Counting Weighted Spanning Trees to Solve Constrained Minimum Spanning Tree Problems

Antoine Delaite and Gilles Pesant

École polytechnique de Montréal, Montreal, Canada
CIRRELT, Université de Montréal, Montreal, Canada
{antoine.delaite,gilles.pesant}@polymtl.ca

Abstract. Building on previous work about counting the number of spanning trees of an unweighted graph, we consider the case of edge-weighted graphs. We present a generalization of the former result to compute in pseudo-polynomial time the exact number of spanning trees of any given weight, and in particular the number of minimum spanning trees. We derive two ways to compute solution densities, one of them exhibiting a polynomial time complexity. These solution densities of individual edges of the graph can be used to sample weighted spanning trees uniformly at random and, in the context of constraint programming, to achieve domain consistency on the binary edge variables and, more importantly, to guide search through counting-based branching heuristics. We exemplify our contribution using constrained minimum spanning tree problems.

1 Introduction

Counting-based search [16] in CP 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 to solve constraint satisfaction problems. Given a constraint $c(x_1, \ldots, x_\ell)$, its number of solutions $\#c(x_1, \ldots, x_\ell)$, respective finite domains D_i $_{1 \leq i \leq \ell}$, a variable x_i in the scope of c, and a value $d \in D_i$, we call

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

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. Such algorithms have already been designed for several families of constraints such as `alldifferent`, `gcc`, `regular`, and `knapsack` [12], `dispersion` [10], and `spanningTree` for unweighted graphs [3].

When faced with optimization problems however, solutions should not be considered on an equal footing since they will have different costs. Let $f : \mathcal{D} \to \mathbb{R}$ (with $\mathcal{D} = D_1 \times \cdots \times D_\ell$) associate a cost to each combination of

values for variables $x_1, \ldots, x_\ell$ 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* $c^\star(x_1, \ldots, x_\ell, z, f)$ holds if $c(x_1, \ldots, x_\ell)$ is satisfied and $z = f(x_1, \ldots, x_\ell)$. 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). Without loss of generality consider minimization problems. Pesant [11] recently generalized the concept of solution density to that of cost-based solution density. Let $\epsilon \geq 0$ be a small real number and $\#c^\star_\epsilon(x_1, \ldots, x_\ell, z, f)$ denote the number of solutions to $c^\star(x_1, \ldots, x_\ell, z, f)$ with $z \leq (1+\epsilon) \cdot \min_{t \in c(x_1, \ldots, x_\ell)} f(t)$. We call

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

the *cost-based solution density* of pair (x_i, d) in $c^\star$. A value of $\epsilon = 0$ 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 since there are likely other constraints to satisfy that will rule out the optimal ones with respect to that single optimization constraint. Algorithms to compute cost-based solution densities have already been given for the optimization versions of the `alldifferent`, `regular`, and `dispersion` constraints [11]. In this spirit we present an algorithm that, given a weighted graph, computes the exact number of spanning trees of any given weight and we contribute cost-aware solution densities for a `spanningTree` constraint.

Section 2 reviews related work on tree constraints in CP. Section 3 presents our counting algorithm for weighted spanning trees. Section 4 derives ways to compute solution densities efficiently. Section 5 gives applications of solution density information. Section 6 reports on the usefulness of this work to solve constrained minimum spanning tree problems.

2 Related work

Previous research in CP on imposed tree structures has focused mostly on filtering algorithms but also includes branching heuristics and explanations. Beldiceanu et al. [2] introduce the tree constraint, which addresses the digraph partitioning problem from a constraint programming perspective. In their work a constraint that enforces a set of vertex-disjoint anti-arborescences is proposed. They achieve domain consistency in $\mathcal{O}(nm)$ time, where n is the number of vertices and m is the number of edges in the graph. Dooms and Katriel [6] introduce the MST constraint, requiring the tree variable to represent a minimum spanning tree of the graph on which the constraint is defined. The authors propose polytime bound consistent filtering algorithms for several restrictions of this constraint. Later Dooms and Katriel [7] propose a weighted spanning tree constraint, in which both the tree and the weight of the edges are variables, and consider several filtering algorithms. The latter is improved by Régin [13], who proposes an

incremental domain consistency algorithm running in $\mathcal{O}(m + n \log n)$ time. Subsequently, Régin et al. [14] improve the time complexity further and also consider mandatory edges. De Uña et al.[5] generate explanations for cost-based failure and filtering that occurs in the weighted spanning tree constraint and feed them to a SAT solver. They show empirically that such an approach greatly improves our ability to solve diameter-constrained minimum spanning tree problems in CP. Brockbank et al.[3] build on Kirchhoff's Matrix-Tree Theorem to derive solution densities for the edges of an unweighted graph and use them in a counting-based branching heuristic. In the next sections we generalize this approach for edge-weighted graphs.

3 Counting Weighted Spanning Trees

In the case of an unweighted graph G on n vertices and m edges Kirchhoff's Matrix-Tree Theorem [15] equates the number of spanning trees of G to the absolute value of the determinant of a sub-matrix (the (i, j)-minor) of the *Laplacian matrix* of G, obtained by subtracting the adjacency matrix of G from the diagonal matrix whose i^{th} entry is equal to the degree of vertex v_i in G. Hence the number of spanning trees can be computed as the determinant of a $(n-1)\times(n-1)$ matrix, in $\mathcal{O}(n^3)$ time.

Let G now be an edge-weighted multigraph with vertex set V, edge set E, and weight function $w : E \to \mathbb{N}$. Following in Kirchhoff's footsteps Broder and Mayr [4] introduce a derived matrix $M = (m_{ij})_{1\leq i,j\leq n}$ whose elements are univariate polynomials built from the edges of G:[1]

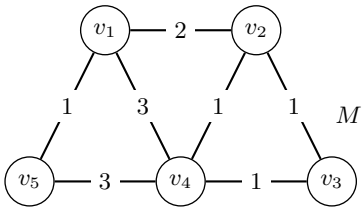
$$m_{ij} = \begin{cases} \sum_{e=(v_i,v_j)\in E} -x^{w(e)}, & i \neq j \\ \sum_{e=(v_i,v_k)\in E} x^{w(e)}, & i = j \end{cases} \tag{1}$$

Off-diagonal entries m_{ij} add one monomial per edge between distinct vertices v_i and v_j (there may be several since we consider multigraphs) whereas entries m_{ii} on the diagonal add one monomial per edge incident with vertex v_i.

Figure 1 shows a small graph and its derived matrix M. Observe that if all weights are 0 (or alternatively if we set x to 1) matrix M simplifies to the previously defined Laplacian matrix: it is thus a true generalization of the unweighted case. If we remove the i^{th} row and column of M for any i and compute the determinant of the remaining matrix, we obtain a unique polynomial with very useful characteristics: each one of its monomials $a_k x^k$ indicates the number a_k of spanning trees of weight k. For example say we remove the first row and column of M in Figure 1 and compute the determinant:

$$\begin{vmatrix} x^2 + 2x^1 & -x^1 & -x^1 & 0 \\ -x^1 & 2x^1 & -x^1 & 0 \\ -x^1 & -x^1 & 2x^3 + 2x^1 & -x^3 \\ 0 & 0 & -x^3 & x^1 + x^3 \end{vmatrix} = 2x^9 + 3x^8 + 7x^7 + 6x^6 + 3x^5$$

[1] We generalize slightly their definition in order to include multigraphs, which may occur when we contract edges in the context of CP search.



$$M = \begin{pmatrix} x^2+x^3+x^1 & -x^2 & 0 & -x^3 & -x^1 \\ -x^2 & x^2+2x^1 & -x^1 & -x^1 & 0 \\ 0 & -x^1 & 2x^1 & -x^1 & 0 \\ -x^3 & -x^1 & -x^1 & 2x^3+2x^1 & -x^3 \\ -x^1 & 0 & 0 & -x^3 & x^1+x^3 \end{pmatrix}$$

Fig. 1: A graph and its derived matrix.

Thus we discover that there are two spanning trees of weight 9, three of weight 8, seven of weight 7, six of weight 6, and three of (minimum) weight 5. We will call this polynomial the *wst-polynomial* of G and denote it $P_G(x)$. It directly gives us the number of spanning trees of each weight though computing it requires $\Theta(n^3 W \max(n, W))$ time in the worst case, where $W = \max\{w(e) : e \in E\}$ [8]. Thus it is dependent on the magnitude of the edge weights.

Broder and Mayr [4] were interested in improving the efficiency of counting the number of *minimum* spanning trees, say of weight $k^\star$, and for that purpose they provide an algorithm that, guided by an arbitrary spanning tree, takes linear combinations of some of the columns in order to factor out the corresponding power $x^{k^\star}$, and then evaluates the remaining determinant at $x = 0$ thus isolating the coefficient $a_{k^\star}$. The time complexity of their algorithm is the same as that of matrix multiplication (for $n \times n$ matrices).

4 Computing Cost-Aware Solution Densities

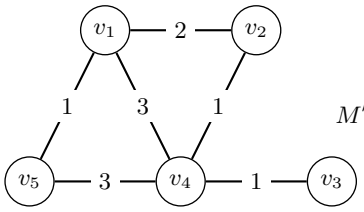
We are interested in counting spanning trees because they are instrumental in establishing the solution density of an edge $e \in E$ given a `spanningTree` constraint on G. A natural way to go about this is to divide the number of spanning trees using that edge by the total number of spanning trees. We first discuss how to compute cost-based solution densities exactly and then present an adaptation of that concept which allows us to lower the time complexity significantly.

4.1 Exact Cost-Based Solution Densities

What is the solution density of a given edge e among all spanning trees of weight k? We can compute the difference between the wst-polynomial of G and that of the same graph without that edge, $P_G(x) - P_{G-e}(x)$. The result is another polynomial giving the number of spanning trees of each weight that use e. From it we can extract the number of spanning trees of weight k which contain e and the corresponding solution densities.

Consider Figure 2 in which we removed edge (v_2, v_3) from G. We have

$$P_{G-(v_2,v_3)}(x) = x^9 + x^8 + 3x^7 + 2x^6 + x^5$$



$$M' = \begin{pmatrix} x^2+x^3+x^1 & -x^2 & 0 & -x^3 & -x^1 \\ -x^2 & x^2+x^1 & 0 & -x^1 & 0 \\ 0 & 0 & x^1 & -x^1 & 0 \\ -x^3 & -x^1 & -x^1 & 2x^3+2x^1 & -x^3 \\ -x^1 & 0 & 0 & -x^3 & x^1+x^3 \end{pmatrix}$$

Fig. 2: The graph at Fig. 1 with edge (v_2, v_3) removed and its derived matrix.

and therefore

$$P_G(x) - P_{G-(v_2,v_3)}(x) = x^9 + 2x^8 + 4x^7 + 4x^6 + 2x^5.$$

So for example edge (v_2, v_3) is used in four spanning trees of weight 7 and the corresponding solution density is $\frac{4}{7}$ since according to $P_G(x)$ there are seven spanning trees of that weight. For cost-based solution densities, we need to take into account the trees of weight at most $(1+\epsilon)k^\star$. In our example $k^\star = 5$ and say we take $\epsilon = 0.2$: there are $4 + 2 = 6$ spanning trees of weight at most $(1 + 0.2) \cdot 5 = 6$ using edge (v_2, v_3) and $6 + 3 = 9$ such trees according to $P_G(x)$ (using that edge or not), yielding a cost-based solution density equal to $\frac{6}{9}$, or $\frac{2}{3}$.

4.2 Cost-Damped Solution Densities

Computing solution densities via wst-polynomials may be too time-consuming for some uses such as in a branching heuristic that is called at every node of the search tree. The success of Broder and Mayr [4] in efficiently computing the number of minimum spanning trees came in large part from the evaluation of the determinant for a fixed x, thus working with scalar values instead of polynomials.

Consider applying an exponential decay of sorts to the number a_k of spanning trees of weight k according to the difference between that weight and that of a minimum spanning tree, $k^\star$, thus giving more importance to close-to-optimal trees. We do this by choosing an appropriate value for x. Consider the wst-polynomial $P_G(x) = \sum_{k=k^\star}^{k^{\max}} a_k x^k$: evaluating it at $x = 1$ yields $P_G(1) = \sum_{k=k^\star}^{k^{\max}} a_k$, the total number of spanning trees regardless of their weight. Using instead some $0 < x < 1$ will have the desired damping effect on the coefficients by applying an additional $x^{k-k^\star}$ factor to a_k relative to $a_{k^\star}$ — the further weight k is from the minimum weight, the smaller the factor. As an illustration if we apply this approach to the example of the previous section with any $x \in [0.3, 0.9]$ we get a cost-damped solution density in the range $[0.62, 0.66]$ (compared to $\frac{2}{3}$ with the previous approach).

To build our matrix M we need to elevate x up to the W^{th} power, which can be done in $\Theta(\log W)$, and add the appropriate power in M for each edge, taking $\Theta(m \log W)$ time overall. Then we have a scalar matrix from which we proceed as

in [3], requiring $\Theta(n^3)$ time to compute the solution density of all the edges incident to a given vertex. So the overall time is in $\mathcal{O}(n^2 \max(n, \log W))$ globally for these incident edges, which is truly polynomial, compared to $\Theta(n^3 W \max(n, W))$ for a single edge if we work directly with the wst-polynomials as described in Section 4.1, and which is pseudo-polynomial. And cost-damped solution densities can be updated incrementally as edges are selected or removed from G, in $\Theta(n^2)$ time instead of $\Theta(n^3)$ (see [3]).

5 Applications of Solution Densities

Filtering. As reviewed in Section 2 several filtering algorithms were previously proposed for spanning tree constraints. But note that if we already spend time computing solution densities we can achieve domain consistency at no further expense: a solution density of 0 means that the edge can be filtered; a solution density of 1 means that the edge can be fixed. This holds as well for cost-damped solution densities since every solution is still taken into account. With cost-based solution densities we can apply cost-based filtering.
Sampling. The ability to compute solution densities exactly allows us to sample uniformly at random the combinatorial structures on which they are computed. In particular if we compute cost-based solution densities on spanning trees of weighted graphs, we can sample spanning trees of a given weight. We simply consider one edge at a time, deciding whether to include it or not according to its solution density (viewed as a probability), and then updating the solution densities accordingly.
Branching. The initial motivation for this work was to provide solution-counting information from constraints in order to branch using a heuristic relying on such information (i.e. counting-based search [12]). Among them, maxSD and its optimization counterpart, maxSD*, select the variable-value pair with the highest solution density among all constraints. In the next section we report on some experiments using the latter.

6 Experiments

To demonstrate the effectiveness of using solution density information from a spanning tree optimization constraint to guide a branching heuristic, we use it to solve some degree-constrained and diameter-constrained minimum spanning tree problems. Our graph instances are the same as [5], with a number of vertices ranging from 15 to 40 and an edge density ranging from 0.13 to 1 (i.e. a complete graph). We used the IBM ILOG CP v1.6 solver for our implementation and performed our experiments on a AMD Opteron 2.2GHz with 1GB of memory. We applied a 20-minute timeout throughout our experiments.

We introduce one binary variable $x_e \in \{0, 1\}$ per edge of the graph and a cost variable k corresponding to the weight of the spanning tree, which we minimize. On these we post optimization constraint `spanningTree`$(G, \{x_e : e \in E\}, k, w)$

equipped to answer queries about solution densities and enforcing

$$\sum_{e \in E} w(e) \cdot x_e = k$$
$$\sum_{e \in E} x_e = n - 1$$
$$\sum_{e=(v,v') \in E} x_e \geq 1 \qquad \forall v \in V$$
$$\sigma^\star(x_e, 1, \texttt{spanningTree}) = 1 \quad \Rightarrow \quad x_e = 1 \qquad \forall e \in E$$
$$\sigma^\star(x_e, 1, \texttt{spanningTree}) = 0 \quad \Rightarrow \quad x_e = 0 \qquad \forall e \in E$$

The last two constraints are only present if we use counting-based search and hence need solution densities. We omit parameter ϵ in $\sigma^\star$ because it is not relevant for cost-damped solution densities. We also compute a lower bound at each node of the search tree by running Kruskal's minimum spanning tree algorithm on G updated to reflect the edges that have been decided.

We compare counting-based branching heuristic maxSD* fed with cost-damped solution densities (using $x = 0.9$; the algorithm is not particularly sensitive to that choice) to a reasonable dedicated heuristic that selects edges by increasing weight.

6.1 Degree-Constrained Minimum Spanning Tree Problem

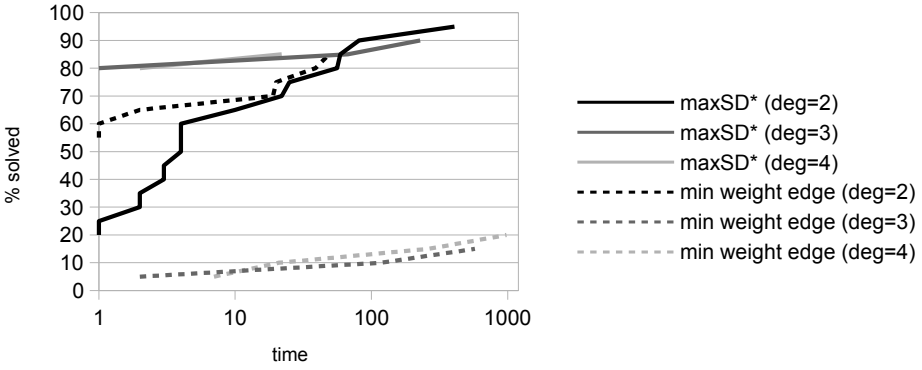


Fig. 3: Percentage of Degree-Constrained MST instances solved to optimality after a given time per instance.

The Degree-Constrained MST Problem requires that we find a minimum spanning tree of G whose vertices have degree at most d [9]. We add to our model

$$\sum_{e=(v,v') \in E} x_e \leq d \qquad \forall v \in V$$

We attempt to solve to optimality the previously-mentioned instances with $d = 2$, 3, and 4. Our results are summarized at Figure 3: the dedicated heuristic performs almost as well as maxSD* on the $d = 2$ instances (which corresponds to finding a minimum Hamiltonian path) but it is clearly outperformed on the larger degree bounds.

6.2 Diameter-Constrained Minimum Spanning Tree Problem

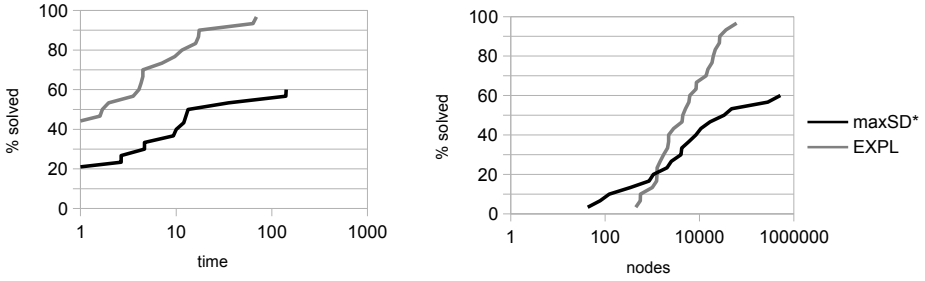


Fig. 4: Percentage of Diameter-Constrained MST instances solved to optimality after a given time (left) or size of search tree (right) per instance.

The Diameter-Constrained MST Problem requires that we find a minimum spanning tree of G such that for any two vertices the path joining them has at most p edges [1]. So in addition to the `spanningTree` constraint we maintain incrementally the length of a shortest path between each pair of vertices and backtrack if any exceed p. We attempt to solve to optimality the previously-mentioned instances with a diameter p ranging from 4 to 10. Our results are summarized at Figure 4. The heuristic selecting edges by increasing weight did not solve any instance so it does not appear in the figure. maxSD* manages to solve about half of the instances but is not competitive in terms of computation time and effectiveness with EXPL, the explanation-generating approach of [5], though on some instances it yields smaller search trees. Here the CP models are also quite different: the other one is tailored to the Diameter-Constrained MST with parent and height variables, and a dedicated branching heuristic.

7 Conclusion

We presented new algorithms to compute cost-aware solution densities for spanning tree constraints on weighted graphs and showed how they can be used in counting-based branching heuristics to improve our ability to solve constrained minimum spanning tree problems in CP. Such solution densities are also useful for domain filtering and uniform sampling.

Financial support for this research was provided by NSERC Grant 218028/2012.

References

1. NR Achuthan, L Caccetta, P Caccetta, and JF Geelen. Algorithms for the Minimum Weight Spanning Tree with Bounded Diameter Problem. *Optimization: Techniques and Applications*, 1(2):297–304, 1992.
2. Nicolas Beldiceanu, Pierre Flener, and Xavier Lorca. The Tree Constraint. In Roman Barták and Michela Milano, editors, *CPAIOR*, volume 3524 of *Lecture Notes in Computer Science*, pages 64–78. Springer, 2005.
3. Simon Brockbank, Gilles Pesant, and Louis-Martin Rousseau. Counting Spanning Trees to Guide Search in Constrained Spanning Tree Problems. In Christian Schulte, editor, *CP*, volume 8124 of *Lecture Notes in Computer Science*, pages 175–183. Springer, 2013.
4. Andrei Z. Broder and Ernst W. Mayr. Counting Minimum Weight Spanning Trees. *Journal of Algorithms*, 24(1):171 – 176, 1997.
5. Diego de Uña, Graeme Gange, Peter Schachte, and Peter J. Stuckey. Weighted spanning tree constraint with explanations. In Claude-Guy Quimper, editor, *CPAIOR*, volume 9676 of *Lecture Notes in Computer Science*, pages 98–107. Springer, 2016.
6. Grégoire Dooms and Irit Katriel. The *Minimum Spanning Tree* Constraint. In Frédéric Benhamou, editor, *CP*, volume 4204 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2006.
7. Grégoire Dooms and Irit Katriel. The "Not-Too-Heavy Spanning Tree" Constraint. In Pascal Van Hentenryck and Laurence A. Wolsey, editors, *CPAIOR*, volume 4510 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 2007.
8. Martin Hromčík and Michael Šebek. New Algorithm for Polynomial Matrix Determinant Based on FFT. In *Proceedings of the 5th European Control Conference (ECC99), Karlsruhe, Germany, September 1-3*, 1999.
9. Subhash C Narula and Cesar A Ho. Degree-Constrained Minimum Spanning Tree. *Computers & Operations Research*, 7(4):239–249, 1980.
10. Gilles Pesant. Achieving Domain Consistency and Counting Solutions for Dispersion Constraints. *INFORMS Journal on Computing*, 27(4):690–703, 2015.
11. Gilles Pesant. Counting-Based Search for Constraint Optimization Problems. In Dale Schuurmans and Michael P. Wellman, editors, *AAAI*, pages 3441–3448. AAAI Press, 2016.
12. Gilles Pesant, Claude-Guy Quimper, and Alessandro Zanarini. Counting-Based Search: Branching Heuristics for Constraint Satisfaction Problems. *J. Artif. Intell. Res. (JAIR)*, 43:173–210, 2012.
13. Jean-Charles Régin. Simpler and Incremental Consistency Checking and Arc Consistency Filtering Algorithms for the Weighted Spanning Tree Constraint. In Laurent Perron and Michael A. Trick, editors, *CPAIOR*, volume 5015 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2008.
14. Jean-Charles Régin, Louis-Martin Rousseau, Michel Rueher, and Willem Jan van Hoeve. The Weighted Spanning Tree Constraint Revisited. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *CPAIOR*, volume 6140 of *Lecture Notes in Computer Science*, pages 287–291. Springer, 2010.
15. W.T. Tutte. *Graph Theory*, volume 21 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 2001. 333 pages.
16. Alessandro Zanarini and Gilles Pesant. Solution Counting Algorithms for Constraint-Centered Search Heuristics. In Christian Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 743–757. Springer, 2007.