Using Cost-Based Solution Densities from TSP Relaxations to Solve Routing Problems

Pierre Coste, Andrea Lodi, and Gilles Pesant

Polytechnique Montréal, Canada
{pierre.coste,andrea.lodi,gilles.pesant}@polymtl.ca

Abstract. The Traveling Salesman Problem, at the heart of many routing applications, has a few well-known relaxations that have been very effective to compute lower bounds on the objective function or even to perform cost-based domain filtering in constraint programming models. We investigate other ways of using such relaxations based on computing the frequency of edges in near-optimal solutions to a relaxation. We report early empirical results on symmetric instances from TSPLIB.

1 Introduction

The Traveling Salesman Problem (TSP) is certainly one of the most well-studied combinatorial optimization problems. It is of theoretical interest as a prominent representative of the class of $\mathcal{NP}$-hard problems but also of great practical importance in routing and other application areas. Several relaxations of this problem have long been investigated as part of the efforts to solve it by computing lower bounds on the objective for search-tree pruning and more recently for cost-based domain filtering given an upper bound. This short paper examines whether recent work related to counting-based branching heuristics in Constraint Programming (CP) may offer new and effective ways of exploiting these relaxations.

Counting-based search [19] represents a family of branching heuristics in CP that guide the search for solutions by identifying likely variable-value assignments in each constraint. Originally introduced for satisfaction problems it was later extended to optimization problems [18]. Given a constraint $c(x_1, \ldots, x_k)$ on finite-domain variables $x_i \in \mathcal{D}_i$ $1 \leq i \leq k$, let $f : \mathcal{D}_1 \times \cdots \times \mathcal{D}_k \to \mathbb{N}$ associate a cost to each k-tuple t of values for the variables appearing in that constraint and z be a finite-domain cost 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 $z = f(x_1, x_2, \ldots, x_k)$. Let $\epsilon \geq 0$ be a small real number and $z^\star = \min_{t\,:\,c(t)} f(t)$ (without loss of generality consider that we are minimizing). We call

$$\sigma^\star(x_i, d, c^\star, \epsilon) = \frac{\sum_{t=(x_1,\ldots,x_{i-1},d,x_{i+1},\ldots,x_k)\,:\,c^\star(t,z,f)\wedge z\leq(1+\epsilon)z^\star}\ \omega(z, z^\star, \epsilon)}{\sum_{t=(x_1,\ldots,x_k)\,:\,c^\star(t,z,f)\wedge z\leq(1+\epsilon)z^\star}\ \omega(z, z^\star, \epsilon)}$$

the *cost-based solution density* of variable-value pair (x_i, d) in $c^\star$ given ϵ. Its value, between 0 and 1, measures how often assignment $x_i = d$ appears in

"good" satisfying assignments to $c^\star$. If $\epsilon = 0$ this corresponds to the solution density restricted to the optimal solutions to the constraint with respect to f. A positive ϵ gives a margin to include close-to-optimal solutions, but at a discount proportional to their distance from $z^\star$ (the minimum value of f over solutions to c), as given by generic weight function $\omega(z, z^\star, \epsilon) \in [0, 1]$ (for example, $\omega(z, z^\star, \epsilon) = 1 - \frac{z - z^\star}{\epsilon z^\star}$) whose definition may vary depending on the constraint. We use cost-based solution densities in CP to favour branching decisions, in the form of a variable assignment, that retain many good-quality solutions from the individual perspective of constraints, keeping in mind that each constraint in CP tends to represent a large combinatorial substructure of the problem.

In the rest of the paper, Section 2 reviews the TSP and in particular its common relaxations, Section 3 sketches the way cost-based solution densities are computed for the few constraints that are relevant here, Section 4 presents how cost-based solution densities can filter out unpromising edges as a preprocessing step or offer insightful branching heuristics, and Section 5 provides an empirical evaluation of these ideas on standard benchmark instances from the TSPLIB.

2 TSP

We are given a complete and undirected graph $G = (V, E)$, where V is called the vertex set and E is called the edge set. We are also given some costs associated with the edges of the graph, namely $t_{ij}, \forall (i, j) \in E$, and we assume that the triangle inequalities hold, i.e., $t_{ij} + t_{jk} \geq t_{ik}$ for every triplet of vertices $i, j, k \in V$. Then, the TSP calls for finding a unique tour visiting each vertex $i \in V$ exactly once. By associating a binary variable x_{ij} that takes value 1 if edge $(i, j) \in E$ belongs to the tour, the Integer Programming model with the most effective linear programming relaxation reads as follows

$$\min \sum_{(i,j) \in E} t_{ij} x_{ij} \tag{1}$$

$$\sum_{(i,j) \in \delta(i)} x_{ij} = 2, \quad i \in V \tag{2}$$

$$\sum_{(i,j) \in \delta(S)} x_{ij} \geq 2, \quad S \subseteq V, 2 \leq |S| \leq |V| - 2 \tag{3}$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in E \tag{4}$$

where $\delta(i)$ (resp. $\delta(S)$) denotes the set of edges incident to vertex i (resp. with one endpoint in set S). Degree constraints (2) establish that each vertex needs exactly two incident edges in a tour, while (3) forbid any subtour, i.e., a cycle of length smaller than V. As stated, the TSP is $\mathcal{NP}$-hard and a number of relaxations have been investigated. The strongest is obtained by optimizing (1) over the so-called *subtour elimination polytope*, that resulting from relaxing the integrality requirement (4) to nonnegativity. However, this relaxation is exponentially large and requires some care computationally (a sequence of

min-cut/max-flow problems has to be solved on carefully constructed graphs). Nowadays, linear-programming based algorithms have taken over the solution of extremely large-scale TSP instances and CONCORDE [1] is the state-of-the-art solver. However, exploiting the combinatorial structure of the TSP on more heterogeneous problems and incorporating its solution within modular programming paradigms like CP is still extremely relevant in practice and more combinatorial TSP relaxations that have been known for decades can be extremely useful. More precisely, in this paper we consider three of them.

1-tree relaxation. For a given vertex, say vertex 1, a 1-tree is a tree spanning the vertices in $V \setminus \{1\}$ plus two edges incident with vertex 1. It is easy to see that any 1-tree has at most one cycle and if the tree is computed by solving a minimum-cost spanning tree and the two edges are those of minimum cost, then the cost of the resulting 1-tree provides a lower bound on the optimal TSP cost. The 1-tree relaxation is computable in polynomial time by solving a minimum-cost spanning tree ($O(|V|^2)$ complexity) and has been used in one of the first breakthrough algorithm for the TSP by Held and Karp [15, 16].

2-matching relaxation. The 2-matching relaxation is obtained from model (1)–(4) by dropping constraints (3) (but keeping the integrality requirements (4)). The resulting integer programming problem is a perfect 2-matching, i.e., a collection of minimum-cost disjoint cycles covering all vertices and can be solved in polynomial time by the famous algorithm of Edmonds [13].

n-path relaxation. This relaxation was introduced by Christofides et al. [7] and generally used with a path representation of the problem where a vertex, say 1, is considered the starting one of the tour and duplicated (vertex $n + 1$), so as to transform a tour in a path of $n + 1$ edges from vertex 1 to itself, i.e., $n + 1$. The idea is to relax the degree constraint of each vertex and, at the same time, imposing that $|V| := n$ edges need to be selected in the resulting path. Through dynamic programming, such a shortest but not-necessarily-elementary path can be computed in polynomial time and this relaxation has been especially used in the column generation approaches for more complex routing problems like the Capacitated Vehicle Routing and its variant.

We end the section by noting that there have been several previous attempts in CP to filter edges by using relaxations. Two of the most relevant ones in this context are Benchimol et al. [3] and Ducomman et al. [12]. In [3], the authors consider the 1-tree relaxation and, in an additive way, the 2-matching one to empower the so-called `weightedCircuit` constraint so as to remove edges from the variables' domain through cost-based domain filtering [14]. The computational experiments on the TSP show strong size reductions and improved computing times with respect to less sophisticated CP models. In [12], the authors extend the work in [3] for the `weightedCircuit` constraint by showing that among the three bounds above, 1-tree, 2-matching and n-path, there is no dominance in terms of filtering and successfully apply the framework to one of the time-constrained TSP variant, namely TSP with time windows.

$\min z = \sum_{i=1}^{n} \gamma_{is_i}$ s.t.
`minWeightAlldifferent`$(\{s_1, \ldots, s_n\}, z, \Gamma)$
`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 1: A basic CP model for the TSP

3 Solution Densities of CP Optimization Constraints

The combinatorial structure of each relaxation presented in the previous section happens to be captured by some existing optimization constraint in CP. In this section we review them and outline the ways solution densities are computed for them. But first, consider the following basic CP model for the TSP as given at Table 1. Without loss of generality we start a tour at vertex 1 and end it at vertex $n+1$, which is a duplicate of 1. We define successor variables s_i so that $s_i = j$ corresponds to using edge (i, j). Γ represents the distance matrix. The `minWeightAlldifferent` optimization constraint [6] is an `alldifferent` constraint to which we add costs for each variable-value assignment and an extra variable representing the sum of the assignments. Here it enforces the assignment part for the successor variables s_i and links them to the objective variable z but does not apply any cost-based domain filtering. Constraint `noCycle` is the subtour elimination constraint.

2-matching relaxation. The collection of minimum-cost disjoint cycles for the former IP model of Section 2 corresponds to a minimum-cost assignment for the latter CP model, captured by the `minWeightAlldifferent` constraint. As described in [18], to derive cost-based solution densities from that constraint we first compute a minimum-weight bipartite matching, say of cost $z^\star$, using the Hungarian algorithm. As a by-product of this computation we get a reduced-cost matrix $R = (r_{ij})$ whose non-negative entries r_{ij} tell how much of an increase in cost we can expect if we assign value j to variable i instead of the value from the computed matching. Next we define related matrix $R' = (r'_{ij})$ as

$$r'_{ij} = \max(0, \frac{(\epsilon z^\star + 1) - r_{ij}}{\epsilon z^\star + 1}).$$

Each entry r'_{ij} lies in the real interval $[0, 1]$, with value 1 corresponding to a reduced cost r_{ij} of 0 and value 0 corresponding to any reduced cost signalling a variable-value assignment whose cost would exceed the ϵ margin.

The permanent of a $n \times n$ matrix $A = (a_{ij})$ is defined as

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

where P denotes the set of all permutations of $\{1, 2, \ldots, n\}$. In the case of a binary matrix representing the domain of each variable (variables on the rows, values on the columns, and a "1" entry if and only if the value appears in the

domain of the variable) this represents the number of solutions to the constraint: we sum over all possible assignments; the inner product is equal to 1 if each variable has the corresponding value in its domain and 0 otherwise. In the case of R' an optimal assignment counts as 1, any assignment whose cost exceeds $(1+\epsilon)z^\star$ or that is simply infeasible counts as 0, and any other assignment is counted at a discount proportional to how far it is from the optimum. We thus achieve a weighted counting of feasible assignments that are within our ϵ margin.

The cost-based solution density of a variable-value pair will be computed as the ratio of two permanents. Because computing the permanent is $\#P$-complete, several computationally-tractable upper bounds have been proposed: we use bound U^1 from [23].

1-tree relaxation. There has been considerable work on (weighted) tree structures in CP about domain filtering [2, 10, 11, 20, 21], failure explanation [8], and (cost-based) solution densities [4, 9]. We recall below the latter work that is relevant here. Note that since our goal is to compute edge frequencies in low-weight trees and not a lower bound on a tour, we consider spanning trees instead of 1-trees.

We define matrix $M = (m_{ij})$ whose elements are univariate polynomials built from edge set E and weight function w defined over E:

$$m_{ij} = \begin{cases} -x^{w(e)}, & i \neq j \quad e = (v_i, v_j) \in E \\ 0, & i \neq j \quad (v_i, v_j) \notin E \\ \sum_{e=(v_i,v_k)\in E} x^{w(e)}, & i = j \end{cases}$$

A remarkable result [5] states that any minor of M (i.e. the determinant of the submatrix obtained by removing from M row and column i for any i) yields a polynomial $\sum_k a_k x^k$ in which monomial $a_k x^k$ indicates the number a_k of spanning trees of weight k. Instead of computing the determinant of a matrix of polynomials, a potentially time-consuming process, we instantiate x to a real value between 0 and 1 (e.g. 0.7 in our experiments), yielding a matrix of scalars. Its effect is to apply an exponential decay 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, thus giving more importance to close-to-optimal trees. Setting $x = 1$ does not apply any decay and so all trees are counted equally — the closer x gets to 0, the more aggressive the decay.

To compute the cost-based solution density of an edge we exploit the fact that the matrix M of a graph without that edge is almost identical to the original one, leading to an efficient computation by matrix inversion, as described in [4]. To consider this relaxation we add a `minWeightSpanningTree` constraint to the CP model at Table 1.

n-path relaxation. We can represent an n-path with a `regular` constraint on variables $\langle x_1, x_2, \ldots, x_n \rangle$ where x_i corresponds to the vertex in ith position on the path and using the input graph as the automaton. This allows us to use `cost-regular`, its optimization variant, for which an algorithm computing cost-based solution densities has already been proposed [18]. The domain-filtering

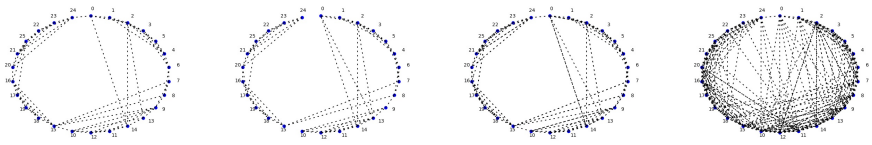

Fig. 1: From left to right: top 15% edges for each vertex according to the 2-matching and 1-tree relaxations, their union, and the 35% cheapest edges per vertex.

algorithm for `regular` builds and maintains a layered digraph built by unfolding the automaton over the sequence of variables. Each path from the first layer to the last in the layered digraph corresponds to a solution (here, an n-path). The cost-based solution density algorithm needs to restrict its attention to paths of cost at most $(1+\epsilon)z^\star$ where $z^\star$ is the cost of the shortest n-path. It therefore computes at each node of the digraph the number of incoming and outgoing partial paths of each cost up to $(1+\epsilon)z^\star$. The number of relevant paths featuring a given variable-value pair is computed as the sum over all corresponding arcs of the products of number of partial incoming/outgoing paths at their endpoints, provided their composition makes an n-path of cost at most $(1+\epsilon)z^\star$ (see [18] for details — note that contrary to that reference, here we weigh paths according to their cost, similarly to the other two relaxations). To consider this relaxation we add a `cost-regular` constraint to the CP model at Table 1.

4 Exploiting Cost-Based Solution Densities

4.1 Preprocessing

We investigate using cost-based solution densities computed from the relaxations as a preprocessing step that discards unpromising edges in an optimal tour. From each relaxation, we consider the k% highest solution densities for each vertex, with the choice of k depending on how aggressive we wish to be, and then combine that information by keeping the union of the corresponding edges. In this way we only discard edges that are considered unpromising *by all relaxations*.

As an illustration consider instance *fri26* from the TSPLIB. Each graph in Figure 1 shows the vertices in clockwise order of its optimal tour and the edges that are kept in each case. We observed that the n-path relaxation (not shown here) does not provide discriminating information unless some variables are fixed and so will be left out for preprocessing. Note that the graph from the 2-matching (and of course its union with the 1-tree) includes all edges of the optimal tour. To preserve all the optimal edges while simply keeping the cheapest edges from each vertex we would need to increase k to 35%, yielding a much denser graph (extreme right).

4.2 Branching Heuristic

Beyond preprocessing, cost-based solution densities provide insightful information for optimistic branching in a search tree. The question of how best to com-

	original graph				graph with discarded edges			
instance	# bbnodes	total time (s)	branching (s)	# cuts	# bbnodes	total time (s)	branching (s)	# cuts
ch150	1	0.49	0	111.5	2.4	0.35	0.21	88.2
kroA150	1	0.88	0	164.3	1	0.19	0	32.9
kroB150	1.2	0.84	0.01	171.3	1.2	0.42	0.02	58.8
si175	2.8	3.42	0.24	294.2	1	0.25	0	53.8
brg180	1	0.71	0	5.0	1	0.09	0	1.8
rat195	5.6	5.49	3.14	314.8	4.6	4.07	2.67	325.6
d198	3.2	2.29	0.31	193.0	1.6	1.12	0.28	95.1
kroA200	1	0.77	0	250.1	1.4	0.39	0.10	103.7
kroB200	1	0.44	0	136.6	1.2	0.21	0.02	33.1
ts225	1.5	8.01	0.23	875.0	1	0.44	0	197.9
pr226	1	0.51	0	101.9	1	0.24	0	42.1
pr264	1	0.44	0	49.7	1	0.28	0	17.9
a280	1	1.18	0	107.0	3	1.04	0.23	111.4
pr299	1.8	3.24	0.09	446.9	2.2	2.11	0.84	387.4
lin318	1	1.77	0	237.0	2.4	3.53	0.62	306.9
rd400	11.2	18.20	12.30	754.1	11.0	17.97	14.62	814.2

Table 2: Average performance (10 runs) of CONCORDE to solve to optimality symmetric instances of size 150 to 400 from the TSPLIB, with and without preprocessing.

bine such information from each relaxation arises again — one simple combination that has worked well in general, called $maxSD^\star$ [18], identifies the highest cost-based solution density over all constraints, variables, and values, makes the corresponding assignment in the left branch and forbids that same assignment in the right branch. Another combination that we will evaluate considers the arithmetic mean of the cost-based solution densities from each relaxation for a given variable-value pair and selects the highest one.

5 Empirical Evaluation

5.1 State-of-the-Art Exact Solver

As discussed in Section 4.1, one algorithmic idea for using the density of an edge in the solutions of the TSP relaxations is to sparsify the instance accordingly, i.e., discard edges of low density. To test the computational effect of this idea, we preprocess 16 classical TSP instances with number of nodes between 150 and 400 and we run CONCORDE both on the original instance and the sparsified one (keeping the 1% highest densities, which notably still happens to include the optimal solution for all 16 instances). Some significant characteristics of the two types of run are reported in Table 2, namely the number of branch-and-bound nodes, the overall computing time, the time for branching and the number of generated cuts. All numbers are averages over 10 runs.

Of course, we know that instances of that size are not challenging for CONCORDE. Indeed the largest tree size for the classical version of the instances is only 11.2 nodes and 9 out of the 16 instances do not require branching at all. Therefore the potential impact is limited. Nevertheless there is an overall improvement in the solving process whose most significant characteristic seems to be the reduction in the number of cutting planes CONCORDE needs to separate

	opt		best	time(s)	fails		opt		best	time (s)	fails		opt		best	time(s)	fails
gr21	2707	a	2707	0.11	173	gr24	1272	a	1272	0.26	359	fri26	937	a	937	0.43	475
		b	2707	0.04	48			b	1272	0.03	25			b	937	1.71	2163
		c	2707	0.14	27			c	1272	0.07	2			c	937	16.54	3156
		d	2707	34.90	60			d	1272	76.67	182			d	937	486.0	1367
		e	2707	0.15	31			e	1272	0.31	47			e	937	12.37	2393
		f	2707	35.23	70			f	1272	30.39	51			f	937	1684.26	3254
bays29	2020	a	2020	125.36	118869	dantzig42	699	a	699	0.3	0	swiss42	1273	a	1464	1537.4	1349413
		b	2020	7.57	9341			b	722	114.7	40888			b	1298	127.3	42503
		c	2020	8.37	1179			c	722	1147.1	73431			c	1397	24.9	1316
		e	2020	9.03	1287			e	718	72.8	3505			e	1410	25.3	1266
gr48	5046	a	5898	1097.1	423271	hk48	11461	a	14734	1486.8	970394	berlin52	7542	a	10434	1162.1	702850
		b	5055	548.6	195503			b	12466	591.6	277333			b	8224	104.6	123179
		c	5174	860.1	31897			c	12039	765.1	29543			c	8193	621.6	18781
		e	-	-	-			e	12032	1379.1	54948			e	8016	558.9	15548

Table 3: Performance of IBM ILOG CP 1.6 on 9 small sparsified instances (30 min timeout).

to reach optimality. A more detailed computational analysis on larger and more challenging instances is needed to confirm this trend and, hopefully, to observe a large impact, which in order to be useful needs to compensate the time spent in computing the edge densities, which currently takes several seconds.

5.2 CP

Next we consider both opportunities mentioned in Section 4 — namely, discarding some edges from the input graph and branching on the s_i variables using cost-based solution densities — and evaluate their impact on solving the CP model at Table 1 for small instances from TSPLIB. Such a model is clearly not competitive with the state of the art but serves our evaluation need.

Table 3 compares several branching heuristics: maximum regret [17] (a); $maxSD^\star$ on 2-matching (b), 2-matching & 1-tree (c), 2-matching & 1-tree & n-path (d); arithmetic mean of 2-matching & 1-tree (e) and of 2-matching & 1-tree & n-path (f). Instances were preprocessed using $k = 15\%$. We do not show explicitly the results on the original graphs but every heuristic performed much worse on them, thus indicating that discarding edges in this way is beneficial. We generally observe a marked improvement when using our proposed branching heuristics with respect to maximum regret. However those which involve the n-path relaxation (d,f) are too computationally expensive and are only reported on the three smallest instances.

6 Conclusion

We introduced new ways of exploiting known relaxations of the TSP and presented some preliminary empirical results to evaluate their usefulness. We believe this line of research requires further investigation and is particularly interesting for a CP approach to solve routing problems with additional constraints that make it difficult to apply other exact approaches directly.

References

1. *Concorde TSP Solver*. https://en.wikipedia.org/wiki/Concorde_TSP_Solver.
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. Pascal Benchimol, Willem Jan van Hoeve, Jean-Charles Régin, Louis-Martin Rousseau, and Michel Rueher. Improved filtering for weighted circuit constraints. *Constraints*, 17(3):205–233, 2012.
4. 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.
5. Andrei Z. Broder and Ernst W. Mayr. Counting Minimum Weight Spanning Trees. *Journal of Algorithms*, 24(1):171 – 176, 1997.
6. Yves Caseau and François Laburthe. Solving various weighted matching problems with constraints. In Gert Smolka, editor, *Proc. CP'97*, volume 1330 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 1997.
7. N. Christofides, A. Mingozzi, and P. Toth. State space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11:145–164, 1981.
8. 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.
9. Antoine Delaite and Gilles Pesant. Counting weighted spanning trees to solve constrained minimum spanning tree problems. In Domenico Salvagnin and Michele Lombardi, editors, *CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 176–184. Springer, 2017.
10. 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.
11. 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.
12. Sylvain Ducomman, Hadrien Cambazard, and Bernard Penz. Alternative filtering for the weighted circuit constraint: Comparing lower bounds for the TSP and solving TSPTW. In Schuurmans and Wellman [22], pages 3390–3396.
13. J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of research of the National Bureau of Standards*, 69B:125–130, 1965.
14. F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In J. Jaffar, editor, *Principle and Practice of Constraint Programming - CP99*, volume 1713 of *Lecture Notes in Computer Science*, pages 189–203. Springer-Verlag, 1999.
15. M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.
16. M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1:6–25, 1970.
17. Philip Kilby and Paul Shaw. Vehicle routing. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 801–836. Elsevier, 2006.

18. Gilles Pesant. Counting-Based Search for Constraint Optimization Problems. In Schuurmans and Wellman [22], pages 3441–3448.
19. Gilles Pesant, Claude-Guy Quimper, and Alessandro Zanarini. Counting-based search: Branching heuristics for constraint satisfaction problems. *J. Artif. Int. Res.*, 43(1):173–210, January 2012.
20. 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.
21. 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.
22. Dale Schuurmans and Michael P. Wellman, editors. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. AAAI Press, 2016.
23. G.W. Soules. New Permanental Upper Bounds for Nonnegative Matrices. *Linear and Multilinear Algebra*, 51(4):319–337, 2003.