Automatic program generation: Virtual Savant for the knapsack problem

Renzo Massobrio[1,2], Bernabé Dorronsoro[2], Sergio Nesmachnow[1], and Francisco Palomo-Lozano[2]

[1] Universidad de la República, Uruguay
renzom@fing.edu.uy sergion@fing.edu.uy
[2] Universidad de Cádiz
bernabe.dorronsoro@uca.es francisco.palomo@uca.es

1 Introduction

This work presents the application of Virtual Savant (VS) for the automatic generation of programs that solve the 0/1 knapsack problem. VS is a new method that uses machine learning techniques to learn how a reference algorithm solves a given problem [6]. VS receives as input a set of problem instances and the results computed by a reference algorithm, which is used to train a machine learning classifier. Once the training phase is completed, VS can run in parallel to solve unknown problem instances. The method showed promising results for a task scheduling problem [5].

This article presents the application of the VS paradigm to the 0/1 knapsack problem. It is a classic $\mathcal{NP}$-hard combinatorial optimization problem that models several important problems arising in real-world applications [3]. Given a set I of items, each with a profit p_i and a weight w_i, the knapsack problem consists of finding a subset of items that maximizes the total profit, without exceeding the weight capacity W. The problem formulation is $\arg\max \{\sum_{i=1}^{n} p_i x_i \mid \sum_{i=1}^{n} w_i x_i \leqslant W\}$, where $x_i \in \{0, 1\}$ indicates whether item i is included or not in the knapsack.

The main contribution of this work is a detailed study of the behavior of VS on the 0/1 knapsack problem. The study is needed to show the usefulness of VS as a combinatorial optimization method since it was only evaluated in one other problem before. The accuracy of VS on a large set of problem instances with different difficulty degrees is analyzed and results are evaluated.

2 Virtual Savant for the knapsack problem

VS is comprised of two phases: *classification*, where results for unknown problem instances are predicted, and *improvement*, where predicted results are further improved using specific search procedures. Support Vector Machines (SVMs) were applied for the classification phase, trained with the Nemhauser-Ullmann algorithm [2] using LIBSVM with a Radial Basis Function kernel [1]. The classifier aims to predict whether or not to include a given item in the knapsack, considering the weight and profit of each item individually. The output of the classification phase is a vector that holds the probability of including each item in the knapsack. The improvement phase takes the resulting vector of the previous phase, generates a random feasible solution considering each item probability, and performs a simple local search heuristic based on random modifications to decide if the items are included in the knapsack or not. Additionally, several correction schemes are included in order to ensure that the returned solution satisfies the knapsack capacity restriction [4].

3 Experimental analysis

The experimental evaluation was performed over a benchmark of problem instances with different size and correlation between weight and profit of items. The correlation evaluates the difficulty to solve an instance [2]. The benchmark includes 50 datasets, with instances of size 100 to 1500 items (stepsize: 100). For each problem size, correlation varies from 0.0 to 1.0 (stepsize: 0.05).

The first dataset is used for training the SVM. Initial experiments evaluated the precision of the SVM when increasing the number of samples used for training. Results show that good precision results (in terms of accuracy of the predicted values) are achieved when using just 15% of the samples of *dataset 1* during training. Kernel parameters were calibrated using cross-validation and problem instances were scaled prior to training.

While the prediction phase of VS is deterministic, the improvement phase—consisting of a local search—is not. Therefore, 30 independent executions with 1000 iterations of the local search heuristic were performed for each problem instance. The obtained results were compared with the known optima for the studied instances, to evaluate the efficacy of VS. Table 1 reports the average ratio to the optimum, grouping instances by the correlation between item weight and profit.Table 2 reports the average ratio to the optimum with problem instances grouped by size.

correlation	0.0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
ratio	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.98	0.99	1.00	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1.00

Table 1. Average ratio to the optimum with varying difficulty.

size	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500
ratio	0.96	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97

Table 2. Average ratio to the optimum with varying size.

Results show that VS allows computing accurate results for all problem instances studied. Considering instances grouped by size, VS is able to compute results that differ 3–4% from the known optima. There are no differences in the computed results between small and larger instances, suggesting good scalability of the VS paradigm. Considering instances grouped by correlation, VS is always able to find the optima for instances with correlation 0.5 and 0.95. The quality of the solutions, on average for all correlations, differs from the optima in only 2.75%. In the worst case, when there is no correlation between items weight and profit, the results computed with VS differ in only 9% from the optimum.

4 Conclusions and future work

Experimental results show that VS allows computing competitive results for 0/1 knapsack problem instances of varying size and difficulty. Results are very encouraging, since the improvement phase of VS consists of a simple local search heuristic and straightforward corrections schemes. The main lines of future work include applying different machine learning classifiers, using other heuristics and metaheuristics for the improvement phase, and evaluating over larger problem instances. Additionally, we intend to apply VS to related optimization problems with harder constraints.

Acknowledgments

This work was partly supported by Fundación Carolina (Spain), MINECO-FEDER (TIN2014-60844-R and RYC-2013-13355, Spain), PEDECIBA and ANII (Uruguay).

References

1. C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.
2. 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.
3. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 2004.
4. R. Massobrio, B. Dorronsoro, F. Palomo-Lozano, S. Nesmachnow, and F. Pinel. Generación automática de programas: Savant Virtual para el problema de la mochila. In *XI Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, 2016.
5. F. Pinel and B. Dorronsoro. Savant: Automatic Generation of a Parallel Scheduling Heuristic for Map-Reduce. *International Journal of Hybrid Intelligent Systems*, 11(4):287–302, 2014.
6. F. Pinel, B. Dorronsoro, P. Bouvry, and S. Khan. Savant: Automatic parallelization of a scheduling heuristic with machine learning. In *World Congress on Nature and Biologically Inspired Computing*, pages 52–57. IEEE, 2013.