Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Counting-Based Search for Constraint Optimization Problems

Gilles Pesant
École Polytechnique de Montréal, Montreal, Canada
CIRRELT, Université de Montréal, Montreal, Canada
gilles.pesant@polymtl.ca

Abstract

Branching heuristics based on counting solutions in constraints have been quite good at guiding search to solve constraint satisfaction problems. But do they perform as well for constraint optimization problems? We propose an adaptation of counting-based search for optimization, show how to modify solution density computation for some of the most frequently-occurring constraints, and empirically evaluate its performance on several benchmark problems.

Introduction

Designing robust generic branching heuristics (e.g. (Boussemart et al. 2004; Refalo 2004; Zanarini and Pesant 2007; Michel and Van Hentenryck 2012)) is an important step in making Constraint Programming (CP) competitive with other approaches to combinatorial problem solving which adopt a "model-and-solve" philosophy, without the user having to design a dedicated search strategy. Among such branching heuristics, *counting-based search* exploits the structure of a model by querying each constraint about how frequently a given variable-value assignment appears in solutions. Branching heuristics based on such information guide search towards areas of the search space likely to contain solutions. These heuristics perform very well to solve constraint *satisfaction* problems (Pesant, Quimper, and Zanarini 2012).

But many combinatorial problems seek a solution that is optimal with respect to some objective and are more naturally framed as constraint *optimization* problems. How well does counting-based search perform when we are not merely looking for any solution but for an optimal one? Unless the best solutions are uniformly distributed in the space of solutions, orienting the search where most solutions are may not guide us very well. In this paper we investigate an adaptation of counting-based search for combinatorial optimization problems. The next section provides some background concepts and reviews the relevant literature. Then we present our contribution of counting-based search for optimization. Finally we describe its empirical evaluation.

Background

In CP a combinatorial problem is represented using a finite set of discrete variables $X = \{x_1, x_2, \ldots, x_n\}$ each taking its value from a finite domain, $x_i \in D_i \subset \mathbb{Z},\ 1 \leq i \leq n$, and a finite set of constraints (i.e. relations) $C = \{c_1, c_2, \ldots, c_m\}$ each expressed on a subset of the variables, $c_j(x_{j_1}, x_{j_2}, \ldots, x_{j_k}) \subset \mathbb{Z}^k,\ 1 \leq j \leq m$. One must find a combination of values from the domain of each variable that simultaneously satisfies every constraint (i.e. belongs to every relation). This formalism is called the *Constraint Satisfaction Problem*. So at its core CP is designed to handle combinatorial existence problems.

Although constraints can be arbitrary relations, typically they each correspond to an important combinatorial substructure of the problem. This explicit access to structure has been the cornerstone of inference in CP , giving rise to many distinct filtering algorithms encapsulated in so-called global constraints, and can also be used to guide search. Generally the exploration of the solution space takes the form of a search tree in which branches correspond to fixing some variable to a value in its domain. This decision process is broken down into a variable-selection heuristic and a value-selection heuristic.

Counting-based search relies on computing the solution density of each variable-value assignment for a constraint in order to build an integrated variable-selection and value-selection heuristic (Zanarini and Pesant 2007). Given a constraint $c(x_1, \ldots, x_k)$, its number of solutions $\#c(x_1, \ldots, x_k)$, respective finite domains $D_i\ {}_{1 \leq i \leq k}$, a variable x_i in the scope of c, and a value $d \in D_i$, we call

$$\sigma(x_i, d, c) = \frac{\#c(x_1, \ldots, x_{i-1}, d, x_{i+1}, \ldots, x_k)}{\#c(x_1, \ldots, x_k)}$$

the *solution density* of pair (x_i, d) in c. It measures how often a certain assignment is part of a solution to c. We can exploit the combinatorial structure of the constraint to design efficient algorithms computing solution densities. Generally speaking such a problem falls within the realm of enumerative combinatorics and is even harder than the usual filtering to achieve consistency, akin to existential combinatorics.

Constraint *optimization* problems are traditionally approached as a succession of constraint satisfaction problems, raising the bar for the objective value at each step until the

last step which is a proof of optimality (alternatively dichotomic search on the objective has also been used). We typically define an additional variable, say z, representing the objective value and add a constraint to link it to the objective function. Whenever a solution is found, thereby fixing z to some value v, the constraint $z < v$ (if we are minimizing) is automatically added in order to constrain the search to better solutions until none can be found, proving the optimality of the latest solution. Because filtering algorithms are a multiway process, domain changes for the variables in X can trigger changes in z's domain and changes in the latter, typically when we add a stronger bound, can in turn impact X. However this so-called *back-propagation* may be rather weak, especially when the objective function is expressed as a sum. As for guiding search, whenever the objective function is separable, i.e. expressed as a sum of costs on individual variables, (Caseau and Laburthe 1997) propose the generic *regret* branching heuristic in which we select a variable maximizing the difference between its best and second best cost for a value in its domain and select the value featuring that best cost.

Optimization constraints were defined in an effort to encapsulate optimization aspects in constraints. Given a constraint $c(x_1, \ldots, x_k)$, let $f : \mathcal{D} \rightarrow \mathbb{R}$ (with $\mathcal{D} = D_1 \times \cdots \times D_k$) associate a cost to each combination of values for the variables appearing in that constraint and $z \in [\min_{t \in \mathcal{D}} f(t), \max_{t \in \mathcal{D}} f(t)]$ be a bounded-domain continuous variable. An *optimization constraint* $c^\star(x_1, x_2, \ldots, x_k, z, f)$ holds if $c(x_1, x_2, \ldots, x_k)$ is satisfied and $z = f(x_1, x_2, \ldots, x_k)$. This is a useful concept if the objective function of the problem can be expressed using the z variables of some optimization constraints (or, even better, a single one). (Focacci, Lodi, and Milano 1999) introduce *cost-based domain filtering* to remove values based on optimality reasoning: a relaxation of the optimization constraint is solved to optimality and a gradient function on variable-value pairs is used to identify assignments which cannot lead to a solution better than the current best one. This typically yields stronger back-propagation. Of particular interest here, they also use the gradient to guide search: they investigate a variable-selection heuristic computing regrets based on gradient information and a value-selection heuristic favouring the assignments exhibited in the optimal solution of the relaxed constraint. Their paper instantiates this idea for the `alldifferent` and `path` constraints. Follow up work includes (Sellmann 2003) for the `shorter_path` constraint and (Demassey, Pesant, and Rousseau 2006) for the `cost_regular` constraint. Still in the context of optimization constraints, (Zanarini 2010) proposes using the average cost of solutions featuring a given variable-value pair as a branching heuristic for optimization problems.

Solution Densities of Best Solutions

Past efforts to guide search in CP when confronted with a combinatorial optimization problem are not fully satisfactory. The regret heuristic only applies if the objective function is separable and is overly optimistic since it implicitly assumes that the best and second best values of a given variable are compatible with the same combinations of values for the other variables, thereby justifying the evaluation of a cost difference based solely on the value taken by that variable.

Using a gradient function to compute regrets is an improvement because cost is taken into account in a more global way (since it does so with respect to an optimization constraint representing a combinatorial substructure) but in general it comes from solving a relaxation of the problem so its accuracy depends on the strength of the relaxation. It is also rather flat information: two variables of equal gradient-based regret may have best values of very different robustness, i.e. one may be compatible with many more combinations of values for the other variables while retaining the optimal cost. The whole idea behind regret is "If that best value becomes incompatible because of future choices for the other variables, what will be the impact on cost?". Therefore it could be useful to know how frequently that best value occurs in very good solutions to the constraint.

We propose to generalize the concept of solution density presented in the previous section to that of cost-based solution density. Without loss of generality we consider minimization problems. Let $\epsilon \geq 0$ be a small real number and $\#c^\star_\epsilon(x_1, \ldots, x_k, z, f)$ denote the number of solutions to $c^\star(x_1, \ldots, x_k, z, f)$ with $z \leq (1+\epsilon) \cdot \min_{t \in c(x_1, \ldots, x_k)} f(t)$. We will call

$$\sigma^\star(x_i, d, c^\star, \epsilon) = \frac{\#c^\star_\epsilon(x_1, \ldots, x_{i-1}, d, x_{i+1}, \ldots, x_k, z, f)}{\#c^\star_\epsilon(x_1, \ldots, x_k, z, f)}$$

the *cost-based solution density* of pair (x_i, d) in $c^\star$. If $\epsilon = 0$ this corresponds to the solution density over the optimal solutions to the constraint with respect to f and if there is a single optimal solution then this identifies the corresponding assignment to x_i with a solution density of 1. A positive ϵ gives a margin to include close-to-optimal solutions. Among the good solutions being considered we favour a variable-value pair that occurs frequently, aiming to branch on a subproblem that will still contain several good solutions with respect to that constraint because we recognize that there are likely other constraints to satisfy. This is very much in the spirit of counting-based search for satisfaction problems.

The following subsections describe cost-based solution-density algorithms for a few common constraint families.

`MinWeightAlldifferent` constraints

Constraint `alldifferent`(X), requiring that all variables in X take distinct values, is probably the most ubiquitous constraint in CP. The original domain filtering algorithm exploits the correspondence between its solutions and maximum matchings in an associated bipartite graph (Régin 1994). Computing solution densities then translates to counting such matchings, which in turn corresponds to computing the permanent of a 0-1 matrix (the adjacency matrix of the bipartite graph), well known to be $\#P$-complete (Valiant 1979). The permanent of a $n \times n$ matrix $A = (a_{ij})$ is defined as

$$\text{per}(A) = \Sigma_{p \in \mathcal{P}} \Pi_{i=1}^{n} a_{i,p(i)}$$

where $\mathcal{P}$ denotes the set of all permutations of $1..n$. With A as the adjacency matrix it is not hard to see that a given

product inside that sum will be equal to 1 if and only if the corresponding entries denote a matching, and so that sum gives the number of matchings. Exact algorithms for the permanent being too time consuming, a few heuristic methods have been proposed. Among them the one computing a ratio of upper bounds on the permanent of 0-1 matrices offers the best compromise between runtime and accuracy (Zanarini and Pesant 2010).

The `MinWeightAlldifferent`(X, z, Γ) optimization constraint adds a cost variable z equal to the sum of the variable-value assignment costs given by cost matrix Γ (Caseau and Laburthe 1997). We build on the previous idea of using upper bounds on the permanent but apply it to "cost-aware" matrices. As a first step we compute a minimum-weight bipartite matching, say of cost $z^\star$, using the Hungarian algorithm as did (Focacci, Lodi, and Milano 1999). As a by-product of this computation we get a *reduced-cost matrix* with at least one zero entry in each row and column identifying the minimum-weight matching and all other entries nonnegative. Entry a_{ij} tells how much of an increase in cost we can expect if we assign value j to x_i instead of the value from the computed matching. As a second step we apply the following transformation

$$a_{ij} \leftarrow \max(0, \frac{(\epsilon z^\star + 1) - a_{ij}}{\epsilon z^\star + 1})$$

which maps the entries of the matrix to the real interval $[0, 1]$, with value 1 corresponding to a reduced cost of 0 and value 0 corresponding to any reduced cost signalling a variable-value assignment whose cost would exceed our ϵ margin. Now the permanent of this transformed matrix can be used as an approximation of the number of solutions of cost at most $(1 + \epsilon)z^\star$, as required. Since upper bounds on the permanent of more general nonnegative matrices are also known (e.g. (Soules 2003)) as a final step we approximate solution densities as a ratio of such upper bounds (here bound U^1 from the previous reference).

`cost_regular` constraints

Constraint `regular`$(X, \mathcal{A})$ holds if the values taken by the sequence of variables X spell out a word belonging to the regular language described by the automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ (Pesant 2004). It provides a very flexible way of expressing rules about sequences of values appearing in a solution. The domain filtering algorithm associated with this constraint is based on the computation of paths in a layered directed acyclic graph generated by unfolding the automaton on X: each such path corresponds to a solution to the constraint but since paths may share arcs this representation of solutions is compact (in fact, the total number of arcs is at most linear in the number of states in $\mathcal{A}$, variables, and values). A variable-value pair is represented by arcs in the graph and is supported by paths through these arcs; otherwise it can be filtered out. Since we have on hand an explicit representation of every solution, (Zanarini and Pesant 2007) compute exact solution densities by storing at each vertex of the graph the number of partial paths from the first and to the last layers: the number of solutions featuring a given variable-value pair equals the sum over all corresponding arcs of the product of the appropriate number of partial paths at their endpoints. The solution density is then the ratio of that number to the total number of solutions.

Optimization constraint `cost_regular`$(X, \mathcal{A}, z, \Gamma)$ adds a cost variable z equal to the sum of the state-variable-value transition costs given by cost matrix $\Gamma = (\gamma_{qxd})$ (Demassey, Pesant, and Rousseau 2006). Observe that these costs depend not only on the variable-value assignment $(x = d)$ but also on the current state q of the automaton, making them possibly sequence-dependent. The cost-based domain filtering algorithm associated with this constraint is based on the computation of *weighted* paths in the same graph. At each vertex we store as well the length of the shortest and longest partial paths from the first and to the last layers. Then in time linear in the number of arcs corresponding to a given variable-value pair we can discard that value from the domain if every such arc belongs to paths (i.e. solutions) that are either too short or too long with respect to the bounds on z. If z is only bounded from below or above but not both, we achieve domain consistency on X. (We refer the reader to the previous references for details.)

In keeping with minimization problems consider an upper bounded cost variable z. Extending the above to cost-based solution densities with $\epsilon = 0$ is a simple matter of storing instead the number of *shortest* partial paths since in a directed acyclic graph shortest paths are composed of shortest partial paths. The number of shortest paths featuring a given variable-value pair then becomes the sum over all corresponding arcs of the product of the appropriate number of shortest partial paths at their endpoints, provided their composition makes a shortest path. For $\epsilon > 0$ we need to restrict our attention to solutions whose cost lies within $1 + \epsilon$ of the lowest cost for a solution. The number of shortest partial paths is no longer sufficient: we need to keep track of the number of partial paths of each different length (up to some threshold) because these could be combined into paths within the $1 + \epsilon$ margin.

We use the same data structures described to compute solution densities for `dispersion` constraints (Pesant 2015), adapted here for `cost_regular`. As before we store at each vertex some information about partial paths from the first and to the last layers: each is a list of pairs recording how many paths of each length there are. The list related to paths from the first layer to vertex (i, q), corresponding to state q in layer i, is denoted $p_{iq}^{\rightarrow} = \langle \ldots, (\ell, m), \ldots \rangle$, each entry (ℓ, m) recording that m partial paths of length *at most* ℓ are achieved by some assignment of $x_1, \ldots, x_{i-1}$ reaching state q. This list is in strict decreasing order of length. The list related to paths from vertex (i, q) to the last layer is denoted $p_{iq}^{\leftarrow} = \langle \ldots, (\ell', m'), \ldots \rangle$ and each entry (ℓ', m') records that m' partial paths of length *exactly* ℓ' are achieved by some assignment of $x_i, \ldots, x_n$ from state q. That list is in strict increasing order of length. Note that provided the costs are nonnegative we do not need to record partial paths of length greater than $(1 + \epsilon)\ell^\star$ where $\ell^\star$ is the length of a shortest path in the graph. These lists can be built through a forward and backward breadth-first traversal of the graph.

Algorithm 1 describes how we compute cost-based solution densities from them. Lines 1-13 compute the number of solutions featuring $x_i = d$ and whose cost is within the margin: in particular at Line 2 we retrieve the length of a shortest path $\ell^\star$; the loops at Lines 5-13 iterate through the lists at the endpoints of a given arc, finding partial paths that combine into full paths through that arc and that are short enough, and adding them up (Line 13). Lines 14-18 compute the total number of solutions within the margin by adding to the number of shortest paths others from that list that lie within that margin. Finally Line 19 returns the desired ratio.

Algorithm 1: Cost-based solution density algorithm for the `cost_regular` constraint

input: variable index i, value d, relative margin ϵ
output: $\sigma^\star(x_i, d, \texttt{cost\_regular}(X, \mathcal{A}, z, \Gamma), \epsilon)$

1 $a \leftarrow 0$;
2 $(\ell^\star, m^\star) \leftarrow$ first entry of list $p^{\leftarrow}_{0q_0}$;
3 **forall the** *arcs between a vertex* (i, q) *for some* q *and vertex* $(i+1, \delta(q,d))$ **do**
4 $\quad (\ell, m) \leftarrow$ first entry of list $p^{\rightarrow}_{iq}$;
5 $\quad$ **forall the** *entries* (ℓ', m') *in list* $p^{\leftarrow}_{i+1,\delta(q,d)}$ **do**
6 $\quad\quad$ **while** $(\ell + \gamma_{qx_id} + \ell' > (1+\epsilon)\ell^\star)$ **do**
7 $\quad\quad\quad$ **if** $(p^{\rightarrow}_{iq}$ *is not exhausted* **then**
8 $\quad\quad\quad\quad (\ell, m) \leftarrow$ next entry of $p^{\rightarrow}_{iq}$;
9 $\quad\quad\quad$ **else**
10 $\quad\quad\quad\quad$ break;
11 $\quad\quad$ **if** $p^{\rightarrow}_{iq}$ *is exhausted* **then**
12 $\quad\quad\quad$ break;
13 $\quad\quad a \leftarrow a + m \times m'$;
14 $b \leftarrow m^\star$;
15 $(\ell', m') \leftarrow$ next entry of list $p^{\leftarrow}_{0q_0}$;
16 **while** $(\ell' \leq (1+\epsilon)\ell^\star)$ **do**
17 $\quad b \leftarrow b + m'$;
18 $\quad (\ell', m') \leftarrow$ next entry of list $p^{\leftarrow}_{0q_0}$;
19 **return** $\frac{a}{b}$;

`dispersion` constraints

One is sometimes asked to balance a given feature of solutions to a combinatorial problem so that a certain level of fairness is achieved. For example this could be the individual workload of nurses in an intensive care unit (Schaus, Van Hentenryck, and Régin 2009) or the number of credits per semester in an academic curriculum. Often balance is even formulated as the objective to optimize. The `spread` (Pesant and Régin 2005) and `deviation` (Schaus et al. 2007) constraints were introduced to constrain the mean of a set of integer variables and the deviation from that mean. (Pesant 2015) recently introduced a unified approach that achieves stronger domain filtering using the concept of L_p-*deviation* of a set of integers $\{x_1, x_2, \ldots, x_n\}$ from their arithmetic mean μ, defined as $\sum_{i=1}^{n} |x_i - \mu|^p$. Constraint $\texttt{dispersion}(X, \mu, \Delta, p)$ states that the collection of values taken by the variables of X exhibits an arithmetic mean μ and an L_p-deviation Δ. Cases $p = 1$ and $p = 2$ respectively generalize `deviation` and `spread`. Note that Δ can be viewed as a cost variable and hence `dispersion` as an optimization constraint whose cost function f is implicit.

As in the case of `cost_regular`, solutions can be represented as paths in a directed acyclic graph. The cost-based solution density algorithm for `dispersion` is essentially the same as Algorithm 1. Note however that its running time is pseudo-polynomial because the size of the graph is linearly related to the span of the domains.

Experiments

This section presents an empirical evaluation of the search guidance efficiency of a branching heuristic built from our cost-based solution densities on three benchmark problems, one for each of the optimization constraints considered in the previous section. For constraint satisfaction problems, branching heuristic maxSD makes an integrated choice of the variable and value with the highest solution density over all the constraints of a model (Zanarini and Pesant 2007); for constraint optimization problems we introduce maxSD*, branching on the variable and value with the highest cost-based solution density over all the optimization constraints of a model.

All experiments were run on Dual core AMD 2.1 GHz processors with 8 GB of RAM, using IBM ILOG Solver 6.7 as the CP solver, and each instance was given a two-hour time limit. Each experiment uses depth-first search and compares maxSD* (with $\epsilon = 0.1$) to standard generic branching heuristics, namely smallest-domain first with lexicographic value selection (dom) and the solver's default impact-based search (IBS), and to some tailored heuristic when applicable. We report the percentage of instances solved to optimality (or to within a given gap) as a function of computation time.

Balanced Academic Curriculum Problem

The Balanced Academic Curriculum Problem (BACP, problem 30 of the CSPlib) asks to assign courses to semesters so as to respect prerequisites between courses, a minimum and maximum number of courses per semester, and to balance the academic load (total number of credits) between semesters. (Schaus 2009) generated 100 instances by randomly assigning between 1 and 5 credits to courses and by randomly choosing a subset of the prerequisites of a large instance from the literature. They feature 66 courses, 12 semesters, and 50 prerequisite pairs. We require between 4 and 10 courses per semester as suggested for that large instance. Table 1 presents a model for this problem using one variable s_i per course i indicating which semester it is assigned to and one variable ℓ_j per semester j indicating its academic load. P denotes the set of prerequisite pairs and $A = (a_i)$ the array of credits per course. Cost variable z, to minimize, represents the L_2-deviation of the ℓ_j values from the known average academic load. A global cardinality constraint (gcc) limits the number of courses per semester and a bin packing constraint links the s_i and ℓ_j variables while considering course credits.

(Schaus 2009) proposed a branching heuristic tailored to

$\min z$ s.t.

$\texttt{dispersion}(\{\ell_j\}, (\sum_{i=1}^{66} a_i)/12, z, 2)$

$\texttt{gcc}(\{s_i\}, \langle [4,10], \ldots, [4,10] \rangle)$

$\texttt{binpacking}(\langle s_1, \ldots, s_{66} \rangle, A, \langle \ell_1, \ldots, \ell_{12} \rangle)$

$s_i < s_{i'}$ $\quad (i,i') \in P$

$s_i \in \{1,2,\ldots,12\}$ $\quad 1 \leq i \leq 66$

$\ell_j \in \{0,1,\ldots,\sum_{i=1}^{66} a_i\}$ $\quad 1 \leq j \leq 12$

$z \in \mathbb{R}^+$

Table 1: A CP model for the BACP

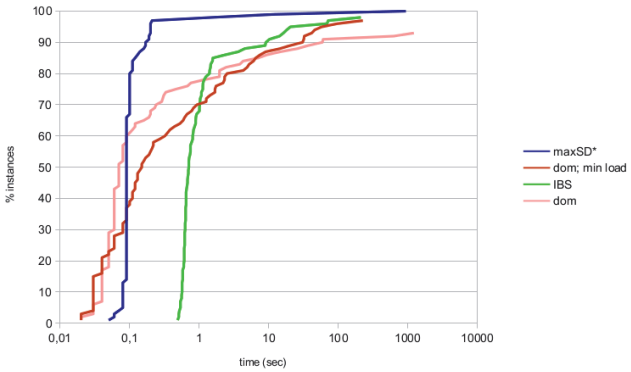

Figure 1: Percentage of BACP instances solved to optimality with respect to time for a few heuristics.

this problem: choose the next s_i variable to branch on according to the smallest-domain-first selection criterion and the next value (semester) to assign to it by favouring the semester currently with the smallest academic load (we denote it as "dom; min load"). In this model we have a single optimization constraint, `dispersion`, whose cost variable corresponds to the objective function and which provides cost-based solution densities for the ℓ_j variables. Our maxSD* heuristic first branches on these ℓ_j variables and then on the s_i variables using the above tailored heuristic.

Figure 1 compares branching heuristics by plotting the number of instances solved to optimality with respect to the computation time spent on individual instances. Every branching heuristic solves about three quarters of the instances in under a second but maxSD* is the only one that is able to solve every instance within the time limit and it solves all but three of them in under 0.2 second. If we consider pure search guidance ability (not shown in the figure), maxSD* requires about two orders of magnitude fewer backtracks during search than the other heuristics to reach the 75% mark: since it is still about one order of magnitude faster we may deduce that it spends about one order of magnitude more time per search node. But its much better guidance still pays off overall, including against a heuristic tailored to this problem.

Traveling Salesman Problem

We evaluate the cost-based solution density algorithm proposed for `MinWeightAlldifferent` using the well-known Traveling Salesman Problem (TSP) and a basic CP model featuring the latter constraint both for the assignment part

$\min z = \sum_{i=1}^{n} \gamma_{is_i}$ s.t.

$\texttt{MinWeightAlldifferent}(\{s_1,\ldots,s_n\}, z, \Gamma)$

$\texttt{noCycle}(\{s_1,\ldots,s_n\})$

$s_i \in \{2,3,\ldots,n+1\}$ $\quad 1 \leq i \leq n$

$z \in \mathbb{N}$

Table 2: A basic CP model for the TSP

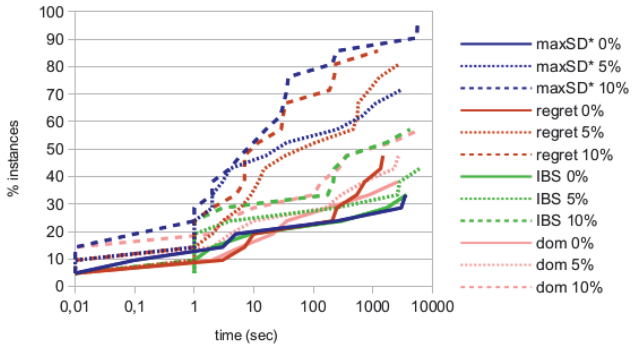

Figure 2: Percentage of TSP instances solved to within $p\%$ of the optimal value with respect to time for a few heuristics.

of the successor variables s_i and to link them to the objective variable z, as well as the `noCycle` subtour elimination constraint (see Table 2). Cities are numbered 1 through n and, as is customary in CP models for this problem, we start a tour at city 1 and end it at city $n+1$, which is a copy of city 1 (thus strictly speaking we are building a Hamiltonian path). Γ represents the distance matrix. Domain consistency is maintained on the s_i variables for the "all different" part of `MinWeightAlldifferent` but no cost-based domain filtering is applied aside from the back-propagation through the sum of individual costs. This clearly is not the state of the art to model and solve TSPs using CP but our goal here is to compare branching heuristics on a model in which the `MinWeightAlldifferent` constraint is prominent so this basic model for a much-studied problem serves us well.

Our benchmark set consists of 21 small symmetric and asymmetric instances from TSPlib ranging from 17 to 71 cities. Figure 2 plots for each heuristic the number of instances solved to optimality (0%) with respect to time, but also the number of instances solved to within 5% and 10% of the optimal value to provide a more complete picture. Here we also evaluate the regret heuristic (regret) previously described in the Background Section since the cost function is separable. Looking at the "10%" curves together we notice two distinct groups: those for the cost-aware heuristics maxSD* and regret, and those for dom and IBS. The former perform much better as they find good solutions to about 90% of the instances as opposed to 55%. The same trend can be observed for the "5%" curves, yielding better solutions but for fewer instances.

As for finding optimal solutions, our heuristic does not perform as well as the others. Upon inspection of the number of backtracks, one possible explanation for this is that we spend up to two orders of magnitude more time per

$$
\begin{array}{ll}
\min \sum_{p \in P} b_p \quad \text{s.t.} & \\
\texttt{cost\_regular}(\langle s_{p*} \rangle, \mathcal{A}, b_p, \Gamma) & p \in P \\
\texttt{gcc}(\{s_{p*}\},\ \langle |T| - |M_p|, 1, \ldots, 1 \rangle) & p \in P \\
\texttt{gcc}(\{s_{*t}\},\ \langle c_0, c_1, \ldots, c_{|M|} \rangle) & t \in T \\
s_{pt} \in \{0\} \cup M_p & p \in P, t \in T \\
b_p \in \mathbb{N} & p \in P \\
c_0 \in \{|P| - 2|L|, \ldots, |P|\} & \\
c_m \in \{0, 2\} & m \in M
\end{array}
$$

Table 3: A simplified CP model for the Business-to-Business Meeting Scheduling Problem

search node: maxSD* quickly obtains good solutions but runs out of time to reach optimality whereas dom and IBS start far away but manage to reach optimality on the smaller instances. Indeed there is much less of a difference between their "0%", "5%", and "10%" curves since they remain far from the optimal value on the larger instances.

Business-to-Business Meeting Scheduling

The Business-to-Business Meeting Scheduling Problem (B2B) consists of scheduling meetings between given pairs of participants to an event while taking into account participant availability and accommodation capacity (Bofill et al. 2014). The challenging aspect of this problem is that breaks in a participant's schedule should be avoided (a break is defined as consecutive free time slots between two meetings). It is therefore cast as an optimization problem in which the sum of the number of individual breaks is minimized. Table 3 presents a simplified model that still retains the essential structure of the problem (see (Pesant, Rix, and Rousseau 2015) for more details). Let P be the set of participants, M the set of meetings between pairs of participants, $M_p \subseteq M$ the set of meetings involving participant p, L the set of locations for meetings, and T the set of time slots. Variables s_{pt}, $p \in P, t \in T$ represent what participant p is scheduled to do at time t, value 0 corresponding to no meeting. Variables b_p represent the number of breaks in the schedule for p. In order to link the b_p variables to the main s_{pt} variables we use a `cost_regular` constraint per participant with an automaton $\mathcal{A}$ tracking breaks and assigning a unit cost to each of them through cost matrix Γ. A `gcc` for each participant ensures that all of his meetings are attended and a `gcc` for each time slot ensures that both participants in a pair attend their meeting at the same time and that not too many meetings occur at the same time. This time the overall cost is not captured by a single optimization constraint but by several of them, each concerned with a distinct term of the sum. Note that the regret heuristic cannot be applied since the objective function is not separable over the s_{pt} branching variables.

We use the 20 instances from (Bofill et al. 2015): they feature from 42 to 78 participants, a few hundred meetings, and from 8 to 22 time slots to schedule them. Most of these are quite challenging for our CP model.

Figure 3 plots for each heuristic the number of instances solved to optimality (solid curves) with respect to time, but also the number of instances solved to within 5 and 10 units of the optimal number of breaks. Here we cannot use a percentage gap as like in the previous problem because some of the optimal solutions have 0 break. Heuristic dom performs very poorly, being unable to solve any instance to optimality and only managing to solve two instances to within 10 breaks of the optimum. Heuristic IBS performs better but each of its curves is clearly dominated by that of maxSD* and again the larger spread of the curves for the latter indicates that good solutions are found more quickly. The number of backtracks during search for each heuristic reveals that the time per search node for maxSD* increases by less than one order of magnitude with respect to the others.

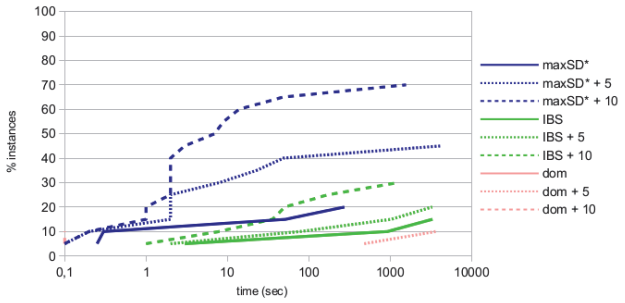

Figure 3: Percentage of B2B instances solved (close) to optimality with respect to time for a few heuristics.

Conclusion

We presented a way to take an idea that is very effective to guide search for solving constraint satisfaction problems, namely counting-based search, and to adapt it to solve constraint optimization problems. We also gave algorithms to realize that adaptation for several useful constraints. Experiments on constraint optimization problems indicate that maxSD* as a generic cost-aware branching heuristic outperforms other generic heuristics and even some tailored heuristics. More computation time is spent before each branching decision but the increased search guidance results in lower total computation times.

In all our experiments we have used a single value for the ϵ parameter included in the concept of cost-based solution density. It would be interesting to run a sensitivity analysis on this parameter with respect to problem type in order to understand better how to set it. Also, computing cost-based solution densities is currently significantly more expensive (by about one order of magnitude) for the `MinWeightAlldifferent` constraint than for `cost_regular` and `dispersion`. One way to improve this that we will investigate is to make some of the computations incremental such as the Hungarian algorithm.

Acknowledgments

Financial support for this research was provided by Discovery Grant 218028/2012 from the Natural Sciences and Engineering Research Council of Canada.

References

Bofill, M.; Espasa, J.; Garcia, M.; Palahí, M.; Suy, J.; and Villaret, M. 2014. Scheduling B2B Meetings. In O'Sullivan, B., ed., *Proc. CP'14*, volume 8656 of *Lecture Notes in Computer Science*, 781–796. Springer.

Bofill, M.; Garcia, M.; Suy, J.; and Villaret, M. 2015. Maxsat-based scheduling of B2B meetings. In Michel (2015), 65–73.

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting Systematic Search by Weighting Constraints. In *Proc. ECAI'04*, 146–150. IOS Press.

Caseau, Y., and Laburthe, F. 1997. Solving various weighted matching problems with constraints. In Smolka, G., ed., *Proc. CP'97*, volume 1330 of *Lecture Notes in Computer Science*, 17–31. Springer.

Demassey, S.; Pesant, G.; and Rousseau, L.-M. 2006. A Cost-Regular Based Hybrid Column Generation Approach. *Constraints* 11(4):315–333.

Focacci, F.; Lodi, A.; and Milano, M. 1999. Cost-based domain filtering. In Jaffar, J., ed., *Proc. CP'99*, volume 1713 of *Lecture Notes in Computer Science*, 189–203. Springer.

Michel, L., and Van Hentenryck, P. 2012. Activity-Based Search for Black-Box Constraint Programming Solvers. In Beldiceanu, N.; Jussien, N.; and Pinson, E., eds., *Proc. CPAIOR'12*, volume 7298 of *Lecture Notes in Computer Science*, 228–243. Springer.

Michel, L., ed. 2015. *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, volume 9075 of *Lecture Notes in Computer Science*. Springer.

Pesant, G., and Régin, J.-C. 2005. sPREAD: A Balancing Constraint Based on Statistics. In van Beek, P., ed., *Proc. CP'05*, volume 3709 of *Lecture Notes in Computer Science*, 460–474. Springer.

Pesant, G.; Quimper, C.-G.; and Zanarini, A. 2012. Counting-Based Search: Branching Heuristics for Constraint Satisfaction Problems. *J. Artif. Intell. Res. (JAIR)* 43:173–210.

Pesant, G.; Rix, G.; and Rousseau, L. 2015. A comparative study of MIP and CP formulations for the B2B scheduling optimization problem. In Michel (2015), 306–321.

Pesant, G. 2004. A Regular Language Membership Constraint for Finite Sequences of Variables. In Wallace, M., ed., *Proc. CP'04*, volume 3258 of *Lecture Notes in Computer Science*, 482–495. Springer.

Pesant, G. 2015. Achieving Domain Consistency and Counting Solutions for Dispersion Constraints. *INFORMS Journal on Computing* 27(4):690–703.

Refalo, P. 2004. Impact-Based Search Strategies for Constraint Programming. In *Proc. CP'04*, volume LNCS 3258, 557–571. Springer.

Régin, J. 1994. A Filtering Algorithm for Constraints of Difference in CSPs. In Hayes-Roth, B., and Korf, R. E., eds., *Proc. AAAI'94*, 362–367. AAAI Press / The MIT Press.

Schaus, P.; Deville, Y.; Dupont, P.; and Régin, J.-C. 2007. The Deviation Constraint. In Van Hentenryck, P., and Wolsey, L. A., eds., *Proc. CPAIOR'07*, volume 4510 of *Lecture Notes in Computer Science*, 260–274. Springer.

Schaus, P.; Van Hentenryck, P.; and Régin, J.-C. 2009. Scalable Load Balancing in Nurse to Patient Assignment Problems. In van Hoeve, W. J., and Hooker, J. N., eds., *Proc. CPAIOR'09*, volume 5547 of *Lecture Notes in Computer Science*, 248–262. Springer.

Schaus, P. 2009. *Solving Balancing and Bin-Packing problems with Constraint Programming*. Ph.D. Dissertation, Université catholique de Louvain.

Sellmann, M. 2003. Cost-Based Filtering for Shorter Path Constraints. In Rossi, F., ed., *Proc. CP'03*, volume 2833 of *Lecture Notes in Computer Science*, 694–708. Springer.

Soules, G. 2003. New Permanental Upper Bounds for Nonnegative Matrices. *Linear and Multilinear Algebra* 51(4):319–337.

Valiant, L. 1979. The Complexity of Computing the Permanent. *Theoretical Computer Science* 8(2):189–201.

Zanarini, A., and Pesant, G. 2007. Solution Counting Algorithms for Constraint-Centered Search Heuristics. In Bessiere, C., ed., *Proc. CP'07*, volume 4741 of *Lecture Notes in Computer Science*, 743–757. Springer.

Zanarini, A., and Pesant, G. 2010. More robust counting-based search heuristics with alldifferent constraints. In Lodi, A.; Milano, M.; and Toth, P., eds., *Proc. CPAIOR'10*, volume 6140 of *Lecture Notes in Computer Science*, 354–368. Springer.

Zanarini, A. 2010. *Exploiting Global Constraints for Search and Propagation*. Ph.D. Dissertation, École Polytechnique de Montréal.