Recovering Indirect Solution Densities for Counting-Based Branching Heuristics

Gilles Pesant[1] and Alessandro Zanarini[2]

[1] École Polytechnique de Montréal, Canada
gilles.pesant@polymtl.ca
[2] Dynadec Europe, Belgium
alessandro.zanarini@dynadec.com

Abstract. Counting-based branching heuristics in CP have been very successful on a number of problems. Among the few remaining hurdles limiting their general applicability are the integration of counting information from auxiliary variables and the ability to handle combinatorial optimization problems. This paper proposes to answer these challenges by generalizing existing solution counting algorithms for constraints and by relaying counting information to the main branching variables through augmented element constraints. It offers more easily comparable solution counting information on variables and stronger back-propagation from the objective function in optimization problems. We provide supporting experimental results for the Capacitated Facility Location Problem.

1 Introduction

The recent development of generic branching heuristics based on counting the number of solutions of individual constraints has led to state-of-the-art results on several benchmark problems [5]. They are indeed generic in principle and even adapt to a problem since they are derived from the constraints present in the model. However in practice they are restricted to problems modeled using constraints for which solution counting algorithms have been designed, a limitation that is being pushed back as new algorithms are introduced. They are also oriented toward solving feasibility problems and not so much optimization problems. Finally a technical incovenience is that some constraints are not expressed on the main decision variables but instead on auxiliary variables dependent on the former: these different sets of variables will each have their own solution counting information, which will be difficult to mix or compare.

This paper proposes to answer the latter two challenges by generalizing existing solution counting algorithms for constraints and by relaying counting information to the main decision variables through augmented element constraints. Section 2 presents the technical contribution. Section 3 discusses its use for combinatorial optimization problems. Section 4 gives a first experimental evaluation of the ideas through the Capacitated Facility Location Problem.

2 Framework

The most interesting indicator arising from solution counting information has been the *solution density* of a variable-value assignment, denoted $\sigma(X, a, \gamma)$, that is the proportion of solutions to a given constraint γ that assign value a to variable X. One of the simplest and most effective branching heuristics built from this is maxSD, which branches on the assignment with the overall highest solution density (among all constraints) [5]. This section describes the adaptations to the existing counting framework in order to recover indirect solution densities from constraints on auxiliary variables.

2.1 Channelling solution densities through `element` constraints

An $\texttt{element}(X, f, Y)$ constraint, with f a fixed array of integers, states the functional relationship $Y = f(X)$, which maps every value in the domain of finite-domain variable X to a value in the domain of finite-domain variable Y. Its inverse, f^{-1}, is potentially a one-to-many mapping. Therefore a constraint expressed on Y will exhibit a solution density for some assignment $Y = b$ which, when transferred to X, should be "shared" between many values.

We define the *multiplicity* of a value $b \in D(Y)$ as $\mu(b) = |\{a \in X \mid f(a) = b\}|$. The implementation of an `element` constraint is augmented with these multiplicities and they are updated whenever the domain of X changes. Given branching variable X_i, we wish to compute the solution density of assignment $X_i = a$ with respect to some constraint $\gamma(Y_1, \ldots, Y_m)$ in whose scope X_i does not appear but to which it is connected through an $\texttt{element}(X_i, f, Y_i)$ constraint:

$$\sigma(X_i, a, \gamma) = \frac{\sigma(Y_i, f(a), \gamma)}{\mu(f(a))}$$

Clearly $\sum_{a \in D(X_i)} \sigma(X_i, a, \gamma) = 1$ since the domain of Y_i is the image of $f(X_i)$ (i.e. the corresponding `element` is enforced) and $\sum_{b \in D(Y_i)} \sigma(Y_i, b, \gamma) = 1$. Also $\sigma(X_i, -, \gamma)$ is nonnegative so it does define a solution density function.

2.2 Generalizing the counting algorithm for `knapsack` constraints

The solution counting algorithm of every family of constraints must be generalized to take into account multiplicities for values. We outline the adaptation for `knapsack` constraints, because these are used in the experimental section. That adaptation will also trivially apply to `regular` constraints since they maintain very similar data structures.

The domain consistency algorithm for `knapsack` builds and maintains a layered digraph that provides a compact encoding of its set of solutions: a path from the first to the last layer spells out a complete satisfying assignment of the variables in its scope, each of its arcs representing an individual assignment [3]. The solution density algorithm counts the number of paths with a certain property, e.g. assigning value a to variable X, through simple recursive formulas

[2]. We generalize the algorithm by introducing multiplicities and adding them as labels on every arc in the graph. When counting paths, they are used as a multiplicative factor. For example, the formula for the number of incoming paths from the initial layer to vertex b in the i^{th} layer, $ip(i, b)$, is generalized as follows:

$$ip(0, 0) = 1$$
$$ip(i, b) = \sum_{(v_{i-1,b'}, v_{i,b}) \in Arcs} \mu((v_{i-1,b'}, v_{i,b})) \times ip(i-1, b'), \quad 1 \leq i \leq m$$

3 Stronger Back-Propagation from the Cost Variable

CP typically solves combinatorial optimization problems as a succession of feasibility problems with increasingly tighter bounds on the objective. Many optimization problems have a linear objective function. When such an objective is expressed as a weighted sum constraint in CP, back-propagation from the cost variable (or the bound on cost) to the decision variables is notoriously weak because inexpensive bounds consistency is typically enforced. Using a `knapsack` constraint can strengthen back-propagation but potentially at a prohibitive computational cost since its propagation algorithm is pseudo-polynomial. To make matters worse, individual cost values are often indexed by decision variables instead of being multiplied by them. For example an objective function such as $\sum_{ij} c_{ij} X_{ij}$ with 0-1 variables will instead be expressed as $\sum_i c_{i,S_i}$ with finite domain variables S_i such that $X_{ij} = 1 \Leftrightarrow S_i = j$. In practice one states `element` constraints linking each S_i to an auxiliary variable which represents its individual cost, the latter variable appearing in the sum constraint stating the objective.

The approach described in the previous section provides the means to handle combinatorial optimization problems more directly with counting-based heuristics. Solution densities from the "objective" constraint will be expressed directly on the main decision variables and put in the mix with other solution densities coming from feasibility considerations. They indicate which assignments often appear in "good enough" solutions, according to that constraint on the objective value.

4 Experiments: Capacitated Facility Location

We ran experiments on the proposed approach using the Capacitated Facility Location Problem (CSPLib problem 34). We are given a set of locations where we may open a facility, each with a maximum capacity c_i, to serve a set of customers, each with a demand d_j. The problem consists of assigning each customer to a single facility while respecting the capacity constraints and minimizing the total cost of the assignment. That cost is the sum of relevant service costs s_{ij}, defined on location-customer pairs and of set-up costs o_i for each open facility (i.e. a location with at least one customer assigned to it). We frame it as a feasibility problem by adding a constraint upper bounding that cost. This corresponds to a single step in the usual CP approach to optimization.

We use a set of 12 relatively small (10 facilities; 50 customers) instances from Holmberg [1]. As for branching heuristics, we compared maxSD to some of the best known generic heuristics in CP (dom/ddeg and IBM-Ilog Solver 6.3 IBS[3]).

$$\texttt{knapsack}((X_{ij})_{1\le i\le m}, \mathbf{1}, 1, 1) \qquad 1 \le j \le n \tag{1}$$

$$\texttt{knapsack}((X_{ij})_{1\le j\le n}, (d_j)_{1\le j\le n}, 0, c_i) \qquad 1 \le i \le m \tag{2}$$

$$O_i \le \sum_{1\le j\le n} X_{ij} \le nO_i \qquad 1 \le i \le m \tag{3}$$

$$\sum_{1\le i\le m, 1\le j\le n} s_{ij}X_{ij} + \sum_{1\le i\le m} o_iO_i \le \beta \tag{4}$$

$$X_{ij} \in \{0,1\} \qquad 1 \le i \le m, \quad 1 \le j \le n \tag{5}$$

$$O_i \in \{0,1\} \qquad 1 \le i \le m \tag{6}$$

Consider a general instance with n customers and m locations for the facilities. The usual 0-1 model (1)-(6) does not behave well in CP (binary variables are usually avoided, with good reason) and even though it features knapsack constraints, only the packing constraints (2) provide non trivial solution densities to be used by counting-based heuristics. On this model, none of the branching heuristics produced any solution within the one-hour time limit.

$$\texttt{alldiff}(C_{ij}) \tag{7}$$

$$\texttt{knapsack}((D_{ij})_{1\le j\le k}, \mathbf{1}, 0, c_i) \qquad 1 \le i \le m \tag{8}$$

$$\texttt{element}(C_{ij}, (d_\ell)_{1\le \ell\le mk}, D_{ij}) \qquad 1 \le i \le m, \quad 1 \le j \le k \tag{9}$$

$$\texttt{element}(C_{ij}, (s_{i\ell})_{1\le \ell\le mk}, S_{ij}) \qquad 1 \le i \le m, \quad 1 \le j \le k \tag{10}$$

$$C_{ij} < C_{i,j+1} \qquad 1 \le i \le m, \quad 1 \le j \le k-1 \tag{11}$$

$$(O_i = 1) \Leftrightarrow (C_{i1} \le n) \qquad 1 \le i \le m \tag{12}$$

$$\sum_{1\le i\le m, 1\le j\le k} S_{ij} + \sum_{1\le i\le m} o_iO_i \le \beta \tag{13}$$

$$C_{ij} \in \{0, 1, \ldots, mk\} \qquad 1 \le i \le m, \quad 1 \le j \le k \tag{14}$$

$$D_{ij} \in \{d_{j'} \mid 1\le j'\le n\} \cup \{0\} \qquad 1 \le i \le m, \quad 1 \le j \le k \tag{15}$$

$$S_{ij} \in \{s_{i'j'} \mid 1\le i'\le m, 1\le j'\le n\} \cup \{0\} \qquad 1 \le i \le m, \quad 1 \le j \le k \tag{16}$$

$$O_i \in \{0,1\} \qquad 1 \le i \le m \tag{17}$$

A facility-centered model that assigns customers to k predefined slots for each location works much better, at least for counting-based heuristics. Since in general we define more slots than there are customers, we add dummy customers $n+1, n+2, \ldots, mk$ (in our experiments we use $k = 6$). Customer demands and service costs are correspondingly extended with 0's. Note that for instances

[3] Impacts are fully initialized at the root node, approximated impacts are used to preselect a subset of 5 variables, node impacts are computed on that subset and further ties are broken randomly.

whose optimal solutions open relatively few locations, such a model would needlessly create too many slots. Model (7)-(17) features an `alldiff` constraint spanning the main decision variables C_{ij} and `knapsack` packing constraints for each facility. However these latter ones are expressed on auxiliary variables D_{ij}. Constraints (11) break customer symmetries at a facility, (12) link the O_i variables, and (13) states a bound on the objective value.

Table 1. Performance of maxSD given different sources of solution densities

source of solution density	time(s) avg	time(s) median	backtracks avg	backtracks median	obj value avg
C_{ij}	55.6	27.6	40744.5	19680	16354
$C_{ij} + D_{ij}$	34.9	9.2	19191.3	5749	16354
C_{ij} + eltSD; 1.2	17.7	7.7	1243.9	4	14309
C_{ij} + eltSD; 1.0	22.7	5.8	2323.6	5	12856

Heuristic dom/ddeg still does not produce any solution within the one-hour time limit while IBS finds a solution to 4 out of 12 instances. Randomized restarts [4] help IBS solve more instances, though not all of them. maxSD performs much better, as indicated in Table 1 (and here randomized restarts generally worsened performance). In all four variants, the upper bound on cost was set to 1.8 times the optimal value. The first two do not use augmented `element` constraints (the second variant branches on both C_{ij} and D_{ij} variables) whereas the last two do use them, both for the knapsack packing constraints and for the service costs. For these service costs, individual knapsack constraints were stated for each facility and the upper bound was set to 1.2 (third variant) and 1.0 (fourth variant) times the optimal value divided by m, the number of facilities. We observe that the additional guidance provided by the indirect solution densities does decrease search effort and also contributes to improve the cost of the solutions found. Note that lowering the shared 1.8 bound or adding a bound per facility as above significantly increased search effort for the first two variants and caused several instances to time out.

5 Conclusion

In this paper, we proposed two generalizations of counting algorithms to overcome the limit of having solution density branching information on auxiliary variables. These algorithms applied to a benchmark problem showed promising results, underlining once more the efficiency of counting-based heuristics. Among the possible improvements, the approximated algorithm for computing the solution densities of the `knapsack` constraint [2] can help whenever the upper bound on the knapsack is quite large. We plan to extend this approach to other constraints such as `alldifferent` or `global-cardinality`; a possible avenue would be to use the multiplicities as entries of the adjacency matrix and compute the solution density with permanent upper bounds for generic nonnegative matrices [6].

References

1. Kaj Holmberg, Mikael Ronnqvist, and Di Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113(3):544–559, March 1999.
2. Gilles Pesant and Claude-Guy Quimper. Counting solutions of knapsack constraints. In Laurent Perron and Michael A. Trick, editors, *CPAIOR*, volume 5015 of *Lecture Notes in Computer Science*, pages 203–217. Springer, 2008.
3. Michael A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118:73–84, 2003.
4. Toby Walsh. Search in a small world. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, IJCAI '99, pages 1172–1177, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
5. Alessandro Zanarini and Gilles Pesant. Solution Counting Algorithms for Constraint-Centered Search Heuristics. *Constraints*, 14:392–413, 2009.
6. Alessandro Zanarini and Gilles Pesant. More robust counting-based search heuristics with alldifferent constraints. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *CPAIOR*, volume 6140 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2010.