

Dynamic Cooperative Search for Constraint Satisfaction and Combinatorial Optimization: Application to a Rostering Problem

Boris Bontoux, Dominique Feillet

Laboratoire d'Informatique d'Avignon, Université d'Avignon et des Pays de Vaucluse, 339 chemin des Meinajaries, Agroparc B.P. 1228, F-84911 Avignon Cedex 9, France, {boris.bontoux, dominique.feillet}@univ-avignon.fr

Christian Artigues

LAAS CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 4, France, artigues@laas.fr

Eric Bourreau

LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France, eric.bourreau@lirmm.fr

In this paper, we are interested in enumerative resolution methods for combinatorial optimization (COP) and constraint satisfaction problems (CSP). We introduce a new approach for the management of branching, called Dynamic Cooperative Search (DCS) inspired from the impact-based search method proposed in [7] for CSPs. This method defines in a dynamic way priority rules for variable and value selection in the branching scheme. These rules are meant to be independent of the considered problem. As in [7], the principle is to take into account, through learning methods, of the impact of branching decisions in already explored subparts of the search tree. We show the interest of DCS on a real-life rostering problem.

1 Dynamic Cooperative Search: principle and first results

Most of the exact resolution methods for combinatorial optimization (COP) and constraint satisfaction problems (CSP), such as tree search, branch and bound and dynamic programming, rely on an intelligent enumeration of the solutions. In tree search methods, used for the resolution of both CSPs and COPs, child generation (or branching) consists in selecting a variable and its tentative value. In many cases, the branching policy, i.e., the strategy for the variable and value selections, have a huge influence on the efficiency of the method. In the case where the strategy allows one to obtain quickly good solutions, the enumerative method can be used within a limited amount of computing time, which is often imposed in practical applications such as the industrial rostering problem considered in this paper.

The Dynamic Cooperative Search (DCS) method is designed for leading the search towards the subparts of the solution space which contain the (best) solutions. The underlying ideas for speeding up the search tree exploration are not recent. Several techniques have been proposed in this way, most of them relying on a exploration of the search tree limited by a heuristic mechanism, such as Beam Search [6], Limited Discrepancy Search [2], Minimal Discrepancy Search [4], Branch-and-Greed [8]. Some of the efficient implementations of these methods have the common characteristic that they base the heuristic criteria on the structure of the studied problem. The principle of the DCS method is, on the opposite, relatively independent of the studied problem.

Refalo has proposed in [7] a general purpose search strategy for constraint programming based on *impacts*. He defines the impact of an assignment of a variable x_i to a value a as $I(x_i = a) = 1 - P_{after}/P_{before}$ where P is an estimation of the size of the search tree. To reduce the search tree, a variable with the greatest impact should be chosen first while a value with the smallest impact should be used first. In [5], an extension of this method to weighted CSPs is proposed.

The DCS method proposes a generalization of this method for both CSPs and COPs. It is used in a depth first search scheme and defines for every node priority rules for variable and value ordering. These orders are deduced from the characteristics of subtrees obtained from the previous

branching on the considered variables. Each variable is associated with a dynamic weight and the dynamic ordering of its values (represented through value weights). During the exploration, at each sub-tree closure (when the branch from the last value of the variable has been explored), the variable weight and the ordering of its values are updated. The way the variable and value weights are updated is a parameter of our method and depends on whether we are solving a CSP or a COP. In what follows we give, for each case, a possible implementation.

For COPs, we can define the weight of both variables and values as the cost of the best solution found in the considered subtree. Other possibilities is to consider the number of node prunings (for a Branch and Bound), the mean value of the solutions. . . Following this principle, we first apply DCS to the well-known Traveling Salesman Problem for which a huge number of solutions can be found quickly. We use a naive branching scheme where each node is associated with a city i and each child node corresponds to the city j visited immediately after i . Here we associate a weight w_{ij} to each arc (i, j) as the value of the best solution found in all its explored subtrees. The branching scheme consists of selecting the candidate arcs in the decreasing weight order. Under the above-described scheme, this correspond to a fixed variable ordering (each node of level k corresponds to fixing a value to variable x_k stating which arc is taken in position k) while values are explored through the DCS principle. Arc weights are initialized through a preprocessing technique described below. Preliminary results show a significant reduction of the size of the search tree and of the consumed CPU time against a simple Depth First Search method.

The Table 1 shows our results on a benchmark of 60 TSP instances with a number of cities varying from 10 to 22. The results presented are the mean CPU time (in seconds) and number of nodes, over all instances.

	Depth First Search	DCS	DCS with random solutions
Number of nodes	6,089,428	4,790,007	3,258,450
CPU time (in sec)	57	45	30

Table 1: Computational results for several methods applied to the TSP

For CSPs, the method is based on the following rules. When a variable is set to a value, we remember as the weight of this value, the maximum depth of the obtained subtree (after applying constraint propagation). At the branching step, once a variable is selected, its values are considered in the decreasing weight order aiming at first exploring the most promising regions. The weight of a variable is equal to the mean weight of its values. At the branching step, the variable are considered in nondecreasing order of their weights, aiming at reducing the size of the sub-tree, as in the impact-based search proposed in [7].

For both COP and CSP, a preprocessing phase is applied, during which severals random solutions are built to initialize the weights. For CSP, a tree search is applied with a random selection of variables and values. Weights are then computed with the method described above. For COP, a large number of solutions is randomly generated in a tree search and the variable and value weights are set with the obtained solution costs. The goal of a preprocessing phase is to lead the search towards the sub parts of the solution space which may contains a (good) solution, as a basic learning process. The time allowed for the preprocessing phase is limited to 10% of the computing time.

2 Application to an industrial rostering problem

The considered industrial problem is a personnel scheduling problem with constraints, similar to the *nurse rostering problem* [1]. Employees may have several skills. The problem is a constraint satisfaction problem aiming at satisfying skill constraints while ensuring that employees who have several skills are employed on each skill in a balanced way. This problem is a real case, proposed in the context of a partnership with the Daumas Autheman et Associés IT company (Aix-En-Provence, France) and concerns employee rostering in a ferry unloading company.

Let n be the number of employees. X_{ijk} denote the activity of employee i , on day j and week k , with $i = 1, \dots, n$, $j = 1, \dots, 7$, $k = 1, \dots, l$. X_{ijk} is equal to 0 when the employee is off duty. $S = 1, \dots, t$ denotes the set of considered activities. Let $C_i \subseteq S$ for $i = 1, \dots, n$ be the set of activities employee i is able to perform. E_{mjk} gives the number of required employees for activity m on day j and week k . The constraints are the following :

$$|\{X_{ijk} | X_{ijk} = m\}| = E_{mjk} \quad j = 1, \dots, 7 \quad k = 1, \dots, l \quad m = 1, \dots, t \quad (1)$$

$$|\{X_{ijk} | X_{ijk} = 0\}| \geq 2 \quad i = 1, \dots, n \quad k = 1, \dots, l \quad (2)$$

$$|\{X_{i6k} | X_{i6k} = 0\}| \geq r \quad i = 1, \dots, n \quad k = 1, \dots, l \quad (3)$$

$$|\{X_{i7k} | X_{i7k} = 0\}| \geq r \quad i = 1, \dots, n \quad k = 1, \dots, l \quad (4)$$

$$\lfloor \frac{5}{|C_i|} \rfloor \leq |\{X_{ijk} | X_{ijk} = m\}| \leq \lceil \frac{5}{|C_i|} \rceil \quad m \in C_i \quad i = 1, \dots, n \quad k = 1, \dots, l \quad (5)$$

$$X_{ijk} \in C_i \quad i = 1, \dots, n \quad j = 1, \dots, 7 \quad k = 1, \dots, l \quad (6)$$

Constraints (1) state that, on each time period, the number of employees who are assigned to activity m is equal to the required number of employees. Constraints (2) ensure that each employee is off duty at least two days a week. Constraints (3) and (4) ensure that the number of vacant Saturdays and Sundays is greater than r (where r is a data). Constraint (5) state that if an employee has several skills, he must be employed for each activity in a balanced way, 5 being the average working days per week.

We have implemented the variant of DCS as stated in Section 1 for CSPs and compared it with standard methods (goals) of the ILOG Solver constraint programming package.

The Table 2 shows our results on a benchmark of 25 randomly generated instances with a number of weeks varying from 4 to 75. The results presented are the mean CPU time (in seconds) to obtain a solution. The time limit is fixed to 600 seconds. Our method has been compared with two default goals in Ilog Solver (First Unbound, which chooses the first unbound variable first and Min Domain, which choose the unbound variable with the smallest domain first). The “Fisrst Unbound” and “Min Domain” rows display the results of these strategies with the smallest value selection rule (column ILOG Solver) and the DCS value selection rule (column “DCS”). The “Minimum average Weight” row displays the result for the full DCS variable and value strategies.

	Ilog Solver	DCS
First Unbound	94 s	117 s
Min Domain	52 s	24,34 s
Minimum Average Weight		0,75 s

Table 2: Computational results for several methods applied to the Rostering Problem

These results show that our method permits to solve much more instances than the default goal in Ilog Solver and it is up to 50 times faster for instances solved by both methods.

References

- [1] B. Cheang, H. Li, A. Lim, and B. Rodrigues (2003), Nurse rostering problems – A bibliographic survey, *European Journal of Operational Research* **151**, 447 – 460.
- [2] W.D. Harvey and M.L. Ginsberg (1995), Limited discrepancy search *In Proceedings of IJCAI95*, 607 – 613.
- [3] J. Hooker (2000), Logic-Based Methods for Optimization - Combining Optimization and Constraint Satisfaction. Wiley-Interscience series in discrete mathematics and optimization. John Wiley and Sons.
- [4] W. Karoui , M.J. Huguet , P. Lopez , W.Naanaa (2006), MDS: a learning method based on discrepancies and propagations, *Rapport LAAS N06093*.
- [5] N. Levasseur, P. Boizumault, S. Loudni (2007), Une heuristique de sélection de valeur dirigée par l'impact pour les WCSP, *Journée 19/01/2007 GT Contraintes et RO*.
- [6] P. S. Ow, T. E. Morton (1988), Filtered beam search in scheduling, *International Journal of Production Research* **26**, 35 – 62.
- [7] P. Refalo (2004), Impact-Based Search Strategies for Constraint Programming, *CP 2004*, 557 – 571.
- [8] F. Sourd, P. Chretienne (1999), Fiber-to-Object Assignment Heuristics, *European Journal of Operations Research* **117**.