Virtual Savant: learning for optimization

Anonymous Author(s)
Affiliation
Address
email

Abstract

This article describes Virtual Savant, a novel paradigm that applies machine learning to derive knowledge from previously-solved optimization problem instances in order to solve new ones in a massively-parallel fashion. Applications of Virtual Savant to two classic combinatorial optimization problems and to one real-world problem are presented and experimental results are discussed.

1 Introduction

Nowadays, there is an increasing need for intelligent algorithms that efficiently solve complex optimization problems arising in different areas. Furthermore, there is an increased interest in techniques that can automatically generate elastic programs that can fully exploit highly-parallel computing platforms and scale in the number of computing resources.

Virtual Savant (VS) is a novel paradigm that takes advantage of machine learning techniques and parallel computing to solve complex optimization problems. VS aims to find patterns in previously-solved problem instances which allow solving unseen instances in a massively-parallel fashion.

This article presents VS and its application to solve two standard combinatorial optimization problems: 0/1-Knapsack Problem (0/1-KP) and Heterogeneous Computing Scheduling Problem (HCSP), and the complex real-world Bus Frequencies Synchronization Problem (BFSP). Experimental results focus on the quality of the solutions computed by VS as well as its scalability when increasing the size of the problem instances.

2 Virtual Savant: learning for optimization

VS is inspired in the *Savant Syndrome*, a rare mental condition in which patients develop far above normal abilities [7]. Patients with savant syndrome (*savants*) excel at specific activities (e.g., related to memory, rapid calculation, or artistic abilities). The main theories suggest that savants learn through pattern recognition to solve problems without understanding their underlying principles.

VS proposes an analogy to the savant syndrome, using machine learning techniques to find patterns (i.e. *building blocks*) that allow solving optimization problems. This approach is also applied in other optimization techniques, such as evolutionary algorithms (EA) and estimation distribution algorithms (EDA), although these methods model building blocks implicitly in the search. In VS, patterns are learned from a set of previously-solved problem instances computed by reference algorithm(s) for the problem. VS does not require knowing 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 aggregating building blocks. Once the training phase is completed, VS can solve unseen—and even larger—problem instances without the need of any further retraining. Similarly to real-life savants, which are thought to have parallel cognitive capabilities [9], VS can also take advantage of multiple computing resources for learning and problem solving.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

One of the key design principles in VS is to learn relationships between the elements of the problem. The simplest case models variables (univariate analysis), but more complex models (bivariate, multivariate) can be considered too. The univariate case considers as many training examples as variables in the problem being solved. Thus, a problem decomposition approach is applied by defining subproblems to learn from. The main challenge during training, as in most machine learning problems, lies in selecting the instance features that maximize the accuracy of the trained model. This decision is highly problem-specific and involves a process of instantiating the general schema of VS to the problem at hand, e.g., by combining expert knowledge and empirical analysis.

In the prediction phase, the trained classifier is used to predict a solution to a new, unseen, problem instance. Each relationship in the problem instance can be predicted independently, thanks to the design followed in the training process of VS. Many machine learning libraries implement methods that allow estimating label probabilities for multi-class classification problems [8]. In this case, the output of the trained model is not a single label but instead a vector of size equal to the number of possible classes, indicating the probability of labeling the given input to each of the possible classes.

During the improvement phase, multiple candidate solutions are generated using the probability distributions of labels predicted in the prior phase, which are further refined using search procedures and heuristics to improve their quality in terms of the objective function of the optimization problem being solved. Additionally, predicted solutions can be given as input seeds to more complex optimization methods such as EAs and other metaheuristics. Additionally, corrective functions are included in the improvement operator to ensure that the returned solutions satisfy all problem constraints. In general, the spirit of VS is to use general optimization strategies during the improvement phase, without relying on problem-specific techniques. By doing so, VS can be applied to solve multiple problems from many different domains.

Thanks to its design VS can be run in a massively parallel fashion. The parallel model of VS applies the MapReduce approach [3]. In the prediction phase, predictions can be made in parallel by using multiple copies of the same trained classifier, since each relationship is learned independently. Similarly, the improvement phase is also subject to a high degree of parallelism. 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 1 outlines the complete workflow of VS. First, a set of solved instances is used to train the classifier, considering features associated with each element in the problem independently. Once training is completed, VS can solve new—unseen—problem instances. In the prediction phase, a copy of the trained classifier is spawn for each of the n elements in the new problem instance. Each classifier receives the vector of features corresponding to one of the problem elements and outputs a vector indicating the probability of predicting each of the possible m labels for that variable. The first synchronization barrier (shown in red) allows waiting for all classifiers to output their predictions. Once all classifiers have completed their task, candidate solutions are generated by drawing values for each variable according to the probabilities predicted by the classifiers. In the example, r candidate solutions are built, each of which is improved by applying the improvement operator in parallel. The second synchronization barrier is in charge of waiting for all the improvement operators to finish. Finally, all solutions are gathered and the best overall solution is returned.

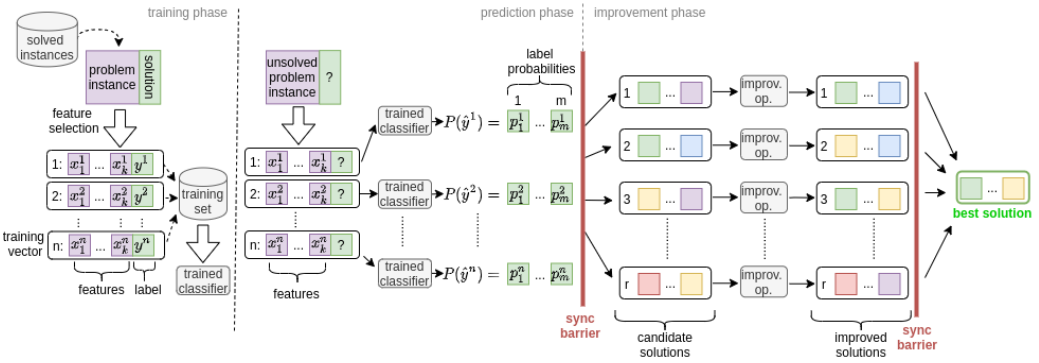

Figure 1: Parallel VS workflow

3 VS for classic combinatorial optimization problems

The application of VS for classic problems considers modeling the relevant features during training and using the trained classifiers to generate solutions which are further improved. Validation results confirm that VS is able to efficiently solve classic problems, as described next.

0/1-Knapsack Problem. 0/1-KP proposes finding a subset of items, each one with weight w_k and profit p_k, to maximize the total profit, without exceeding a weight capacity C: $\max\{\sum_{k=1}^{n} p_k x_k \mid \sum_{k=1}^{n} w_k x_k \leq C\}$; $x_k = 1$ if item k is included in the knapsack or 0 otherwise.

The VS implementation for 0/1-KP uses Support Vector Machines (SVMs) as supervised machine learning classifiers [1]. Optimal solutions computed by the Nemhauser-Ullmann algorithm [4] are used as reference. Training vectors have three features (item weight, item profit, and capacity) and the label is 0/1, according to the reference solution. The improvement phase applies a greedy algorithm.

Experimental evaluation was performed on benchmark instances of varying size and correlation between item profit and weight. Experimental results demonstrated that VS computes highly-accurate solutions, closer than 1% from the optima in the worst case. Interestingly, difficult instances for the reference algorithm are not necessarily difficult to solve for VS.

Heterogeneous Computing Scheduling Problem. The HCSP proposes finding an assignment of tasks T to heterogeneous computing resources M to optimize the *makespan* (i.e., the time between the start of the first task and the completion of the last task): $\min \max_{m_j \in M} \sum_{t_i \in T, f(t_i)=m_j} ET(t_i, m_j)$, where $ET(t_i, m_j)$ is the execution time of task t_i on machine m_j and f is the assignment function.

The VS implementation for the HCSP uses SVMs for the training phase and MinMin [5] as a reference algorithm [2]. Tasks are considered individually during training, allowing VS to scale to significantly larger instances. The feature vector includes the execution time of each task on each machine and the classification label is the machine assigned by MinMin. A local search (LS) including problem knowledge is applied in parallel to candidate solutions during the improvement phase.

In the experimental evaluation carried out over 180 problem instances VS outperformed MinMin by up to 15% in terms of makespan. Furthermore, VS showed excellent scalability when increasing both the computational resources and the problem dimensions, taking advantage of its massively-parallel design to reduce the execution times and the makespan values when using more resources.

4 VS for a complex, real-world optimization problem

When applied to complex problems, with hard and sparse search spaces, VS must properly model the underlying structure of the solution space and the objective function landscape.

The BFSP models a relevant problem in public transportation. Consider a set of bus lines that synchronize in a set of synchronization nodes $B = \{<i, j, wt_b^{ij}>\}$, where i, j are two lines and wt_b^{ij} is the time needed to walk between bus stops of lines i and j. TT_b^i is the time to reach node b for line i. P_b^{ij} is the number of passengers that transfer from line i to line j in node b. W_b^{ij} is the maximum time passengers are willing to wait for a synchronization of lines i and j in node b. The separation between consecutive trips of line i (headway) is within $[h_i, H_i]$, defined by transportation authorities.

The BFSP proposes finding the best combination of headways for each line to maximize synchronization for all lines in a planning period $[0, T]$, where the demand is split uniformly among the number of trips of line i (f_i). Synchronizations are bounded by the capacity C (Eq. 1).

$$\text{maximize} \quad \sum_{b \in B} \sum_{i \in I} \sum_{j \in J(i)} \sum_{r=1}^{f_i} \sum_{s=1}^{f_j} Z_{rsb}^{ij} \times \min\left(\frac{P_b^{ij}}{f_i}, C\right) \tag{1a}$$

$$\text{subject to} \quad X_r^i \in \{1, \ldots, T\}, Z_{rsb}^{ij} \in \{0, 1\} \tag{1b}$$

$$X_1^i \leq H^i;\ T - H^i \leq X_{f_i}^i \leq T \tag{1c}$$

$$h^i \leq X_{r+1}^i - X_r^i \leq H^i \tag{1d}$$

$$wt_b^{ij} < (X_s^j + TT_b^j) - (X_r^i + TT_b^i) \leq W_b + wt_b^{ij} \text{ if } Z_{rsb}^{ij} = 1 \tag{1e}$$

In constraints 1b–1e, X_r^i is the departure time of trip r of line i. Z_{rsb}^{ij} defines if trip r of line i and trip s of line j are synchronized in node b. Eq. 1e defines the condition for two trips to be synchronized: trip s of line j and trip r of line i must arrive to node b such that the waiting time is lower than W_b.

The VS design for the BFSP uses Random Forests as machine learning classifiers for the training phase, which uses solutions computed by an EA as reference [6]. Each synchronization point in a given instance in considered independently in the learning process of VS. Thus, each feature vector is comprised of synchronization node features (i.e., walking time between stops wt, transfer demand P, maximum waiting time allowed W), inbound line features (i.e., travel time to synchronization node TT_i, minimum allowed headway h_i, maximum allowed headway H_i), and outbound line features (analogous to inbound line). Two different classifiers are trained to predict the headway of each line.

A given bus line can appear on multiple synchronization nodes as the inbound or outbound line. Thus, multiple headways can be predicted for each line. If only one prediction is available for the line it is selected as the headway value; otherwise a random headway is selected (uniform distribution) among the multiple predictions. If no valid predictions are available for the line, a random headway is chosen (uniform distribution) within the valid range of headways for that line. Multiple candidate solutions are generated and improved using a LS that performs random modifications in the defined headways. The change is accepted if the solution improves and is discarded otherwise.

45 real-world problem instances with up to 110 synchronization nodes were built using data from the bus system in Montevideo, Uruguay (30 for training and 15 for validation). Table 1 reports the best results of VS before the improvement phase (*VS*), after 5000 iterations of the LS (*VS+LS*), and the following baseline solutions: the current real solution applied by the transportation administration (*real*), a randomly-generated solution (*random*), a solution where every bus line is set to its minimum allowed headway (h_i), and the best solution from the reference EA (*EA*). Results are normalized using *EA* as reference. Boxplots in Figure 2 summarize the results. In turn, Figure 3 presents the scalability of VS when solving larger instances with up to 450 synchronization nodes.

#I	*real*	*random*	h_i	*EA*	*VS*	*VS+LS*
1	0.9282	0.9168	0.9757	1.0	0.9952	0.9989
2	0.8816	0.8835	0.9689	1.0	0.9964	1.0002
3	0.9290	0.9093	0.9765	1.0	0.9965	0.9994
4	0.9161	0.9090	0.9769	1.0	0.9877	0.9994
5	0.9279	0.9059	0.9749	1.0	0.9984	1.0005
6	0.9170	0.9182	0.9740	1.0	0.9984	1.0008
7	0.8861	0.8639	0.9695	1.0	0.9895	1.0009
8	0.8973	0.8787	0.9670	1.0	0.9958	1.0004
9	0.8817	0.8842	0.9647	1.0	0.9951	1.0017
10	0.9165	0.9027	0.9773	1.0	0.9907	0.9996
11	0.9280	0.9127	0.9750	1.0	0.9985	1.0008
12	0.9170	0.9142	0.9740	1.0	0.9976	1.0003
13	0.9239	0.9268	0.9747	1.0	0.9912	1.0006
14	0.8883	0.8707	0.9683	1.0	0.9933	1.0010
15	0.9238	0.9074	0.9746	1.0	0.9954	1.0002

Table 1: VS and baseline results for BFSP.

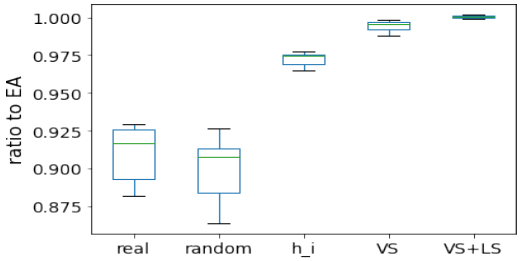

Figure 2: VS and baseline results for BFSP.

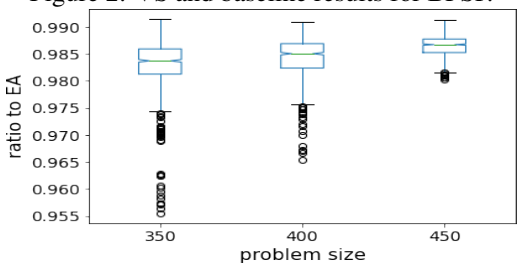

Figure 3: Scalability of VS.

VS is able to compute accurate results even when considering only the prediction phase, outperforming the real solution by up to 11.5%, the randomly-generated solution by up to 12.6%, and the minimum headway solution by up to 3.0%. On average, the prediction phase computes solutions that are 99.4% as good as the reference solution (98.7% in the worst case). These are highly competitive results, showing that only the prediction phase of VS is able to compute accurate solutions for the studied instances. Notwithstanding, including the LS operator allows further refining the computed solutions. When performing a 5000-step LS the ratio to the reference solution increases up to 1.56% in the best case, outperforming the reference EA in 11 out of 15 problem instances. The experimental evaluation also shows the good scalability and robustness of VS, for larger problem instances than those seen during training, comprised of 110, 350, 400, and 450 synchronization nodes.

Broader Impact

Virtual Savant, the paradigm presented in this article, proposes combining machine learning and parallel computing to efficiently solve large instances of complex combinatorial optimization problems. As outlined in the article, Virtual Savant is an efficient tool to solve both standard optimization problems and real-world applications. Therefore, both researchers and practitioners can take advantage of the good scalability offered by Virtual Savant when solving large problem instances of complex combinatorial optimization problems. We do not foresee any negative societal concerns related to this proposal.

References

[1] Omitted for peer-review.

[2] Omitted for peer-review.

[3] J. Dean and S. Ghemawat. Mapreduce: Simplied data processing on large clusters. In *Sixth Symposium on Operating System Design and Implementation*, pages 137–150, 2004.

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

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

[6] S. Nesmachnow, J. Muraña, G. Goñi, R. Massobrio, and A. Tchernykh. Evolutionary approach for bus synchronization. In J. L. Crespo-Mariño and E. Meneses-Rojas, editors, *High Performance Computing*, pages 320–336, Cham, 2020. Springer International Publishing.

[7] D. Treffert. *Extraordinary People: Understanding Savant Syndrome*. iUniverse, Bloomington, Indiana, USA, 2006.

[8] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.

[9] M. Yamaguchi. Savant syndrome and prime numbers. *Polish Psychological Bulletin*, 40(2):69–73, 2009.