Learning sensitivity of schedules by analyzing the search process [⋆]

Marc-André Ménard[1], Claude-Guy Quimper[1], and Jonathan Gaudreault[1]

Université Laval, Québec QC, CA
marc-andre.menard.2@ulaval.ca
Claude-Guy.Quimper@ift.ulaval.ca
jonathan.gaudreault@ift.ulaval.ca

Abstract. Solving the problem is an important part of optimization. An equally important part is the analysis of the solution. Several questions can arise by analyzing the solution. In a scheduling problem, is it possible to obtain a better solution by increasing the capacity of a resource? What happens to the objective value if we change a coefficient? It is important to answer these questions not only to have a better solution, but also for the acceptability of the solution. A lot of research has been done on sensitivity analysis, but few articles can be applied to constraint programming. We present a new method for sensitivity analysis that can be applied to constraint programming. This method collects information during the search for a solution by the solver, and more precisely about the propagation of the CUMULATIVE constraint. It also collects information about the solution returned by the solver. We predict if increasing the capacity of the CUMULATIVE allows obtaining a better solution. We experiment this approach on a scheduling problem but this method can be used on other problems. The results obtained validate the presented method.

Keywords: Sensitivity analysis · Learning · Scheduling.

1 Introduction

Scheduling problems are important for companies to execute their tasks on time and maximize their productivity. Scheduling also helps companies to make better decisions.

Once the scheduling is done, it is interesting for a company to answer questions and to test different scenarios. For example, how much time a company would save by increasing the capacity of a resource? Is it possible to complete a new order in time? What is the company's bottleneck? That is, which resource causes a slowdown in the production because there is not enough of this resource in inventory? Answering these questions is important to improve the

[⋆] Supported by APN inc, CRIQ, and the Natural Sciences and Engineering Research Council of Canada (NSERC).

solution. This corresponds to the sensitivity analysis of the solution. The sensitivity analysis is the study of the impact on the solution of changing the value of a parameter or several parameters.

The sensitivity analysis can be done by altering the model to encode a "what-if" scenario and solving the new problem. This process can take a lot of time if the company has hundreds of resources or tasks. The solver may take several minutes or hours to return a single solution. If the problem is a linear program, it is possible to use the solution of the dual to find a range of changes for the coefficients of the objective function or the values of the constants to the right of the inequalities while keeping the optimality of the solution. In constraint programming, the dual solution is not as well defined as for linear programs and not trivial to compute. It is therefore impossible to use the same methods as for the linear programming to make the sensitivity analysis.

This article presents a method for predicting whether increasing the capacity of a resource would improve the solution to a scheduling problem. The method uses the information collected on the constraint Cumulative [1] when solving the problem and information about the resource in the solution found. It also solves variations of the instances where the resource capacities are modified. Once a machine learning classifier is trained, it can quickly predict, for new instances of the problem, whether it is possible to obtain a better solution by increasing the capacity of the resource. We take the scheduling problem, but it is possible to apply this method to any other problem that has a Cumulative constraint.

The rest of the article is presented as follows. First, we present the background of this research. Second, we explain the methodology. Third, we present preliminary results of the method on a scheduling benchmark. Fourth, we present some ideas for future work.

2 Background

2.1 Sentivity analysis

The sensitivity analysis is the study of the impact on the solution of changing the value of a parameter. This problem is widely studied in the literature [5][6].

Hall et al. [8] identify 4 questions that sensitivity analysis attempts to answer. The first question is to know the range of possible changes for each parameter to keep the optimal solution. This corresponds to the shadow price of the resource. The second question is to know the new objective value for a specific change of a parameter. The third question is to know the new optimal solution following a change to a parameter. Finally, the fourth question is to know the differences for the answers to questions 1 to 3 if we apply several changes to the parameter values simultaneously. This article focuses on the second question. Hall et al. [8] present algorithms for the sensitivity analysis for some type of scheduling problem. They point out that it is more difficult to answer questions when the problem is NP-hard.

Hooker [9] proposes an approach to do sensitivity analysis on linear or discrete problems. This method involves using the *dual inference* to obtain a proof of the optimality of the solution. Any optimization problem has a dual inference. Solving the inference dual of a problem consists in inferring the best possible bound on the optimal objective value from the constraints. With the proof found with the inference dual, it is possible to change the data of the problem as long as the proof remains valid. Hooker [9] shows an example on a 0-1 linear programming problem. Dawande et al. [4] extend this idea to Mixed Integer linear programming problem.

Hadzic et al. [7] use binary decision diagrams (BDD) to enumerate possible solutions to the problem. This can take a lot of time for a big problem, but allows afterward to answer questions very quickly. It allows the user to answer questions about the right-hand side of an inequation but also about the variables domain. For example, what is the best solution to the problem if a variable is set to a value?

2.2 Lazy clause generation

The solver we use for our experiments is Chuffed [3]. It uses the lazy clause generation to take advantage of the high level modeling and understanding of the structure of the problem of constraints programming and the inference graph of SAT solvers [11]. This approach makes it possible to model the problem using global constraints that make stronger connections between variables and to create no-goods that avoid the search to explore an already visited subtree for which no solution exists. Our method uses no-goods as a feature. We modify the chuffed solver to collect data on the no-goods.

In a solver using no-goods, propagators must find an explanation for their filtering. This explanation can be represented as a clause which is a disjunction of literals. A literal is a Boolean variable which can be represented in this form $[\![x \leq v]\!]$ where x is a variable and v is a value. This litteral indicates that x is smaller than or equal to v. A literal can also take the following form $[\![x = v]\!]$ for the equality between a variable and a value. All the clauses found by the propagators form an implication graph to explain the domain changes of the variables. When a constraint is unsatisfiable, it is possible, using the implication graph, to extract a reason why the constraint is unsatisfied. This creates a no-good which is itself a clause i.e. a disjunction of literals that can be used for the rest of the search anywhere in the search tree.

3 Methodology

This section shows our method used to predict whether increasing the capacity of a resource provides a better solution.

We use a machine learning classifier to predict whether increasing the capacity of a resource improves the objective value of a scheduling problem. This is a classification problem with 2 classes. Class 0 represents that there is no change to the objective value and class 1 represents an improvement in the objective value.

We use two types of information as features. First, there is the information collected during the solver's search for a solution. Then there is the problem-specific information called *knowledge specific.*

For the constraint CUMULATIVE, we use the time-tabling [2] and time-tabling-edge-finder [12] filtering rules. During the search for a solution by the solver, it is possible to collect several information about the CUMULATIVE constraints. The information we collect is the number of inconsistencies of the time-tabling, the number of times that the time-tabling filtered the domain of a variable, and the amount of the filtering of the time-tabling. From this information, it is possible to have the average filtering amount when the constraint filters. It is possible to obtain the same information for the time-tabling-edge-finder.

We use the constraints programming solver Chuffed. Since we use a solver generating no-goods, it is important to link a no-good to the constraints used to generate it. Without this link, a lot of information about the constraints would be lost. No-goods can perform the majority of filtering for some part of the problem. As a no-good is generated from explanations of several constraints, it is not possible to link a no-good to a single constraint. To link the constraints to the no-good we first create a map between the explanations and the constraints that generated this explanation. Then, we link the no-good to the constraints by checking the explanations used to generate the no-good and the constraints that generated these explanations. Also, if a no-good b is generated by an explanation of another no-good a, we bind the new no-good b to the constraints that generated the no-good a. In the end, if a no-good induces a filtering or a backtrack, it will be possible to give credits to the constraints that inferred that no-good.

We use three knowledge specific information for the prediction. First, there is the utilization rate of the resource. The utilization rate U_r^{rate} of a resource r is calculated by the equation (1) where p_i is the processing time of task $i \in I$, h_{ri} is the amount of the resource r used to execute task i and M is the makespan of the solution. It is the amount of energy $(p_i \cdot h_{ri})$ consumed by the tasks over the availability of the resource $(c_r \cdot M)$.

$$U_r^{rate} = \frac{\sum_{i \in I} p_i \cdot h_{ri}}{c_r \cdot M} \tag{1}$$

Secondly, we computed, as a feature, the duration U_r^{max} for which the resource r is used at full capacity. Let $U_{r,t}$ be the usage of the resource r at time t. U_r^{max} is the number of time points at which the resource is fully used.

$$U_{r,t} = \sum_{i \in I : s_i \leq t < s_i + p_i} h_{ri} \tag{2}$$

$$U_r^{max} = |\{t \mid U_{r,t} = c_r\}| \tag{3}$$

Third, there is the number of times W_r a task waits after a resource to start. In other words, the number of times a task could have started if it did not need the resource. To get this information, we must make sure that the task does not start because of a resource and not because of a precedence with another task. This number is calculated with equation (4) where $pred_i$ is the set of predecessors of the task i.

$$W_r = |\{i \in I \mid s_i \neq \max_{j \in pred_i}(s_j + p_j) \land U_{r,s_i-1} + h_{ri} > c_r\}| \tag{4}$$

To train our classifier, we need to create a training dataset. First, we solve the original problem by collecting the constraint information when solving the problem and the knowledge specific resource information. Subsequently, for each resource, the problem is solved again by increasing the capacity of the resource by a certain percentage. We use the new objective value to know whether or not increasing the capacity allows to have a better solution to the scheduling problem. Creating the training dataset takes a lot of time as it requires to solve multiple instances, but once the classifier is trained, it can be used to predict whether increasing the capacity of the resource leads to a better objective function.

While training, it is possible that the solver does not find the optimal solution to the problem. We still keep the instance for training. However, if the solver finds a solution that is worse than the original solution when increasing the capacity of a resource, we remove these observations from the dataset as it is not possible to obtain a worse solution by increasing the capacity of the resource. With a higher capacity for a resource and a sufficient time for the solver, the solver would find a solution with an objective value equal to or better than the original objective value.

4 Results

For our experiments, we use the PSPLib dataset [10]. We experimented on four benchmarks of the job-shop scheduling problem: j30, j60, j90 and j120. The benchmark j30, j60 and j90 contain 480 instance and the benchmark j120 contains 600 instance. Each instance of the problem has precedence between tasks and four Cumulative constraints. The objective function is to minimize the makespan. We used the model provided by Minizinc[1]. For the constraint Cumulative, we use the time tabling and time-tabling-edge-finder as filtering rules.

For each resource, we experimented with increasing the capacity of the resource by different percentages. The higher the percentage, the higher the objective value may improve.

The dataset is unbalanced. There are often more resources for which increasing capacity does not change the objective value (class 0). The most unbalanced instance is for the j60 benchmark with 10% change in resource capacity. For this instance, there is 76% of the resources in class 0. Even with 100% increase in capacity, the dataset remains unbalanced in favor of class 0. There is only for

[1] https://github.com/MiniZinc/minizinc-benchmarks/tree/master/rcpsp

the benchmark j120 that there is more class 1 than class 0 with 62% of class 1 with a change of capacity of 100%.

For our experiments, we separate our dataset into training datasets and test datasets. In the dataset, we do not take into consideration which instance the resource belongs. For example, for experiments on the benchmark j30 instances of PSPLib, we mix all resources without considering which instance they belong to. We take 80% of the resources for the training set and the remaining 20% for the test dataset.

We apply a min-max normalization on the data to scale them between 0 and 1. The equation (5) shows the transformation applied to each values of the features where x is the value of a feature and $\min(x)$ and $\max(x)$ returns the minimum and maximum values of the feature in the dataset.

$$x_i^{new} = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{5}$$

To perform the prediction, we tested three different classifiers in scikit-learn[2] with different parameters using a grid search: logistic regression, SVM and random forest.

For the logistic regression classifier, we tested different values for the C parameter that controls the trade-off between the simplicity of the model and degree of learning. The values tested are: 0.001, 0.01, 0.1, 1, 10, 100, and 1000. The best value of the parameter changes according to the data set and the percentage of change.

For support vectors classifier (SVM), we use a radial core basis function (rbf) for the kernel. We tested the following values for the parameter γ: 0.01, 0.1, 1, 10, and 100. The higher the value of γ, the more the model tries to fit the training dataset. The other parameter tested is the C parameter which controls the trade-off between the simplicity of the model and the degree of learning. The values tested are: 0.001, 0.01, 0.1, 1, 10, 100, 1000. Like the logistic regression classifier, the best values of the parameters change according to the dataset and the percentage of change.

For the random forest classifier, we took 100 estimators and tested different values for the max depth parameter. This parameter indicates the maximum depth of the tree. The values tested are all numbers between 1 and 32.

Table 1 shows the accuracy obtained by each classifier on the different datasets and the percentage of change in the capacity of the resources. The random forest classifier usually gets better accuracy than the other classifiers tested. We can also notice that we get a worse precision for the problems j30. This may be due to the fact that these problems are much easier to solve than others and little information is collected on the CUMULATIVE constraints during the resolution.

Table 2 shows the precision and the recall for the random forest classifier on the different datasets and different percent of change of capacity. Precision indicates how many times the classifier made a good prediction when it predicted the class. The recall indicates the number of times that the classifier has

[2] https://scikit-learn.org/stable/

Table 1. Accuracy (%) of classifiers on different datasets.

		Classifiers accuracy (%)		
Instances	Capacity change (%)	Logistic regression	SVM	Random forest
j30	10	78.24	**78.70**	75.93
	20	75.46	**77.31**	75.46
	30	80.09	80.09	**81.94**
	50	75	75.46	**79.63**
	100	75	75.93	**80.09**
j60	10	84.64	85.39	**88.01**
	20	89.22	89.59	**91.45**
	30	86.62	**89.22**	88.1
	50	88.1	**89.22**	88.85
	100	87.36	89.59	**90.33**
j90	10	89.49	90.22	**91.67**
	20	86.07	88.93	**89.29**
	30	88.26	86.83	**89.68**
	50	89.75	89.75	**90.11**
	100	87.94	88.3	**90.43**
j120	10	80.7	81.8	**86.62**
	20	85.16	86.24	**88.39**
	30	86.41	87.26	**89.17**
	50	85.17	86.44	**87.92**
	100	83.9	86.65	**88.35**

predicted the class on the number of times the class is present in the dataset. For example, if the classifier only predicts class 0, it would have a recall of 1, but a low precision. In the results, precision and recall are always close to accuracy. The classifier does not favor one class more than another during training.

5 Future work

The results obtained are preliminary results. Several other classifiers remain to be tested such as neural networks. We also want to experiment with different types of normalization to improve the prediction accuracy.

It would be possible to add other knowledge specific information to improve the prediction. For example, rather than counting the number of times the task waits after a resource, we could add the length of the wait as a feature.

Also, sensitivity analysis usually involves finding the range of change of a parameter. It would be possible to test whether to decrease the capacity of the resource allows to keep the same objective value.

We also want to test doing a regression instead of a classification to predict how much we are improving the objective value by increasing the capacity of a resource. It would be more interesting for a company to predict which resource to buy and how much the objective value changes with these new resources. So, the company could compare the gain of buying a resource with its cost.

Table 2. Precision and recall of random forest classifier on different datasets.

			Precision (%)		Recall (%)	
Instances	Capacity change (%)	Accuracy (%)	0	1	0	1
J30	10	75.93	78	72	83	66
	20	75.46	77	73	79	71
	30	81.94	82	81	84	79
	50	79.63	80	79	81	78
	100	80.09	81	80	81	79
J60	10	88.01	91	83	89	87
	20	91.45	90	94	95	87
	30	88.1	88	88	91	85
	50	88.85	88	90	92	85
	100	90.33	88	94	95	85
J90	10	91.67	94	86	94	87
	20	89.29	93	82	91	86
	30	89.68	94	82	91	87
	50	90.11	94	83	91	88
	100	90.43	94	84	92	88
J120	10	86.62	86	87	78	92
	20	88.39	88	89	80	93
	30	89.17	90	89	82	94
	50	87.92	87	88	80	93
	100	88.35	89	88	82	93

Furthermore, it would also be interesting to predict for an instance of the problem which resource is the bottleneck. This will avoid having to run the classifier on all resources and instead give a suggestion on the important resources to buy. We could also incorporate the detection of a combination of resources that creates the bottleneck of the problem. In other words, increasing the capacity of one of the two resources does not improve the objective value, but increasing the capacity of both resources improves the objective value.

We want to experiment this method on real data from the manufacturing company APN inc. APN is a high-precision manufacturing company working in the aeronautical and military field. Their scheduling problem involves more than one hundred resources. It is therefore interesting for APN to detect which resource would allow them to increase their production by purchasing these resources. Especially if these resources have a low cost.

6 Conclusion

This paper presents a method to predict whether it is advantageous to have more capacity of a resource for constraint programming problems. The combination of learning models and the learning about the structure of the problem through constraint programming makes it possible to answer interesting questions for companies. Although there is still a lot of experimentation to be done, the results obtained are very promising for the rest of the project.

References

1. Aggoun, A., Beldiceanu, N.: Extending chip in order to solve complex scheduling and placement problems. Mathematical and Computer Modelling **17**(7), 57 – 73 (1993)
2. Beldiceanu, N., Carlsson, M.: A new multi-resource cumulatives constraint with negative heights. In: Van Hentenryck, P. (ed.) Principles and Practice of Constraint Programming - CP 2002. pp. 63–79. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
3. Chu, G.: Improving combinatorial optimization. PhD thesis, Department of Computing and Information Systems, University of Melbourne (2011)
4. Dawande, M.W., Hooker, J.N.: Inference-based sensitivity analysis for mixed integer/linear programming. Operations Research **48**(4), 623–634 (2000)
5. Geoffrion, A.M., Nauss, R.: Exceptional paperparametric and postoptimality analysis in integer linear programming. Management Science **23**(5), 453–466 (1977)
6. Greenberg, H.J.: An Annotated Bibliography for Post-Solution Analysis in Mixed Integer Programming and Combinatorial Optimization, pp. 97–147. Springer US, Boston, MA (1998)
7. Hadzic, T., Hooker, J.: Postoptimality analysis for integer programming using binary decision diagrams. In: GICOLAG Workshop (Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geometry), Vienna. Technical report, Carnegie Mellon University (2006)
8. Hall, N.G., Posner, M.E.: Sensitivity analysis for scheduling problems. Journal of Scheduling **7**(1), 49–83 (Jan 2004)
9. Hooker, J.N.: Inference duality as a basis for sensitivity analysis. Constraints **4**(2), 101–112 (May 1999)
10. Kolisch, R., Sprecher, A.: Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program. European Journal of Operational Research **96**(1), 205 – 216 (1997)
11. Stuckey, P.J.: Lazy clause generation: Combining the power of sat and cp (and mip?) solving. In: Lodi, A., Milano, M., Toth, P. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 5–9. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
12. Vilím, P.: Timetable edge finding filtering algorithm for discrete cumulative resources. In: Achterberg, T., Beck, J.C. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 230–245. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)