Hybridizing Constraint Programming and Monte-Carlo Tree Search: Application to the Job Shop problem

Manuel Loth, Michèle Sebag, Youssef Hamadi, Marc Schoenauer, Christian Schulte

Hybridizing Constraint Programming and Monte-Carlo Tree Search: Application to the Job Shop problem

Manuel Loth[1,2], Michèle Sebag[2], Youssef Hamadi[1], Marc Schoenauer[2], and Christian Schulte[3]

[1] Microsoft Research, Cambridge, UK
[2] TAO, CNRS−INRIA−LRI, Université Paris-Sud, France
[3] School of ICT, KTH Royal Institute of Technology, Sweden

Abstract. Constraint Programming (CP) solvers classically explore the solution space using tree search-based heuristics. Monte-Carlo Tree-Search (MCTS), a tree-search based method aimed at sequential decision making under uncertainty, simultaneously estimates the reward associated to the sub-trees, and gradually biases the exploration toward the most promising regions. This paper examines the tight combination of MCTS and CP on the job shop problem (JSP). The contribution is twofold. Firstly, a reward function compliant with the CP setting is proposed. Secondly, a biased MCTS node-selection rule based on this reward is proposed, that is suitable in a multiple-restarts context. Its integration within the Gecode constraint solver is shown to compete with JSP-specific CP approaches on difficult JSP instances.

1 Introduction

This paper focuses on hybridizing Constraint Programming (CP) and Monte-Carlo Tree Search (MCTS) methods. The proof of concept of the approach is given on the job-shop problem (JSP), where JSPs are modelled as CP problem instances, and MCTS is hybridized with the Gecode constraint solver environment [3]. This paper first briefly presents the JSP modeling in constraint programming and the MCTS framework, referring the reader to respectively [2] and [5,4] for a comprehensive presentation. The proposed hybrid approach, referred to as *Bandit-Search for Constraint-Programming* (*BaSCoP*), is thereafter described. The first experimental results on the difficult *Taillard 11-20* 20×15 problem instances are presented in section 5. The paper concludes with a discussion w.r.t related work [10], and some perspectives for further research.

2 CP-based resolution of JSP

The job-shop problem, one of the classical scheduling problems, is concerned with allocating jobs to machines while minimizing the overall makespan. The CP modelling of JSP proceeds by considering a sequence of problems: after a

given solution with makespan m has been found, the problem is modified by adding the constraint $makespan < m$.

While specific JSP-driven heuristics have been used in CP approaches, e.g. [11], our goal in this paper is to investigate the coupling of a generic CP framework, specifically Gecode [3], with MCTS. The JSP modeling in Gecode, inspired from the reportedly best CP approach to JSP [2], is as follows:

- the restart policy follows a Luby sequence [6]: the search is restarted after the specified number of failures is reached or when a new solution is found;
- the variable ordering is based on the *weighted-degree* heuristics − found as Max Accumulated Failure Count (AfcMax) in Gecode [3];
- the search heuristics is a Depth-First-Search, starting from the last found solution, i.e. the left value associated to a variable is the value assigned to this variable in the previous best solution.

3 Monte-Carlo Tree Search

MCTS [5] is concerned with optimal sequential decision under uncertainty, and is known in particular for its breakthrough in the game of Go [4]. MCTS proceeds by gradually and asymmetrically growing a search tree, carefully balancing the exploitation of the most promising sub-trees and the exploration of the rest of the tree. MCTS iterates N tree-walks, a.k.a simulations, where each tree-walk involves four phases:

- In the so-called bandit phase, the *selection rule* of MCTS selects the child node of the current node −starting from the root− depending on the empirical reward associated to each child node, and the number of times it has been visited. Denoting respectively $\hat{\mu}_i$ and n_i the empirical reward and the number of visits of the i-th child node, the most usual selection rule, inspired from the Multi-Armed Bandit setting [1], is:

$$\text{Select } i^* = \arg\max \left\{ \hat{\mu}_i + C\sqrt{\frac{\log \sum_i n_i}{n_i}} \right\} \tag{1}$$

- When MCTS reaches a leaf node, this node is attached a child node −the tree thus involves N nodes after N tree-walks− and MCTS enters the roll-out phase. This expansion may also occur only every k tree-walk, where k can be referred to as an *expand rate*.
- In the roll-out phase, nodes (a.k.a. actions) are selected using a default (usually randomized) policy, until arriving at a terminal state.
- In this terminal state, the overall reward associated to this tree-walk is computed and used to update the reward $\hat{\mu}_i$ of every node in the tree-walk.

MCTS is frequently combined with the so-called RAVE (Rapid Action Value Estimate) heuristic [9], which stores the average reward associated to each action (averaging all rewards received along tree-walks involving this action). In particular, the RAVE information is used to select the new nodes added to the tree.

4 *BaSCoP*

The considered CP setting significantly differs from the MCTS one. Basically, CP relies on the multiple restart strategy, which implies that it deals with many, mostly narrow, trees. In contrast, MCTS proceeds by searching in a single, gradually growing and eventually very large tree. The CP and MCTS approaches were thus hybridized by attaching average rewards (section 4.1) to each value of a variable (section 4.2). Secondly, *BaSCoP* relies on redefining the selection rule in the bandit- and in the roll-out phases (sections 4.3,4.4).

4.1 Reward

Although an optimization problem is addressed, the value to be optimized − the makespan − is of no direct use in the definition of the reward associated to each tree-walk. Indeed, all but a few of these tree-walks are terminated by a failure, that is an early detection of the infeasibility of an improvement over the last-found schedule. No significant information seems to be usable from the makespan's domain at a failure point. Hence, the depth of a failure is used as a base for the reward, as an indication of its closeness to success, with the following argument: the more variables are assigned the correct value, the deeper the failure.

Since the assignments of a given variable can occur at different depths within a tree and through the successive trees, it seemed reasonable, and was empirically validated, to consider the *relative failure depth* rather than the absolute one: after a tree-walk failing at depth d_f, for each variable v that was assigned, letting d_v be its assignment depth and x the assigned value, a reward $(d_f - d_v)$ is added to the statistics of (v, x).

4.2 RAVE

In the line of [4], the most straightforward option would have been to associate to each node in the tree (that is, a (variable, value) assignment conditioned by the former assignment nodes) the average objective associated to this partial assignment. This option is however irrelevant in the considered setting: the multiple restarts make it ineffective to associate an average reward to an assignment *conditioned by other variable assignments*, since there are not enough tree-walks to compute reliable statistics before they are discarded by the change of context (the new tree). Hence, a radical form of RAVE was used, where statistics are computed for each (variable,value) assignment, independently of the context (previous variables assignments).

4.3 Depth-First-Search Roll-Out

The roll-out phase also presents a key difference with the usual MCTS setting. The roll-out policy launched after reaching a leaf of the MCTS tree usually

implements a stochastic procedure, e.g. Monte-Carlo sampling (possibly guided using domain knowledge [4]); the roll-out part of the tree-walk is discarded (not stored in memory) after the reward has been computed.

In the considered CP setting, it is however desirable to make the search complete, i.e. exhaustive given enough time. For this reason, the roll-out policy is set to a depth-first search. As mentioned, in each node the left branch corresponds to setting the variable to its value in the last solution. DFS thus implements a search in the neighborhood of this last solution.

Contrary to random roll-outs, DFS requires node storage; yet only the last path of one DFS need be stored. DFS thus provides a simple and light-storage roll-out policy from which to gather reward statistics. As desired, the use of DFS as roll-out policy within MCTS enforces a complete search, provided that the restart sequence includes a "sufficiently long" epoch.

Overall, the coupling of MCTS with DFS in *BaSCoP* is similar in spirit to the *Interleaved Depth-First Search* [8]; the difference is that *BaSCoP* adaptively explores different regions of the search tree (within a restart).

4.4 Selection rules

As the left branch associated to each variable corresponds to the value assigned to this variable in the previous best solution, the selection rules determine the neighborhood of the previous solution which is explored in the current tree.

Several rules have been considered:

- **Balanced**: selects alternatively the left and the right node;
- **ϵ-left**: selects the left node with probability $1 - \epsilon$, and can be seen as a stochastic emulation of Limited Discrepancy Search;
- **UCB**: selects a node according to eq. (1) with no bias towards the left branch;
- **UCB-Left**: same as UCB, where different constants C_{right} and C_{left} are used to enforce the bias toward the left branch.

Figure 1 illustrates the domains and shapes designed by the selection rule and roll-out policies, by an example using a *Balanced* selection rule and DFS roll-outs.

5 Experimental results

Figure 2 depicts the overall results in terms of mean relative error w.r.t. the best (non CP-based) solution found in the literature, on the *Taillard 11-20* problem suite (20×15), averaged on 11 independent runs, versus the number of tree-walks. The computational cost is ca. 30mn on a PC with Intel dual-core CPU 2.66GHz. Compared to DFS, a simple diversification improves only on the early stages, while a left-biased one yields a significant improvement, of the same order as a failure-depth one, and improvements seem to add up when combining both biases.

Overall, *BaSCoP* is shown to match the CP-based state of the art [2]: the use of MCTS was found to compensate for the lack of JSP-specific variable ordering.

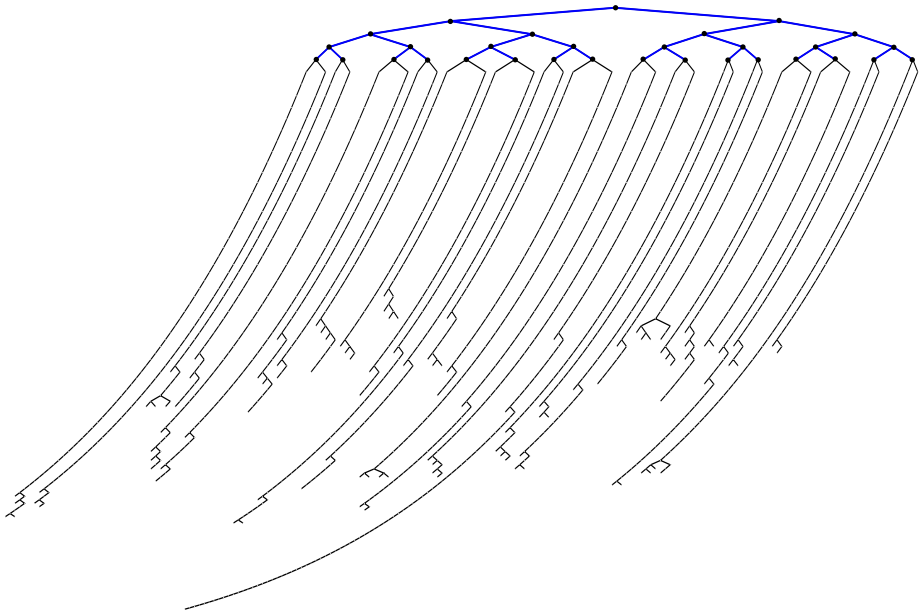

Fig. 1. Balanced + DFS search tree. Concurrent DFS are run under each leaf of the −growing− MCTS tree (dotted nodes).

6 Discussion and perspectives

The work most related to *BaSCoP* is [10], who compared an ϵ-greedy variant of MCTS to the so-called Pilot method on JSP problems. The Pilot method iteratively optimizes the option selected at each choice point, while sticking to the default heuristics for other choice points: it can thus be viewed as a particular and simplified case of MCTS. Interestingly, [10] concluded that MCTS was more effective than Pilot methods for small problem sizes; but Pilot methods were shown to catch up on large-sized problems, which was blamed on the inefficient random roll-out policies within MCTS.
Basically, *BaSCoP* most differs from [10] as it tightly integrates MCTS within a multiple-restart CP scheme, where the objective function (the makespan) cannot be used as reward.

Further work is concerned with investigating new variable-ordering heuristics, exploiting the RAVE information and combining per-node and per-variable statistics. Another perspective is to assess the generality and limitations of *BaSCoP* on other CP problems, such as BIBD [7] and car sequencing.

References

1. Auer P., Cesa-Bianchi N., Fischer P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47(2-3), 235–256 (2002)

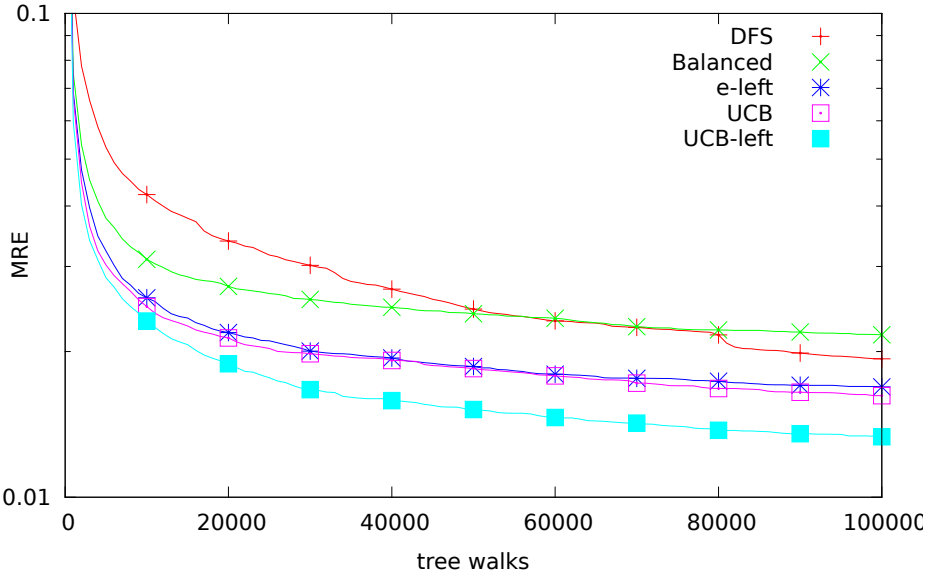

Fig. 2. Mean relative error, for 11 runs on the 10 20x20 Taillard instances

2. Beck J.C.: Solution-Guided Multi-Point Constructive Search for Job Shop Scheduling. Journal of Artificial Intelligence Research 29, 49–77 (2007)
3. Gecode Team: Gecode: Generic constraint development environment, available from `www.gecode.org`.
4. Gelly S. et al: The grand challenge of computer Go: Monte Carlo tree search and extensions. Communications of ACM 55(3), 106–113 (2012)
5. Kocsis L., Szepesvári C.: Bandit based Monte-Carlo planning. In: ECML 2006, pp 282–293 (2006).
6. Luby M., Sinclair A., Zuckerman D.: Optimal speedup of las vegas algorithms. Information Processing Letters 47, 173–180 (1993)
7. Mathon R., Rosa A.: Tables of parameters for BIBD's with $r \leq 41$ including existence, enumeration, and resolvability results. Annals of Discrete Mathematics 26, 275–308, (1985)
8. Meseguer P.: Interleaved depth-first search. In: IJCAI 1997, vol. 2, pp 1382–1387 (1997)
9. Rimmel A., Teytaud F., Teytaud O.: Biasing Monte-Carlo Simulations through RAVE Values. In: ICCG 2010, pp 59–68 (2010).
10. Runarsson T.P., Schoenauer M., Sebag M.: Pilot, Rollout and Monte Carlo Tree Search Methods for Job Shop Scheduling. In: LION 2012, pp 160–174 (2012)
11. Watson J.P., Beck J.C.: A Hybrid Constraint Programming / Local Search Approach to the Job-Shop Scheduling Problem. In: CPAIOR 2008, pp 263–277 (2008)