Supervised Learning to Control Energetic Reasoning : Feasibility Study

Sascha Van Cauwelaert[1] *, Michele Lombardi[2], and Pierre Schaus[1] **

Université Catholique de Louvain,
Università di Bologna

Abstract. Propagation is a double-edged sword, with more pruning power coming at the price of larger computation time. For each problem constraint, the best propagator depends on the specific instance and may change at search time. We propose to use an oracle function, obtained via Machine Learning, to decide whether to run complex propagators for a target constraint. In this paper, we focus on investigating the feasibility of building an oracle for the Energetic Reasoning propagator used in scheduling. Our experiments show that high prediction accuracy can be obtained, provide suggestions for classification features, and highlight important issues to address when building such an oracle.

Propagation is a double-edged sword: more powerful filtering algorithms provide an increased chance to prune values, but they also have larger computation time, that must be paid regardless of whether additional propagation is actually achieved.

The `cumulative` constraint is widely employed to model resource restrictions for scheduling problems with non-preemptive activities. The constraint counts a wide range of propagators. Timetable propagation (TT) is known to be dominated by Energetic Reasoning (ER, see [4]), which is however seldom used in practice because the algorithm has higher time complexity ($O(n^3)$ against $O(n^2)$). However, there exist Energetic Reasoning overload checkers (i.e., detecting infeasibility) [4, 8] that run in $O(n^2)$.

The best propagator depends on specificities of the target problem and instance, and it is far from trivial to select. The choice is typically done by the model designer based on personal experience, intuition, and pilot tests, with mixed outcomes (we refer to [17] for interested reader).

The work presented in this paper is strictly related to Algorithm Selection, meaning the activity of deciding the best algorithm for tackling a given problem, first formalized in [16]. For an excellent overview on Algorithm Selection in the context of Combinatorial Optimization (covering Hydra – and derivatives –, SATzilla, ParamILS and ISAC), the reader is referred to [13]. Despite the extensiveness of the literature about Algorithm Selection, only a few works so

* PhD. S.
** Adv.

far have addressed the problem of choosing propagators (or adjusting the consistency levels) in Constraint Programming(CP) automatically. According to our knowledge, the related works in this domain are [9, 6, 18, 2, 5, 14, 10].

We propose to use an oracle, obtained via Machine Learning(ML), to predict at run time if running a specific propagator for a constraint will be beneficial. The decision will be based on the current domain of the variables in the constraint scope, i.e., on the input of the propagator itself.

Deploying effectively such an approach is an ambitious endeavour. As a start, in this paper we investigate the feasibility of building an oracle for the ER propagator. In particular, we focus on the problem of detecting whether running ER will narrow the domains after TT and ER checker has reached a fix-point. In this study, we show that high prediction rates can indeed be obtained, we suggest effective features to be used as input for the classifier, and we highlight critical issues to be addressed in the design of the oracle.

1 Oracle to Predict Propagation

Given a target constraint c and the current domains of the variables in its scope $S(c)$, we consider the problem of predicting whether a propagator π_c will cause some pruning or not. Formally, we are interested in designing an oracle[1] function O_{π_c} such that:

$$O_{\pi_c}(D_i|x_i \in S(c)) = \begin{cases} \textit{true} \text{ if some value is pruned} \\ \textit{false} \text{ otherwise} \end{cases}$$

where D_i is the domain of the variable x_i and $S(c)$ is the scope of the constraint c. The O_{π_c} function is meant to be used as a guard condition for the execution of the propagator.

This problem formulation has a number of advantages: first, the oracle is *guaranteed to have enough information to make a correct guess*[2]. The challenge is therefore to devise an O_{π_c} function with lower complexity than the propagator itself. Second, if a new propagator for the constraint is introduced, a new oracle must be trained, but *the existing ones require no modification at all*. Third, the oracle *can be checked at any point during search*, making the designer completely free about how to combine propagators (as long as the fallibility of the oracle is taken into account). More importantly, this also makes the approach well suited for use in complex search strategies and Large Neighborhood Search.

Although π could be applied at any time during the search process, when π subsumes another lighter propagator already used during the fix point, it makes sense to use it only to perform *additional deductions*. In this case, it is clear that the decision should be made after the fix point is reached. Once reached, a post fix point procedure would be applied in order to make as many additional

[1] Strictly speaking, an *estimator* of an oracle.
[2] Provided the estimator is perfect.

deductions as possible. In this procedure, for each constraint c, we check if O_{π_c} is true and if so, we apply π_c. After each call to a given π_c, if some pruning has occured, we call the fix point algorithm for which *only* the *other* propagators are involved. This mechanism is applied as long as some pruning is performed. If an inconsistency is derived, backtracking occurs. Informally, this procedure is a fix point for which domain reductions are at most as strong as the ones obtained with a fix point in which the π_c are always applied.

2 Case study : Energetic Reasoning

Resource Constrained Project Scheduling Problems (RCPSP) consist in finding a start time for a set A of activities. Each activity a_i has a fixed duration d_i and requires an amount r_{ik} of each resource r_k from a set R. Each resource has limited capacity cap_k. The activities may be connected by precedence constraints. The goal is to minimize the worst case completion time (makespan). An RCPSP is modeled in CP by introducing a start variable s_i for each activity and by modeling the resource restrictions via `cumulative` constraints [1, 3] for each resource r_k : $\forall t = 0..eoh \sum_{s_i \leq t < s_i + d_i} r_{ik} \leq cap_k$, i.e., no resource overusage can occur. The term eoh refers to the maximum possible end time, where each end time e_i corresponds to $s_i + d_i$. The bounds for s_i and e_i have conventional names: est_i and lst_i are the earliest and latest start times, respectively, while ect_i and lct_i are the earliest and latest completion (end) times.

ER is a propagator for the cumulative constraint that is rarely used due to its cubic complexity. This work focus on the following problem : for a given instance of the *RCPSP*, build an oracle O_{ER_c} for each cumulative c. Particularly, the oracle decides for a cumulative constraint whether or not to run ER after a TT propagator and the ER checker have reached the fix point (as well as the other constraints of the problem). Moreover, in order to quantify the gain of using oracles in terms of number of nodes reduction and number of backtracks reduction, we use a *static* search strategy. Variable branching order is fixed and value branching is binary : left branch assign the variable to its minimum, right branch removes the minimum from the domain.

First of all, let us assume that for each cumulative c we have access to a perfect oracle $O^{perfect}_{ER_c}$ that always predicts correctly. Clearly, if using $O^{perfect}_{ER_c}$ in order to solve problems provides no time gain (i.e., the time difference compared with the minimum time required when O_{ER_c} always returns true or always returns false), the overall approach is useless. For instance, in the case of the BL instances [3] with 20 activities, the maximum[3] time gain on average is 27.3%.

Input Features

In this part, we try to provide guidelines for picking meaningful features by describing the ones we used for the oracle functions in the case of ER. In particular,

[3] Maximum from the point of view of our approach, i.e., when the local prediction is always correct. It is possible that making wrong predictions actually implies time gain due to the trade-off between search and propagation.

we obtain our features by first extracting intermediate characterizations for the cumulative constraint (*cumulative characterizations*), and then by computing aggregated statistics. The cumulative characterizations are numbers obtained from static information about the cumulative constraint and from the domains. For their description it is useful to introduce some definitions (most are introduced in [4] and [5]) :

- Point of interests considered by ER:
  $O_1 = \{est_i\} \cup \{ect_i\} \cup \{lst_i\} \quad O_2 = \{lct_i\} \cup \{lst_i\} \cup \{ect_i\}$
- Time Intervals considered by ER (here $O(t) = \{est_i + lct_i - t\}$):

$$\{[t,t')\}\,\forall t \in O_1, \forall t' \in O_2, t' \geq t \cup \{[t,t')\}\,\forall t \in O_1, \forall t' \in O(t), t' \geq t \\ \cup \{[t,t')\}\,\forall t' \in O_2, \forall t \in O(t'), t' \geq t$$

- Relative Energy: $\tilde{e}_i = \frac{d_i \cdot r_{ik}}{lct_i - est_i}$
- Relative Energy Histogram: $\widetilde{E}(t) = \sum_{a_i \in A : est_i \leq t < lct_i} \frac{d_i \cdot r_{ik}}{lct_i - est_i}$
- Relative Requirement Histogram: $\widetilde{R}(t) = \sum_{a_i \in A : est_i \leq t < lct_i} r_{ik}$

Generic cumulative characterizations: Table 1 presents our generic cumulative characterizations. They are computed for each activity a_i and have $O(1)$ complexity, so that for a given cumulative constraint we have a vector of values for each characterization type. Obtaining each vector has complexity $O(n)$, where n is the number of activities. H stands for the value of the current horizon.

Characterization Name	**Value**
compulsoryPart	$\frac{max(0, est_i + dur_i - lst_i)}{(lst_i + dur_i - est_i)}$
fixedActivity	$\lfloor est_i / lst_i \rfloor$
domainTightness	$(lst_i - est_i)/H$
estLstRatio	est_i / lst_i
relativeEnergy	$\tilde{e}_i / C$
est	est_i / H
lct	lct_i / H
durationHorizon	d_i / H
durationDomain	$d_i / (lst_i + dur_i - est_i)$
requirement	r_{ik} / cap_k

Table 1: Generic cumulative characterizations.

ER specific characterizations: The characterizations in Table 2 are specific to ER. Both *timePoints* (i.e., points of interests considered by ER) and *intervalSize* are computed for every Time Interval considered in ER. The notation *lub* stands for the *least upper bound* and is used instead of *max* as the intervals are right-open. The characterizations *relativeEnergyTimePoints* and *relativeRequirementTimePoints* are computed for every time point in the *timePoints* characterization. Moreover, we also have characterizations based on *relativeEnergyTimePoints* and *relativeEnergyTimePoints* that consider only time points crossing at least one activity i with a non-fixed s_i. Similarly, we also generate vectors for which an activity is only considered if its compulsory part[4] (i.e., $[lst_i, ect_i)$, if

[4] Information used by TT.

$lst_i < ect_i$) crosses the considered time point. The size of every characterization is $O(n)$ except for *intervalSize* which is $O(n^2)$. For each characterization, the time complexity for computing all the values is $O(n^2)$.

Characterization Name	**Parameter**	**Value**
timePoints	Interval $\mathcal{I}$	$\{min(\mathcal{I})/H, lub(\mathcal{I})/H\}$
intervalSize	Interval $\mathcal{I}$	$(lub(\mathcal{I}) - min(\mathcal{I}))/H$
relativeEnergyTimePoints	Time point t	$\bar{E}(t)/C$
relativeRequirementTimePoints	Time point t	$\bar{R}(t)/C$

Table 2: ER specific characterizations.

Final features: The features used as input for the classification algorithm are aggregation statistics computed for each characterization type. In particular, for each vector of values we consider the *minimum*, *maximum*, *arithmetic mean*, *geometric mean*, *median*, *first quartile*, *third quartile*, *population variance*, *sample variance*, *kurtosis*, *skewness*, *length*, *cardinality* (i.e., the number of distinct elements).

Complexity: The time complexity to compute a cumulative characterization is at most $O(n^2)$, thus lower than the one of ER. Most of the statistics that we employ as features have complexity $O(m)$, where m is the size of the vector from which the feature is extracted. Some statistic operations involve an ordering step and have therefore a complexity of $O(m \log(m))$. Hence, the worst time complexity for computing some features is $O(n^2 log(n^2))$[5], but most of them are actually obtained in $O(n)$, $O(n \log(n))$, or $O(n^2)$.

3 Experiments

We considered the BL instances with 20 activities to test our approach. The learning method we used is Random Forests [7] with default parameters from the Weka [12] API. We used the CP solver from OscaR [15]. Difficult problems directly arise :

- how to build a representative training set, i.e., representative of the search nodes that will be met during search.
- how to select the features ; there is a trade-off between computation time of the features and prediction performances.

We do not have clear answers to those problems. For now, let us assume we have access to the complete search tree for which each cumulative c, O_{ER_c} returns true with a probability p, in order to simulate the use of O_{ER_c} in a given search. We fix $p = 0.5$, as we do not have knowledge about the a priori probability that ER will prune or not at a given node. With this knowledge, if during an actual search, we are able to make good prediction results with a small sample of the search nodes, we make a (mandatory) first step towards our goal. We experimented

[5] Clearly, as ER is in $O(n^3)$, those features will not be used once the oracles will be considered in a search process.

this approach on several BL instances and present preliminary results in Fig. 1. Notice we only used the features with time complexity of $O(n.log(n))$ or lower[6].

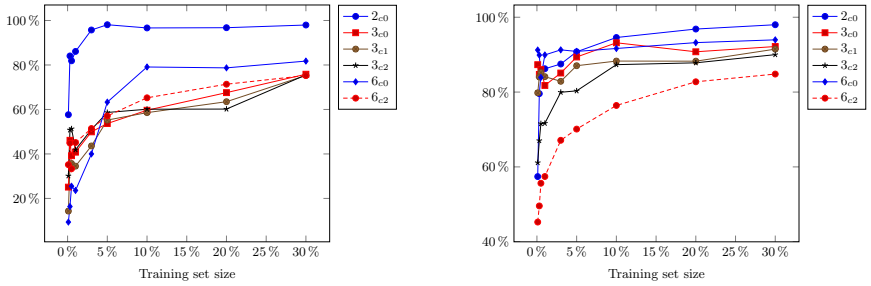

Fig. 1: True Positive (left) and True Negative (right) Rates of oracles used during search. Legends provide the instance number and the cumulative number.

It is worth to mention that those results depend on the actual search tree. A decision or the other at a given search node can have important impacts on the corresponding subtree, and therefore, on the set of elements to be classified.

4 Conclusion and Future Work

This paper proposes an approach to take better advantage of complex propagators, by running them only when they provide an actual benefit in terms of pruning. We propose to achieve this goal by relying on an oracle function, obtained via ML techniques. In this work we focus on investigating the feasibility of an accurate oracle for ER in the context of the `cumulative` constraint. We show that a high prediction accuracy can be obtained with a reasonably low time complexity.

A classifier should always be imperfect in the general case. Therefore, one of the future direction is to study the effect of prediction mistakes, in order to know what prediction performances our oracle must have to allow time gain. Average correct prediction results may not be sufficient ; prediction errors of certain type can have drastic effects on the overall search process, e.g., missing pruning in the upper part of the search tree. We also need to make the approach faster, by relying on fewer or cheaper-to-compute features, by using incremental computation, or by simplifying the classifiers (e.g., fewer trees in a Random Forest). Moreover, we are interested in using adaptable classifiers, in order to incorporate the approach in a Randomized Large Neighborhood Search for Cumulative Scheduling [11]. The knowledge of the oracles would then grow more and more with the restarts. We also plan to use the approach for other constraints (such as Bin-Packing).

[6] Although the others allow better prediction performances.

References

1. Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.
2. Amine Balafrej, Christian Bessiere, Remi Coletta, and El Houssine Bouyakhf. Adaptive parameterized consistency. In *Principles and Practice of Constraint Programming*, pages 143–158. Springer, 2013.
3. Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1-2):119–139, 2000.
4. Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer, 2001.
5. Timo Berthold, Stefan Heinz, and Jens Schulz. An approximative criterion for the potential of energetic reasoning. In *Theory and Practice of Algorithms in (Computer) systems*, pages 229–239. Springer, 2011.
6. James E Borrett, Edward PK Tsang, Natasha R Walsh, and Colchester Co Sq. Adaptive constraint satisfaction: The quickest first principle. In *European Conference on Artificial Intelligence*, pages 160–164. Citeseer, 1996.
7. Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
8. Alban Derrien and Thierry Petit. The energetic reasoning checker revisited. *Principles and Practice of Constraint Programming (Doctoral Program)*, 2013. arXiv preprint arXiv:1310.5564.
9. Hani El Sakkout, Mark G Wallace, and E Barry Richards. An instance of adaptive constraint propagation. In *Principles and Practice of Constraint Programming*, pages 164–178. Springer, 1996.
10. Ian Gent, Lars Kotthoff, Ian Miguel, and Peter Nightingale. Machine learning for constraint solver design–a case study for the alldifferent constraint. *3rd workshop on Techniques foR Implementing Constraint programming Systems*, 2010. arXiv preprint arXiv:1008.4326.
11. Daniel Godard, Philippe Laborie, and Wim Nuijten. Randomized large neighborhood search for cumulative scheduling. In *Fifteenth International Conference on Automated Planning and Scheduling*, pages 81–89, 2005.
12. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
13. Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 2014. arXiv preprint arXiv:1210.7959.
14. Olivier Lhomme. Backtracking adaptatif. *7ièmes Journées Francophones de Programmation par Contraintes (JFPC'11)*, pages 173–182, 2011.
15. OscaR Team. OscaR: Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.
16. John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.
17. Barbara M Smith. Modelling. In Francesca Rossi, Peter Van Beek, and Toby Walsh, editors, *Handbook of constraint programming*, pages 377–406. Elsevier, Oxford, 2006.
18. Kostas Stergiou. Heuristics for dynamically adapting propagation in constraint satisfaction problems. *Ai Communications*, 22(3):125–141, 2009.