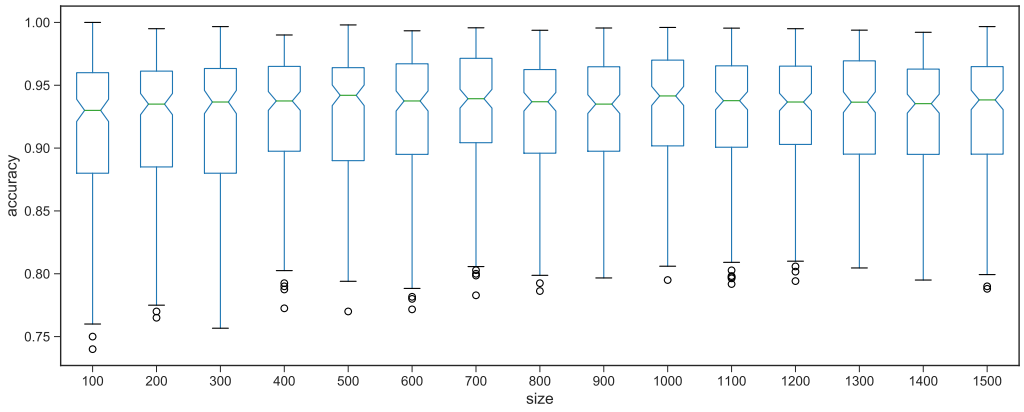

Figure 5: SVM accuracy to predict the optimal solution with varying problem size.

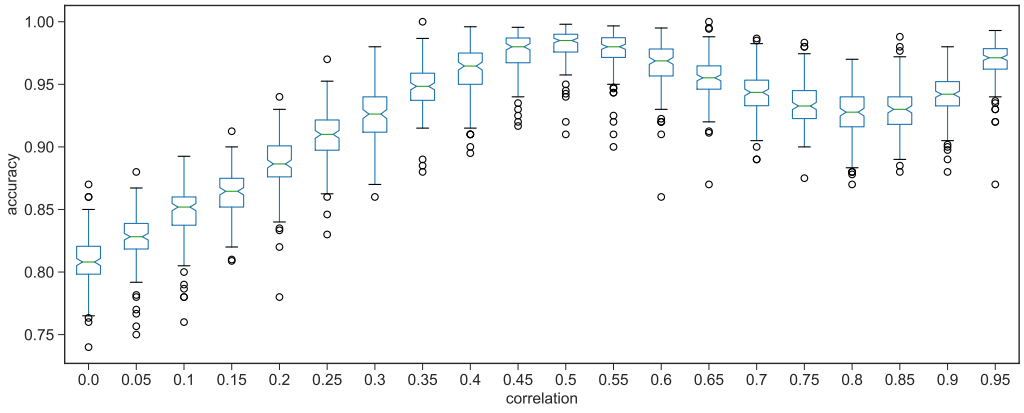

Figure 6: SVM accuracy to predict the optimal solution with varying correlation.

The median accuracy achieved by the SVM is over 90% for all problem sizes studied. Additionally, no significant differences are appreciated among instances of different sizes. This might be explained due to the prediction

scheme of VS, where each item is considered individually. When grouping instances by weight/profit correlation, it is noticeable that instances with a correlation of 0.5 are the simplest to predict for the SVM, with a median accuracy over 97%. In the worst case, when the weight/profit correlation is 0.0, the median accuracy of the SVM is still higher than 80%. Overall, the mean accuracy for all problem instances studied was 92.3% with a standard deviation of 5.3%. The worst prediction was made for an instance with size 100 and correlation 0.0 with a prediction accuracy of 74.0%. The best prediction accuracy was achieved in two instances of size 100, with correlations 0.35 and 0.65, where the optimal solution was predicted (i.e., prediction accuracy was 100%).

In order to answer the second research question (RQ 2), we need to evaluate the contribution of each phase in VS to the overall computed results. In order to measure the contribution of the prediction phase of VS, the quality of the predicted solutions was evaluated prior to the application of the improvement operators. As explained in Section 2.4, the predicted solutions might be infeasible (i.e., when the sum of the weights of the included items exceeds the knapsack capacity). Thus, in some cases, it is necessary to apply a correction scheme to ensure solution feasibility. For this experiment we applied the greedy correction described in Section 4, without applying any of the improvement operators. Results show that no corrections at all were needed for over 58% of the studied problem instances. On average, only 2.2% of the items in a given problem instance needed correction in order to guarantee solution feasibility. Figures 7 and 8 show the ratio to the optimum achieved during the prediction phase of VS with varying problem size and correlation, respectively. The ratio to the optima is defined as the quotient between the profit of the computed solution and the profit of the optimal solution.

Results show that the prediction phase of VS is able to compute good quality solutions in most problem instances. On average, solutions computed based only on the prediction phase of VS differ in 9% from the known optima. No significant differences are noticed among the quality of solutions when grouping problem instances by size, with predicted solutions within 10% of the optimal value in median. When grouping problem instances by correlation, results show that the SVM predictions are able to compute better quality solutions for instances with larger correlation. This result is consistent with the previous analysis of prediction accuracy.

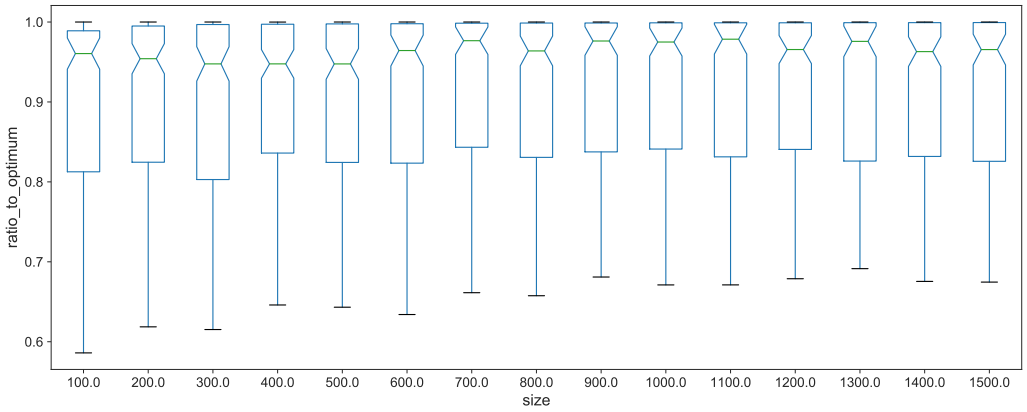

Figure 7: Ratio to optimum of SVM predictions with varying problem size.

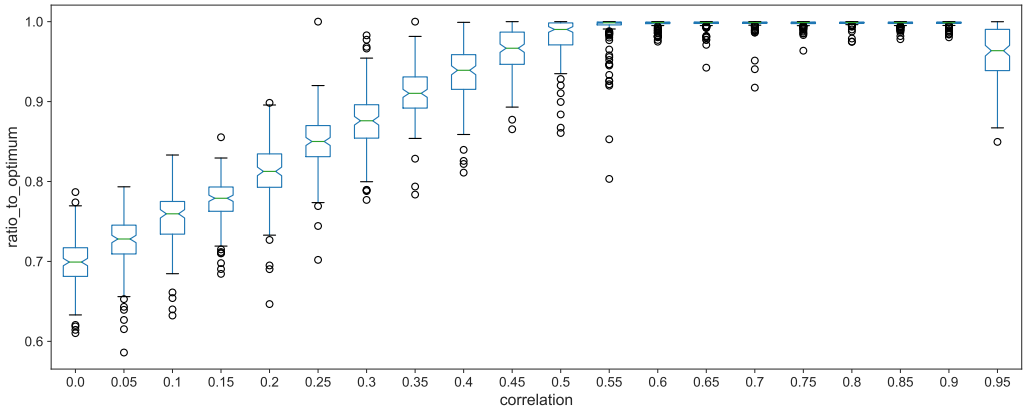

Figure 8: Ratio to optimum of SVM predictions with varying correlation.

4.4. Improvement phase

The experimental results when using the different strategies for the improvement phase are presented next.

4.4.1. Corrections and local search

The ratio to the optima is used as a metric to evaluate the results computed by VS. Figs. 9 and 10 show the average ratio to optima achieved when applying only the correction by profit (Cprofit), only the correction by weight (Cweight), the local search followed by the correction by profit (LS+Cprofit), and the local search followed by the correction by weight (LS+Cweight). Av-

erage results are presented when grouping instances by size (Fig. 9) and by profit/weight correlation (Fig. 10). Results correspond to 30 independent executions of each problem instance. The local search was executed with 1000 steps. The evaluation function of the local search used the following parameters: $m = 0.2$ and $f = 2$, thus allowing the LS to explore solutions that violate the knapsack capacity constraint in up to 20%.

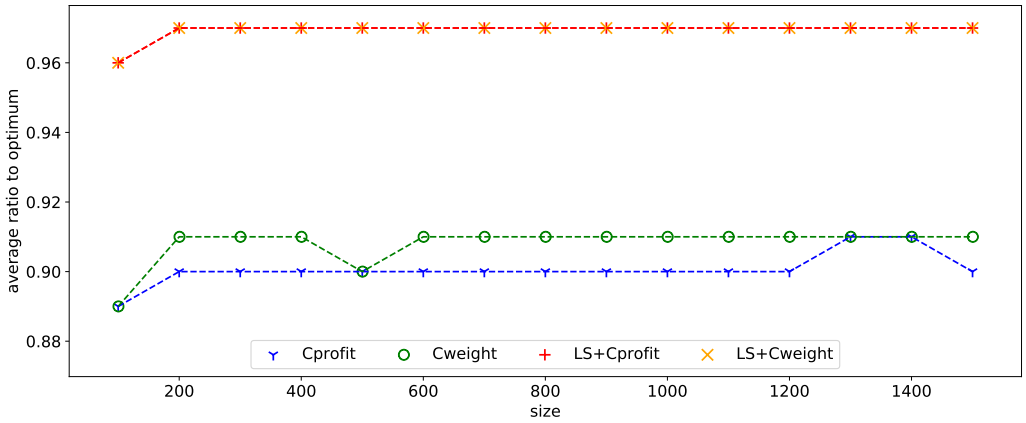

Figure 9: Average ratio to optimum with varying problem size.

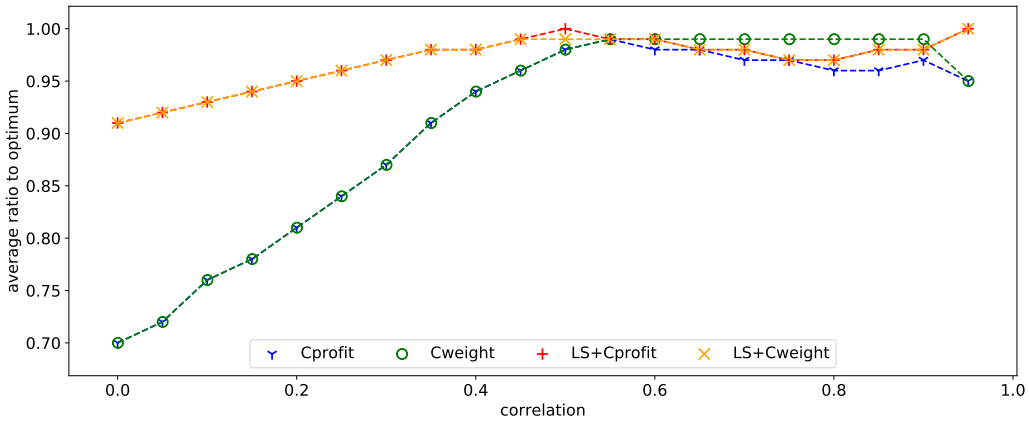

Figure 10: Average ratio to optimum with varying weight/profit correlation.

Results show that VS is able to compute accurate results for the studied instances. When grouping instances by size, solutions computed by VS are,

in average, only 3% worse than the optimal solutions. If only the correction schemes are applied (without a local search) VS is still able to compute accurate solutions, within 10% from the optima. If we group instances by correlation, it is particularly interesting to notice that VS is able to solve to optimality instances with correlations of 0.5 and 0.95. In average, VS differs by 3.15% and 3.20% from the known optima when using LS+Cprofit and LS+Cweight, respectively, for all correlations. Finally, it is worth noting the low effect (slightly negative in some cases) the local search has on the quality of solutions with respect to the predicted one by VS for correlations between 0.5 and 1.0.

4.4.2. Greedy correction and improvement

Finally, the results achieved by VS when using the greedy correction and improvement are presented. Fig. 11 presents the achieved ratio to optimum when grouping instances by size. Similarly, Fig. 12 shows the computed results when grouping instances by the weight/profit correlation of their items.

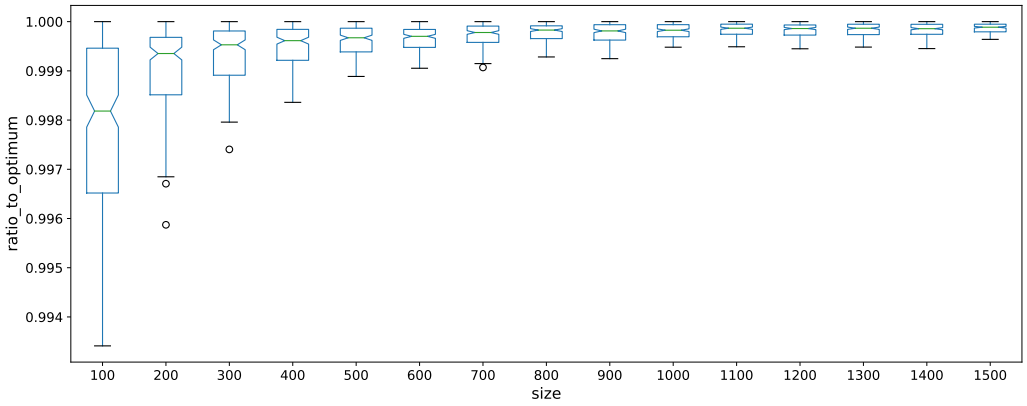

Figure 11: Average ratio to optimum with varying problem size using greedy correction and improvement.

The results achieved when using the greedy and correction improvements are within 1% from the optima in all the instances under study (in the comparisons in this section we are considering the fitness value). On average, computed results are within 0.04% from the optima. In the worst case, for an instance of size 100 and correlation 0.7, the solution computed by VS differs in only 0.66% from the known optima. Additionally, the optimal solution was computed for 5.5% of all problem instances studied. VS solved

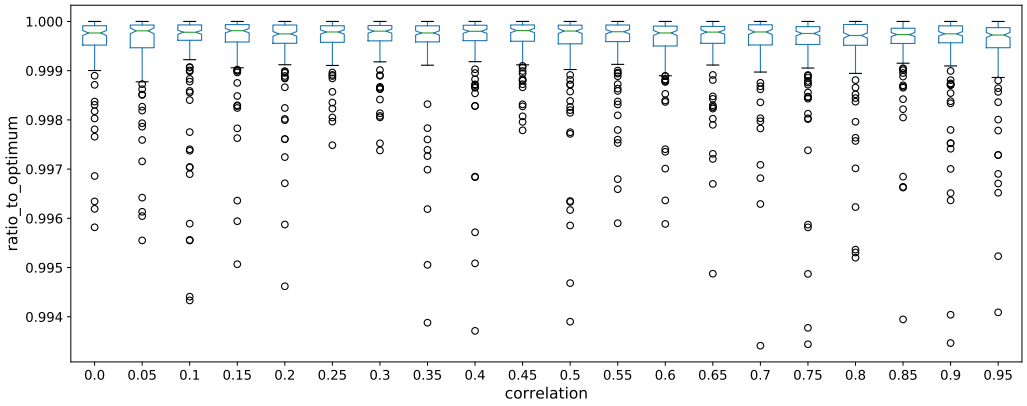

Figure 12: Average ratio to optimum with varying weight/profit correlation using greedy correction and improvement.

to optimality instances of all sizes and correlations. The group of instances having size 100 and correlation 0.40 is the one with the highest number of instances that were optimally solved. When looking at instances by size, VS performs better on larger instances. The median ratio to optima differs in less than 0.2% from the known optima for all problem sizes studied. No significant differences can be noticed among problem instances when grouping by profit/weight correlation. It is worth noting that this correction and improvement scheme is based on a well-known greedy heuristic for the 0/1 KP, thus including valuable domain-specific information in the VS, unlike the LS and corrections presented earlier. These results positively answer the first research question (RQ 1), showing that VS is able to accurately solve problem instances of different size and difficulty.

In order to completely address the second research question (RQ 2), we need to evaluate the contribution of the prediction phase of VS. For this purpose, we applied the greedy correction and improvement operators, which achieved the best overall results, to randomly-generated initial solutions. The goal of this experiment was to show that the prediction phase plays an important role in computing a good quality initial solution which can then be improved using the greedy heuristics. Figures 13 and 14 show the ratio to the optimum achieved when applying the greedy correction and improvement to randomly-generated solutions on problem instances with varying size and correlation, respectively.

Results achieved when starting from a randomly-generated solution are,

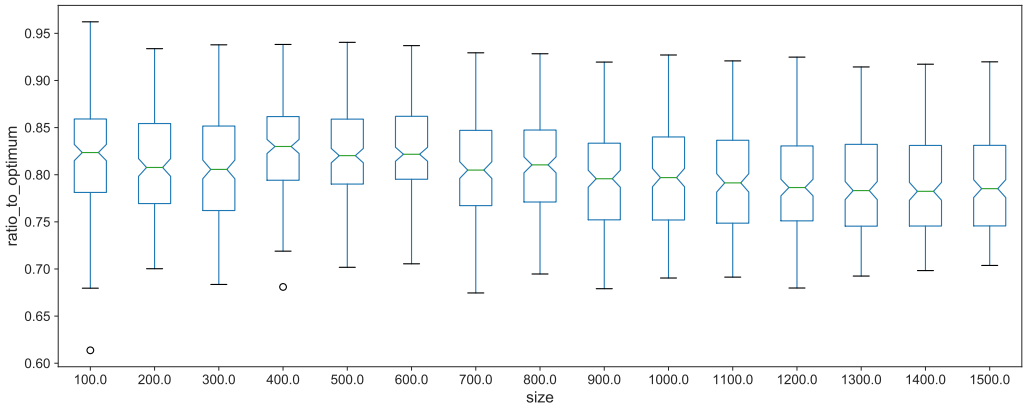

Figure 13: Average ratio to optimum with varying problem size starting from random solution and using greedy correction and improvement.

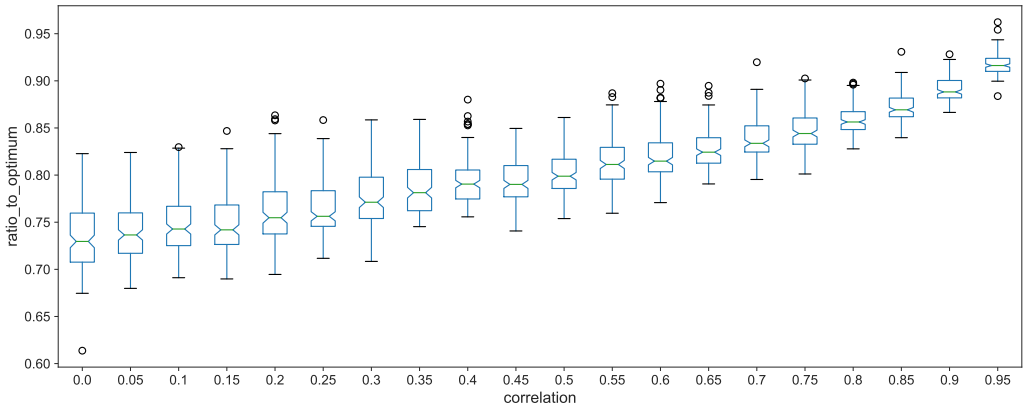

Figure 14: Average ratio to optimum with varying weight/profit correlation starting from random solution and using greedy correction and improvement.

on average, 20% away from the optima. The importance of both phases of VS can be noticed when comparing these results against those reported in Figures 11 and 12.

4.5. Parallelism in VS

As outlined in Section 2.4, the workflow of VS allows for massive parallelization. The following experiments were performed to characterize the execution time and parallel capabilities of VS when solving very large problem instances, answering the last research question posed (RQ 3). A set

of problem instances was generated comprised of instances with 100 000 to 1 000 000 requirements. Four instances were generated for each problem size, considering a step of size 100 000. These instances were solved using the proposed VS approach on a Xeon Phi many-core computing platform.

Figure 15 shows the execution time of VS when varying the number of threads. Each boxplot outlines the execution times over the four instances of a given problem size. Results are condensed in Figure 16, which aggregates the speedup achieved on all instances when varying the number of threads used. The speedup is defined as t_1/t_n where t_1 is the execution time when using only one processing unit and t_n corresponds to the execution time when spawning n threads. Similarly, Figure 17 outlines the efficiency achieved when varying the number of threads, which is equal to the speedup divided by the number of threads used. Finally, Figure 18 shows the execution time with respect to the size of the problem instance when using 64 cores (similar results were obtained for other numbers of cores).

Results show that good execution time improvements are achieved when using more than one thread on all studied instances. However, execution times stop improving when using more than 64 cores and even a significant negative impact when using more threads can be noticed. This could be explained due to the fact that the Xeon Phi has 68 physical cores. Therefore, when using more threads, some of the CPU resources are shared among the threads, incurring in a noticeable overhead. Results in Figure 18 show that the execution time of VS scales linearly with the size of the problem instance. These results demonstrate that VS is able to scale when using multiple computing resources, answering our final research question (RQ 3).

5. Conclusions and future work

This article presents how the VS paradigm can be used to efficiently solve a complex problem in Requirements Engineering (RE). VS represents a novel optimization method, in which machine learning techniques are used to learn how to solve a given problem from a reference algorithm. The process is comprised of three stages, namely training, prediction, and improvement; and it is subject to massive parallelization. The reference algorithm is executed to provide solutions to randomly-chosen parameterized instances and the instance-solution pairs are used to train VS. The main advantages of VS are: i) it can solve larger instances that may not be tractable by the reference algorithm; ii) it is massively parallelizable; iii) it is flexible with regards

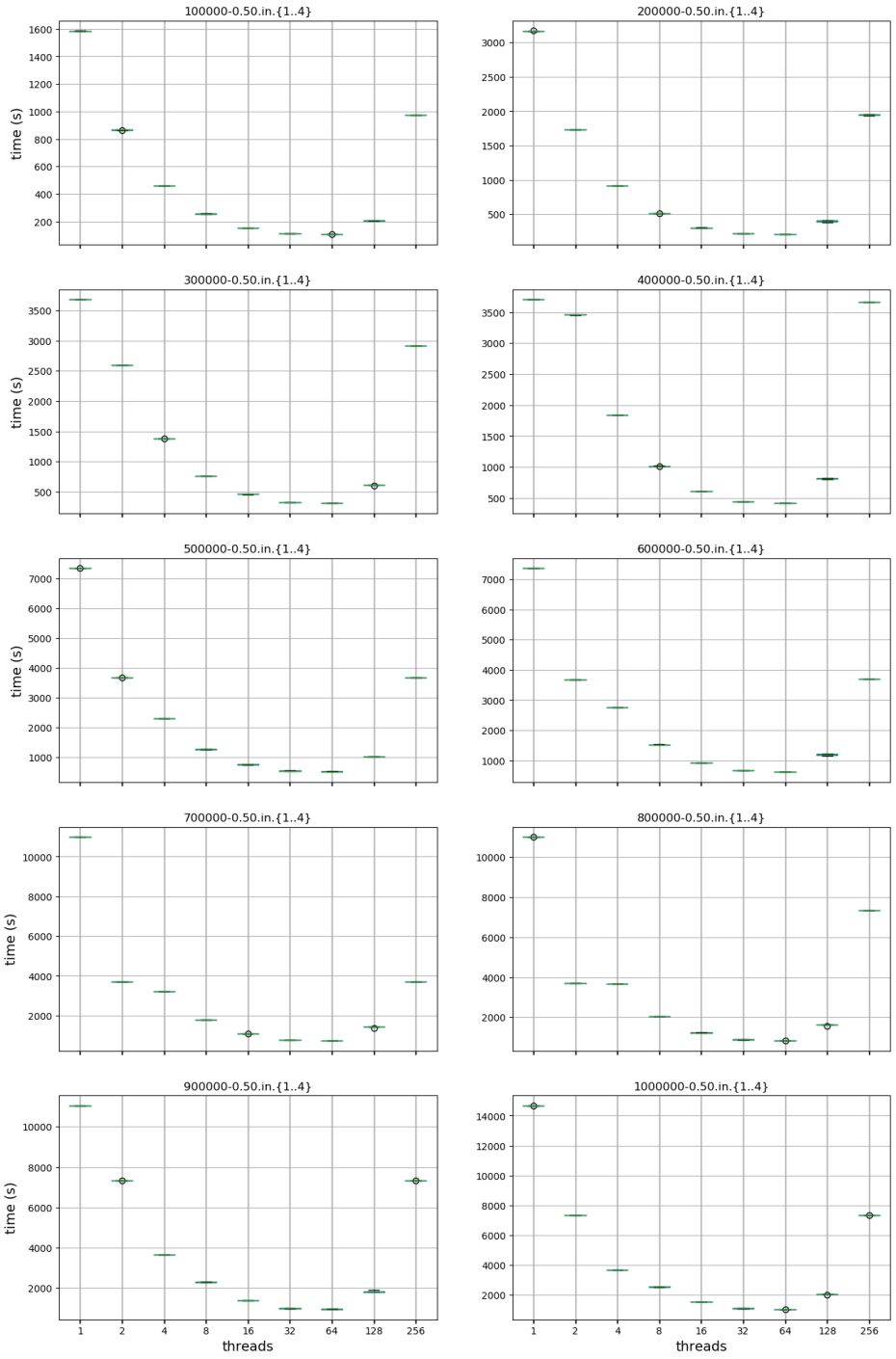

Figure 15: Execution times with varying number of threads.

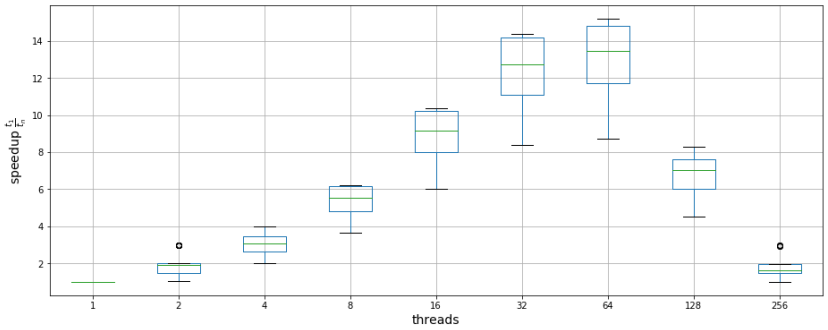

Figure 16: Average speedup with varying number of threads on all studied instances.

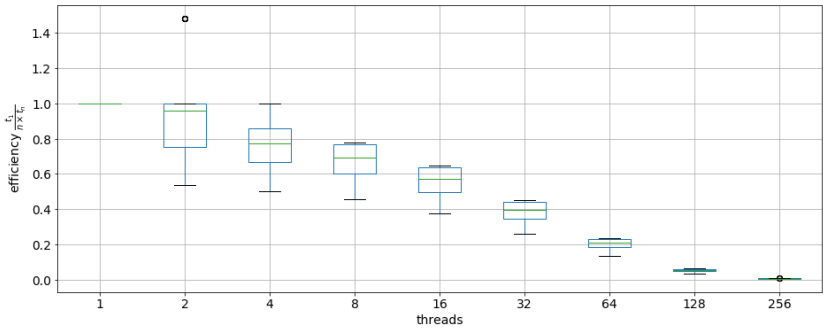

Figure 17: Average efficiency with varying number of threads on all studied instances.

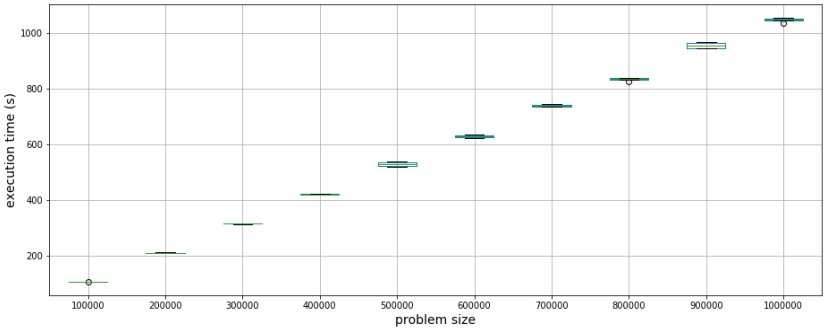

Figure 18: Average execution time with varying problem size on all studied instances using 64 cores.

to the specific classifier used for the prediction phase and operators used for the improvement phase. The main challenges in VS lie with effectively decomposing the problem during the training phase and selecting the problem features that allow achieving the best accuracy during predictions.

In this article, VS was applied to the basic independent Next Release Problem, which can be reformulated as a 0/1 Knapsack Problem (KP). A thorough study of the training and prediction phases of VS was presented, and five different variants for the improvement phase of VS were analyzed for the problem under consideration.

Experimental evaluation was performed using a publicly-available benchmark of problem instances of varying size and correlation. Correlation between the profit and the weight of the items is a measure of instance hardness for the Nemhauser-Ullmann algorithm, where higher correlations degrade the efficiency of the algorithm. Nemhauser-Ullmann algorithm is an exact method, and we adopt it in this work as the reference algorithm. Therefore, we are using VS to learn how to generate an optimal solution to the problem.

We first performed a study on the training set size required for accurate learning to solve the problem. We considered the use of 10%, 15%, 25%, 50%, and 100% observations in dataset-1 (out of the 50 datasets available). From the study, we conclude that a training set composed by 15% of all instances in the dataset was enough to produce close to optimal results. Beyond that percentage, only marginal improvements were observed. When considering only the performance 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 profit/weight correlation.

The improvement phase in VS helps to further refine the solutions generated in the prediction step. Experimental results show that VS is able to compute highly accurate solutions. Among the five versions studied in this work, the simplest variant (a greedy mechanism that first corrects the solution and then improves it, based on the profit/weight ratio of elements) is the one that achieved the best results. The solutions computed by the VS version implementing this greedy mechanism are closer than 1% from the optima in the worst case, according to fitness value. Furthermore, VS was able to generate the optimal solution in many cases, and the median of the reported fitness values is never farther than 0.2% from the optimum computed by the reference algorithm. It was also observed that, interestingly, difficult instances for the reference algorithm are not necessarily difficult to

solve by VS.

Finally, we evaluated the scalability of VS when solving very large problem instance on a Xeon Phi many-core computing infrastructure. Experimental results showed that VS was able to take advantage of the availability of multiple processing units to significantly reduce the execution times. A significant performance improvement is observed up to the point when the number of threads used exceeds the number of physical cores in the platform. At this point, the sharing of computational resources becomes detrimental to the performance of VS.

As future work, we consider that extending our study with the analysis of the performance of VS for the hardest known instances of the 0/1 KP will be very interesting, as well as tackling harder variants of the NRP with different constraints. Additionally, the adaptability of VS to different computing environments (e.g., many and multi-core architectures, cluster, grids) needs to be analyzed in the near future. Moreover, it is possible to enhance the improvement step of VS with specific heuristics making use of problem knowledge in order to design a highly competitive tool against the state of the art. Regarding the VS framework, we will evaluate its performance when using other machine learning classifiers for the prediction phase. Additionally, we plan on embedding attribute selection methods into the VS model, which might be especially relevant in problems with large number of features. Finally, we will work on the application of the VS paradigm to other different optimization problems. Particularly, we are interested in modelling optimization problems on graphs, involving dependencies among the different variables.

Acknowledgements

The work of R. Massobrio and S. Nesmachnow was partially funded by ANII and PEDECIBA, Uruguay. R. Massobrio thanks Fundación Carolina, Spain. The work of F. Palomo-Lozano was partially supported by the Spanish Ministry of Science, Innovation and Universities under projects RTI2018-093608-B-C33 (FAME) and RED2018-102472-T (SEBASENet 2.0). The work of B. Dorronsoro was supported by the Spanish Ministerio de Ciencia, Innovación y Universidades under project RED2018-102472-T (SEBASENet 2.0), the Spanish Ministerio de Ciencia, Innovación y Universidades and the ERDF under contract RTI2018-100754-B-I00 (iSUN project), and ERDF under project FEDER-UCA18-108393 (OPTIMALE).

References

[1] B. Nuseibeh, S. Easterbrook, Requirements engineering, in: Proceedings of the Conference on The Future of Software Engineering, ACM Press, 2000.

[2] A. Aurum, C. Wohlin, Engineering and Managing Software Requirements, Springer-Verlag, Berlin, Heidelberg, 2005.

[3] A. Bagnall, V. Rayward-Smith, I. Whittley, The next release problem, Information and Software Technology 43 (14) (2001) 883–890.

[4] S. Nesmachnow, An overview of metaheuristics: accurate and efficient methods for optimisation, International Journal of Metaheuristics 3 (4) (2014) 320–347.

[5] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, 2004. doi:10.1007/978-3-540-24777-7.

[6] F. Pinel, B. Dorronsoro, Savant: Automatic Generation of a Parallel Scheduling Heuristic for Map-Reduce, International Journal of Hybrid Intelligent Systems 11 (4) (2014) 287–302.

[7] F. Pinel, B. Dorronsoro, P. Bouvry, The virtual savant: Automatic generation of parallel solvers, Information Sciences 432 (2018) 411–430.

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

[9] D. Treffert, The savant syndrome: an extraordinary condition. a synopsis: past, present, future, Philosophical Transactions of the Royal Society B: Biological Sciences 364 (1522) (2009) 1351–1357.

[10] D. Treffert, Extraordinary People: Understanding Savant Syndrome, iUniverse, 2006.

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

[12] N. Veerapen, G. Ochoa, M. Harman, E. K. Burke, An integer linear programming approach to the single and bi-objective next release problem, Information and Software Technology 65 (2015) 1–13.

[13] G. L. Nemhauser, Z. Ullmann, Discrete dynamic programming and capital allocation, Management Science 15 (9) (1969) 494–505.

[14] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology 2 (2011) 27:1–27.

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

[16] G. B. Dantzig, Discrete variable extremum problems, Operations Research 5 (1957) 266–277.

[17] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Willey & Sons, 1990.

[18] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, in: C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems 28, Curran Associates, Inc., 2015, pp. 2692–2700.

[19] I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, arXiv preprint arXiv:1611.09940 (2016).

[20] H. Hu, X. Zhang, X. Yan, L. Wang, Y. Xu, Solving a new 3d bin packing problem with deep reinforcement learning method, CoRR abs/1708.05930 (2017). arXiv:1708.05930.

[21] K. Li, J. Malik, Learning to optimize, Tech. rep., CoRR, abs/1606.01885 (2016).

[22] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, L. Song, Learning Combinatorial Optimization Algorithms over Graphs, arXiv e-prints (2017).

[23] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, D. L. Dill, Learning a sat solver from single-bit supervision, in: International Conference on Learning Representations, 2019.

[24] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmller, T. Bauernhansl, A. Knapp, A. Kyek, Optimization of global production scheduling with deep reinforcement learning, Procedia CIRP 72 (2018) 1264 – 1269, 51st CIRP Conference on Manufacturing Systems.

[25] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, H. Xie, Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning, IEEE Access 7 (2019) 39974–39982.

[26] A. A. Araújo, M. Paixão, Machine learning for user modeling in an interactive genetic algorithm for the next release problem, in: C. Le Goues, S. Yoo (Eds.), Search-Based Software Engineering, Springer International Publishing, Cham, 2014, pp. 228–233.

[27] A. A. Araújo, M. Paixao, I. Yeltsin, A. Dantas, J. Souza, An architecture based on interactive optimization and machine learning applied to the next release problem, Automated Software Engineering 24 (3) (2016) 623–671.

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

[29] R. Massobrio, B. Dorronsoro, S. Nesmachnow, Virtual savant for the heterogeneous computing scheduling problem, in: International Conference on High Performance Computing & Simulation, 2018b, pp. 1–7.