Explaining WeightedCircuit Constraint Filtering

Raphaël Boudreault and Claude-Guy Quimper

Université Laval, Québec, QC, Canada
`raphael.boudreault.1@ulaval.ca`, `claude-guy.quimper@ift.ulaval.ca`

Abstract. We study the travelling salesman problem (TSP) in the context of constraint programming. We propose a new filtering rule for the Circuit constraint based on the cuts used in the state-of-the-art exact TSP solver *Concorde*. We also propose to explain the filtering of the WeightedCircuit constraint based on the 1-tree relaxation of Held and Karp.

Keywords: Travelling salesman · Constraint programming · Circuit · WeightedCircuit · nogood · Cutting planes · 1-tree

1 Introduction

The *travelling salesman problem* (TSP) is a classic NP-Hard problem in combinatorial optimization and operational research that fascinates researchers since its first mathematical formulation in 1930 [1]. It consists, given a number of cities and the distances between each of them, to find a circuit with minimum total distance, i.e. a minimal path visiting all cities and returning to its starting point (a Hamiltonian circuit). The TSP, including its variants, has several applications in a variety of areas, from logistics to DNA sequencing. In particular, it appears naturally in many transportation and industrial problems.

At the moment, the state-of-the-art exact TSP solver *Concorde* can obtain optimal solutions for instances of thousands of cities in a few seconds [1]. It uses a *branch-and-cut* approach based on a linear relaxation in addition to specific linear cuts for the TSP. Particular data structures allow an efficient cut generation. However, in practical situations, we usually need to add side constraints to the TSP, such as time windows, vehicle capacities, and traffic avoidance. In that case, *Concorde* can no longer be used and we need to use another technique.

A natural approach for this kind of problem with side constraints comes from *constraint programming* (CP). In this programming paradigm, we consider a domain for each variable and the relations between them are initially expressed in the form of constraints. These constraints can be linear inequalities, satisfaction formulae, or any other global constraints, such as AllDifferent$(x_1, \ldots, x_n)$ which specifies that the variables $x_1, \ldots, x_n$ must be pairwise distinct. For example, a TSP with time windows could be modelled with a constraint requiring that the variables representing the solution path are a minimal tour and another one constraining the time of the visits. Each constraint has its respective domain filtering algorithm that removes inconsistent values from the domains. That way, we avoid trying assignments that lead to failure and we reduce significantly the search tree.

In modern CP systems, the constraint CIRCUIT encodes the requirement to have a Hamiltonian cycle in an unweighted graph. Effective filtering algorithms associated with this constraint are already widely used [2, 11]. For the TSP, i.e. the case of a weighted graph, we rather need the WEIGHTEDCIRCUIT constraint. This constraint and its filtering algorithms [3, 5, 6] improved the power of constraint systems to solve the *pure* TSP, even if we are still well behind what *Concorde* offers. For example, on a randomly generated TSP instance of 300 nodes, Benchimol *et al.* [3] showed that their filtering algorithm reduced the solving time from 771 to 24 seconds, while *Concorde* needed only 2.32 seconds to find an optimal solution.

In the past few years, constraint programming solvers became significantly more efficient with the learning of constraints called *nogoods*. Each of these constraints corresponds essentially to the *explanation* of an infeasibility caused by a specific assignment of variables. For example, *Chuffed*, *OR-Tools* (Google) and *CP-Optimizer* (IBM) are constraint solvers that generate nogoods. In particular, the *lazy clause generation* is a useful technique that combines domains filtering and Boolean satisfiability to generate nogoods. While filtering domains, each change is recorded as a clause, and it is only when a failure is encountered in the search tree that a nogood is learnt. Then, they allow the solver to avoid making the same choices that lead to these inconsistencies. Francis and Stuckey [8] used this method to explain efficiently the filtering of the CIRCUIT constraint.

Knowing that the global optimization efficiency of *Concorde* comes from specific linear cuts that are gradually added to the base model, it would be natural to think that the constraints solvers should add nogoods to the model as well. Thus, we propose to involve *Concorde*'s specific cuts in the filtering of the CIRCUIT constraint and to explain the filtering of the WEIGHTEDCIRCUIT constraint in order to generate nogoods.

Take note that I, Raphaël, started my master's degree in Computer science only at the beginning of September, 2019. Therefore, I do not perfectly comprehend all the subtleties of constraint programming yet. Thus, in this article, I will present the ideas that I want to explore during the time of my research.

2 Background

2.1 CIRCUIT

Let $G = (V, E)$ be an unweighted graph with vertices set V and edges set E, $|V| = n$. Also, let $\overline{X} = (X_1, \ldots, X_n)$ be a vector of variables such that X_i represents the successor of node $i \in V = \{1, \ldots, n\}$. Recalling that the *domain* of a variable Z, denoted by $\mathrm{dom}(Z)$, is the set of possible assignments for Z, we have that $\mathrm{dom}(X_i)$ initially corresponds to the possible successors of node $i \in V$. Thus, the constraint $\textsc{Circuit}(\overline{X})$ is satisfied if and only if the edges represented by the values of $\overline{X}$ form a Hamiltonian cycle of the graph, i.e. a path going through all vertices exactly once and having the same starting and ending vertex.

2.2 *Concorde* and the TSP

Now, let $w\colon E \to \mathbb{R}$ be the edge weights of the graph such that $w(e)$ corresponds to the weight of edge $e \in E$. Let $\delta(i)$ represent all edges adjacent to the vertex $i \in V$, and more generally, $\delta(S)$ for $S \subset V$ represent all the edges (i, j) with $i \in S$ and $j \in V \setminus S$, what we refer to the edges *adjacent* to S. The TSP can be formulated as an integer linear program as follows, where x_e is a binary variable corresponding to whether edge $e \in E$ is included in the tour or not [1]:

$$\begin{aligned}
\min \sum_{e\in E} & w(e)x_e & & \\
\text{s.t.} \sum_{e\in\delta(i)} & x_e = 2 & \forall i \in V & \quad (1)\\
\sum_{e\in\delta(S)} & x_e \geq 2 & \forall S \subset V, |S| \geq 3 & \quad (2)\\
& x_e \in \{0,1\} & \forall e \in E & \quad (3)
\end{aligned}$$

The equations (1) are known as the *degree constraints* and require that each node have only two adjacent edges. The inequalities (2) are known as the *subtour elimination constraints* and ensure the connectivity of the tour. Finally, (3) specifies that it must be binary variables.

The *Concorde* solver [1] relaxes the problem and solves the TSP with a global optimization. At first, it considers the *assignment problem relaxation* that is obtained by relaxing the subtour (2) and integrality (3) constraints:

$$\begin{aligned}
\min \sum_{e\in E} & w(e)x_e & \\
\text{s.t.} \sum_{e\in\delta(i)} & x_e = 2 & \forall i \in V\\
& 0 \leq x_e \leq 1 & \forall e \in E
\end{aligned}$$

Then, it searches for flaws in the obtained relaxed solution and corrects them by adding specific linear constraints to the relaxation that are called *cuts*. This technique is called the *cutting-plane method*. At each iteration, *Concorde* uses a *branch-and-cut* scheme to resolve a new linear relaxation, find new cuts and improve its TSP solution. The procedure terminates when no further improvements can be made and that an optimal solution is found. Two types of cuts are particularly efficient : *subtour elimination cuts* and *comb inequalities*. The former consists to add specific subtour elimination constraints when the relaxed solution is not a connected or 2-connected tour. The latter is a more interesting concept that we describe in what follows.

A *comb* consists of $H \subset V$ (the *handle*) and $T_1, \ldots, T_s \subset V$ (the *teeth*) such that:

- $T_1, \ldots, T_s$ are pairwise disjoint;
- $H \cap T_i \neq \emptyset$ and $T_i \setminus H \neq \emptyset$ for all $i \in \{1, \ldots, s\}$;
- $s \geq 3$ and is odd.

The *comb inequality* is

$$\sum_{e \in \delta(H)} x_e + \sum_{i=1}^{s} \sum_{e \in \delta(T_i)} x_e \geq 3s + 1. \tag{4}$$

In other words, the number of edges adjacent to the handle plus the number of edges adjacent to the teeth must be at least $3s + 1$, where s is the number of teeth. The Venn diagram of a comb in the case of $s = 3$ is depicted in figure 1.

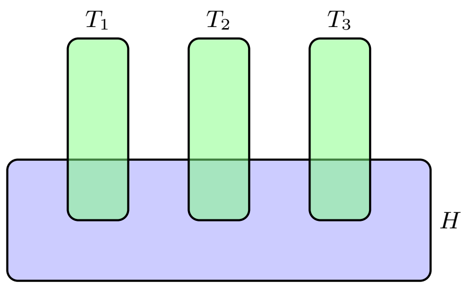

Fig. 1. Venn diagram of a comb with $s = 3$.

Let us give the intuition behind that inequality for this particular case. First, notice that for every nonempty proper subset $S \subset V$, there must be an even number of tour edges crossing the border of S. In addition, if S has exactly two adjacent tour edges, which is the minimal case, there must be a single path through S. Thus, if T_1, T_2 and T_3 consist minimally of single paths, the tour edges must cross the border of H at least three times. Hence, the left-hand side of the inequality (4) must be at least 9, even in fact at least 10, since it must be an even number.

In CP, the Circuit constraint filtering is essentially based on the subtour elimination cuts [2, 5, 6]. Indeed, the filtering is mostly done by forcing the 2-connectivity of the solution. As we can see, there is an interesting analogy to do between *Concorde* and CP for the Circuit constraint filtering.

2.3 WeightedCircuit

Keeping the notation of the last subsections, let now $G = (V, E, w)$ be a weighted graph with weight function w. We introduce the variable z corresponding to the total weight of the possible tour represented by $\overline{X}$, with $T \subset E$ the set of the edges forming this tour. Thus, the constraint $\textsc{WeightedCircuit}(\overline{X}, z)$ is defined by

$$\textsc{WeightedCircuit}(\overline{X}, z) = \textsc{Circuit}(\overline{X}) \wedge \left(\sum_{e \in T} w(e) \leq z \right)$$

i.e. will be satisfied if and only if the edges represented by the values of $\overline{X}$ form a Hamiltonian cycle on the nodes V with total weight at most z. In CP, the TSP can thus be reformulated simply as

$$\begin{aligned} \min \quad & z \\ \text{s. t.} \quad & \textsc{WeightedCircuit}(\overline{X}, z) \end{aligned}$$

A particularly efficient filtering technique for the constraint WeightedCircuit is called *reduced-cost-based filtering* [4–7]. It consists of first relaxing some constraints of the problem and finding an optimal solution to this relaxed problem. Say we obtain a solution of cost c^*. Then, starting from that solution, we calculate the *reduced cost* of a variable assignment, i.e. the marginal cost increase of the solution if a variable, say X_i, is forced to take a certain value, say v. Let the reduced cost of this assignment be $\bar{c}$. If

$$c^* + \bar{c} > \max(\text{dom}(z))$$

i.e. the cost of the relaxed solution plus the reduced cost of the variable assignment $X_i = v$ is greater than the maximum value the optimal cost can take, then we need to remove v from the domain of X_i. A recent use of this filtering method can be found in the work of Benchimol *et al.* [3].

2.4 1-tree relaxation

A relaxation that is widely used in both *Concorde* [1] and constraint solvers [3] is the *1-tree relaxation*, introduced by Held and Karp [9, 10], and is the result of relaxing the degree constraints (1). Recalling that the vertices set is $V = \{1, 2, \ldots, n\}$, a *1-tree* is defined by a spanning tree over the subgraph induced by the nodes $V \setminus \{1\}$ to which we add two distinct edges adjacent to the node 1. Take note that the choice of the vertex 1 is arbitrary. The 1-tree relaxation consists only to find a 1-tree with minimum total weight. We can see that it is a relaxation of the TSP by noting that every tour in a graph is a 1-tree and that if a minimum 1-tree is a tour, then it is an optimal solution of the TSP. A simple example of a minimum 1-tree is depicted in figure 2.

The lower bound obtained by the 1-tree relaxation can be improved by a Lagrangian relaxation of the degree constraints (1). Recalling that the degree of a node, noted $\deg(i)$ for $i \in V$, corresponds to the number of adjacent edges of that node, if the degree of a vertex in the 1-tree is greater than two, we add a penalty

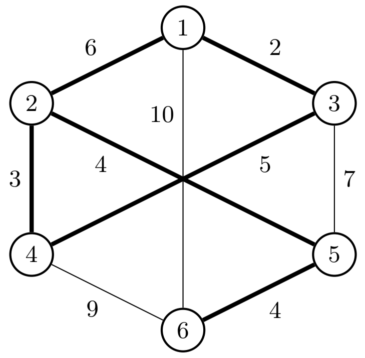

Fig. 2. Example of a minimum 1-tree that is not a tour, represented by bold edges. The number near each edge corresponds to its weight.

on the weight of all the adjacent edges of that vertex. Let $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$ be the penalty vector that is applied to the 1-tree (with $\pi_1 = 0$), we have that

$$\sum_{e \in E} w(e)x_e + \sum_{i=2}^{n} \pi_i(\deg(i) - 2)$$

is a lower bound for the TSP, for all penalty vectors. There exist subgradient optimization methods that allow to find a good penalty vector and reduces greatly the gap with the optimal solution. In summary, this technique leads to the following relaxation:

$$\begin{aligned}
\min_{x,\boldsymbol{\pi}} & \sum_{e \in E} w(e)x_e + \sum_{i=2}^{n} \pi_i(\deg(i) - 2) & & \\
\text{s.t.} & \sum_{e \in \delta(1)} x_e = 2 & & (5)\\
& \sum_{e \in E} x_e = |V| & & (6)\\
& \sum_{e \in \delta(S)} x_e \geq 2 & \forall S \subset V, |S| \geq 3 & \quad (7)\\
& x_e \in \{0, 1\} & \forall e \in E, &
\end{aligned}$$

where constraints (5), (6) and (7) define the 1-tree structure of the relaxed problem. The Held and Karp [9, 10] original approach for this problem was to first compute a minimum spanning tree on $G \setminus \{1\}$ and add the two edges with the lowest costs adjacent to the node 1 to form an initial minimum 1-tree. Then, in an iterative sequence, they perturb the one-tree by penalizing all the edges $(i, j) \in E$. The new weights $\tilde{w}(i, j)$ are defined by

$$\tilde{w}(i, j) = w(i, j) + \pi_i + \pi_j$$

where $\pi_i = C \cdot (\deg(i) - 2)$ for a constant C. This way, we obtain a new 1-tree from which we perturb again until a specific stopping criterion is met. Note that the optimal TSP tour is invariant under these transformations of the weights. Benchimol *et al.* [3] used the reduced costs of this relaxation to filter the WeightedCircuit constraint.

3 Ideas

3.1 Filtering and explanations for Circuit

In CP, we said before that adding subtour elimination cuts is the main filtering technique for the Circuit constraint. Besides, we also noted that *Concorde* uses a variety of linear cuts to efficiently solve the TSP without side constraints, including the comb inequalities. Thus, we propose to develop a filtering technique of the constraint involving the comb inequalities. For example, if we could find a comb with three teeth that has a maximum of 10 intersection edges, then these 10 edges would need to be forced. We would, therefore, propose a new filtering method and explanations would also need to be produced. This filtering method could then be generalized to other similar types of inequalities used by *Concorde* such as clique-tree inequalities, path inequalities and star inequalities [1].

3.2 Explanations for WeightedCircuit

Using the filtering method based on the 1-tree relaxation explained before, we would like to generate an explanation in order to eventually obtain a nogood. As for Circuit, explanations with fewer literals are usually more efficient in CP solvers. Thus, we would like to know how to generate the smallest possible explanation for the filtering of an edge with the 1-tree relaxation. Also, we would like to know if there exist some penalty vectors that would filter an edge, but would also minimize the size of this eventual explanation. Finding the answers of those questions would then allow us to include the constraint WeightedCircuit in a solver such as *Chuffed*.

4 Experimentation

During our research, we would like to implement our constraints in *Chuffed* solver and test them on classic instances of the TSP from TSPLIB library [12]. We would also like to test the improvement of the generation of explanations for the constraint WeightedCircuit on scheduling problems with setup times.

5 Conclusion

The travelling salesman problem is an NP-Hard problem that appears in many situations and under many faces. The state-of-the-art exact TSP solver *Concorde* offers the fastest way to obtain optimal solutions. However, it does not provide the possibility to add side constraints, such as time windows. Thus, constraint programming seems to be a natural way to get around this problem.

As we have seen, in CP, the constraint WeightedCircuit, an extension of Circuit, encodes the TSP. The filtering algorithms associated with this constraint do not provide, for the moment, explanations and so do not generate nogoods. We would like to improve the ability of CP solvers to solve the TSP by proposing a filtering algorithm that generates explanations based on the 1-tree relaxation of Held and Karp. Also, it seems that some interesting cuts used in *Concorde*, such as combs, could be used in the filtering algorithms of the Circuit constraint. In short, this is great research opportunities for years to come!

References

1. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Saleman Problem : A Computational Study. Princeton University Press (2006)
2. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. Mathematical and Computer Modelling **20**(12), 97–123 (1994)
3. Benchimol, P., Hoeve, W.J.V., Régin, J.C., Rousseau, L.M., Rueher, M.: Improved filtering for weighted circuit constraints. Constraints **17**(3), 205–233 (2012)
4. Focacci, F., Lodi, A., Milano, M.: Cost-based domain filtering. In: Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP). Lecture Notes in Computer Science, vol. 1713, pp. 189–203. Springer (1999)
5. Focacci, F., Lodi, A., Milano, M.: Embedding relaxations in global constraints for solving TSP and TSPTW. Annals of Mathematics and Artificial Intelligence **34**(4), 291–311 (2002)
6. Focacci, F., Lodi, A., Milano, M.: A hybrid exact algorithm for the TSPTW. INFORMS Journal on Computing **14**(4), 403–417 (2002)
7. Focacci, F., Lodi, A., Milano, M., Vigo, D.: Solving TSP through the integration of OR and CP techniques. Electronic Notes in Discrete Mathematics **1**, 13–25 (1999)
8. Francis, K., Stuckey, P.J.: Explaining circuit propagation. Constraints **19**(1), 1–29 (2014)
9. Held, M., Karp, R.M.: The traveling-salesman problem and minimum spanning trees. Operations Research **18**, 1138–1162 (1970)
10. Held, M., Karp, R.M.: The traveling-salesman problem and minimum spanning trees: Part II. Mathematical Programming **1**, 6–25 (1971)
11. Kaya, L.G., Hooker, J.: A filter for the circuit constraint. In: Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP). Lecture Notes in Computer Science, vol. 4204, pp. 706–710. Springer (2006)
12. Reinelt, G.: TSPLIB - A traveling salesman problem library. ORSA Journal on Computing **3**, 376–384 (1991), library available online at https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/