Improving a Branch-and-Bound Approach for the Degree-Constrained Minimum Spanning Tree Problem with LKH*

Maximilian Thiessen[1,2], Luis Quesada[3], and Kenneth N. Brown[3]

[1] Dept. of Computer Science
University of Bonn, Germany
thiessen@cs.uni-bonn.de
[2] Fraunhofer IAIS, Germany
[3] Insight Centre for Data Analytics, School of Computer Science & IT
University College Cork, Ireland
{luis.quesada,ken.brown}@insight-centre.org

Abstract. The degree-constrained minimum spanning tree problem, which involves finding a minimum spanning tree of a given graph with upper bounds on the vertex degrees, has found multiple applications in several domains. In this paper, we propose a novel CP approach to tackle this problem where we extend a recent branch-and-bound approach with an adaptation of the LKH local search heuristic to deal with trees instead of tours. Every time a solution is found, it is locally optimised by our new heuristic, thus yielding a tightened cut. Our experimental evaluation shows that this significantly speeds up the branch-and-bound search and hence closes the performance gap to the state-of-the-art bottom-up CP approach.

Keywords: Degree-Constrained Minimum Spanning Tree · Branch-and-Bound · Local Seach · LKH

1 Introduction

The degree constrained minimum spanning tree problem (DCMSTP) involves finding a minimum spanning tree (MST) of a given graph where the degree of every vertex is bounded. Minimum spanning trees are commonly used in the design of protocols for wireless sensor networks [30]. Having upper bounds on the degree of the vertices of a tree is a very common constraint in the design of such protocols due to many factors (e.g., bounded number of radios per vertex, limited capacity to store routing tables, etc) [14].

The data collection process (convergecasting) in wireless sensor networks is commonly accomplished by using a routing tree between the sensors [5, 1, 20],

* This work is supported by the Chist-ERA project Dyposit with funding from IRC, by Science Foundation Ireland under Grant No. 12/RC/2289 P2 and 16/SP/3804, by the EU and Enterprise Ireland under grant IR 2017 0041/ EU 737422-2 SCOTT, and by the German Academic Scholarship Foundation (Studienstiftung).

mostly due to time and energy efficiency reasons. The authors of [5] showed that the network topology is one of the main efficiency bottlenecks and significantly improved the practical performance by enforcing degree constraints on the vertices of the routing tree. Additionally, degree constraints are crucial in situations with battery-driven sensors in inaccessible terrains [20]. Similarly, these constraints can play an important role in the diversion of the flow to avoid interdicted links that result from cyber attacks in cyber-physical networks [25].

Another area of application is Software Defined Networks (SDNs). An SDN attempts to centralise network intelligence in one network component by disassociating the forwarding process of network packets (data plane) from the routing process (control plane). The control plane consists of one or more controllers which are considered as the brain of the SDN where the whole intelligence is incorporated [8]. The OpenFlow protocol is an open standard and is the main and most widespread enabling technology of the SDN architecture. An OpenFlow switch is equipped with a Forwarding Information Base (FIB) table, storing matching rules for the incoming packets, one or more actions (e.g., forward to a port, drop the packet, or modify its header) and counters. If an incoming packet matches a rule in the FIB, the corresponding action is taken and the counters are updated [26]. Reducing the energy impact of SDNs is an important challenge nowadays. Researchers have proposed protocols based on MST to address this challenge [26, 23]. The limited space for storing FIB tables seems to motivate the degree constraint on the vertices naturally.

DCMSTP subsumes the path version of the Traveling Salesman Problem (TSP), where one is interested in finding a Hamiltonian path of minimum cost. If we set the degree bound to 2 for each vertex, the Hamiltonian path problem reduces to the DCMSTP, and thus it is NP-hard [13]. While several CP approaches have managed to push the state-of-the-art of the TSP by primarily taking advantage of relaxations of the problem [10, 12, 2, 11, 17], we are only aware of one CP approach that has managed to do the same for DCMSTP [6].

A common feature in the applications we have mentioned is that the user is not necessarily interested in finding an optimal solution. In reality, the user is much more concerned about the time spent in the computation of the solution and is usually satisfied with a solution that is close to the optimal one. For instance, consider the case where a wireless network has to be restored after a link failure to a, possibly non-optimal, acceptable working state as fast as possible and only afterwards make further adjustments to save costs. This is certainly an issue with the recent bottom-up approach proposed in [6] since it only produces a satisfiable solution (the optimal one) at the end of the search process, besides the typically bad initial one. Branch-and-bound approaches do not have this drawback but suffer from poor performance due to the lack of good upper bounds.

In this paper, we propose a novel CP approach to tackle the DCMSTP where we extend a branch-and-bound approach with a local search heuristic inspired by LKH [16], which is the most widely used heuristic method for TSP, to combine the benefits of a branch-and-bound approach with an acceptable runtime. We

apply the heuristic to every found solution to improve its cut value, which not only skips a lot of intermediate solutions, but also strengthens the filtering of all used propagators. Our experiments show that our approach is competitive with respect to the state-of-the-art when it comes to CP [6] and is preferable when it comes to finding close-to-optimal solutions.

2 Background

Let $G = (V, E)$ be an undirected graph with integer edge costs $c : E \to \mathbb{Z}$ and a degree upper bound for each vertex, given by the mapping $d : V \to \mathbb{N}$. For all vertices $v \in V$ let $\delta(v)$ denote the set of incident edges of v. We want to find a spanning tree, satisfying the degree constraints for all vertices $v \in V$: $|\delta(v)| \leq d(v)$, of minimum total cost, which is the sum of all edge costs in the tree.

A typical CP formulation is:

$$\text{Minimize:} \quad Z \tag{1}$$

$$\text{s.t.} \quad \texttt{WST}(G, Z, c) \tag{2}$$

$$|\delta(v)| = D_v \leq d(v) \qquad \forall v \in V \tag{3}$$

It consists of a graph variable G, an integer variable Z representing an upper bound on the total cost, and integer degree variables D. A graph variable is an abstraction over the edge set of the graph, where the lower bound forms the set of already fixed (mandatory) edges and the upper bound is a superset of the lower bound. The difference between the upper bound and the lower bound is the set of optional edges [4]. The weighted spanning tree constraint $\texttt{WST}(G, Z, w)$ (2) forces G to contain a spanning tree of cost at most Z [28]. Additionally, we have the degree upper bound constraints (3). Due to the minimization Z will become the total cost of a spanning tree eventually.

The current state-of-the-art CP approach [6] combines (2) and (3) into the powerful $\texttt{DCWST}(G, Z, w, D)$ constraint. It is a direct generalization of the `WST` constraint and uses an adapted sub-gradient method of [15] to provide a lower bound on the total cost, typically much tighter than a normal minimum spanning tree would yield. On top of this model, the authors of [6] additionally adapted the Last-Conflict search strategy [18] to graph variables, which significantly improves the performance.

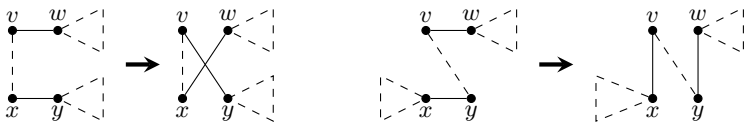
As a search procedure, they suggest to first greedily find any feasible solution, which is needed for the `DCWST` constraint, and continue with a simple bottom-up approach. They fix the total cost variable Z to a lower bound obtained via the pruning performed by the `DCWST` constraint and then branch on the graph variable. If a feasible solution is found it is guaranteed to be optimal. Otherwise, the lower bound is increased by one and the process is repeated.

Due to the lack of a good upper bounding heuristic on the cost, a basic branch-and-bound search is typically much slower than the bottom-up approach, as can be seen in our experiments.

Therefore, we suggest an improved branch-and-bound approach, which tackles this problem. Our ideas are inspired by the local search technique k-Opt [21] for the TSP and especially its popular generalization the Lin-Kernighan algorithm [22]. k-Opt has recently been applied to the DCMSTP [19], but using an incomplete local search approach, and without using LKH. At each step, they test all possible k-Opt swaps, including those which would change the vertex degree. In contrast, our approach is complete, and motivated by LKH, we check only a subset of possible moves.

The idea of k-Opt is to iteratively perform improving k-Opt swaps, which consist of swapping k edges of the current spanning tree (or tour in the TSP case) with k new ones to reduce the total cost of the graph until no more swaps can be found. As the original k-Opt swaps for tours, we consider only moves that do not change the vertex degrees. The case $k = 2$ is particularly simple because for any spanning tree there is only one correct way to reconnect two removed edges without violating the connectivity of the tree and changing the vertex degrees (see fig. 1):

Proposition 1. *Two non-incident edges $\{v, w\}$ and $\{x, y\}$ of a tree can be exchanged with the edges $\{v, y\}$ and $\{w, x\}$ without violating the tree property of the graph and changing the vertex degrees if and only if the (unique) connecting path between v and x is not using w or y.*



(a) x is closer than y to v (b) y is closer than x to v

Fig. 1: If we remove two edges $\{v, w\}$ and $\{x, y\}$ from a tree, either (a) $\{v, y\}, \{w, x\}$ or (b) $\{v, x\}, \{w, y\}$ can be used to reconnect the tree without changing the vertex degrees. Depending on whether x or y is closer to v in the tree without using w determines the valid swap. The dashed parts represent the rest of the tree.

Indeed the tree property is maintained as no cycle is introduced with the addition of edges $\{v, y\}$ and $\{w, x\}$ since these edges are connecting the disconnected components that were created by the removal of the edges $\{v, w\}$ and $\{x, y\}$. Similarly, the degrees are unmodified as the only vertices affected by the change are v, y, w and x, which trivially maintain their original degrees.

The quality of this local optimisation is strongly influenced by k. Typically, $k = 2$ is not enough and therefore $k \geq 3$ is used to produce satisfactory results. However, this is also the main drawback of the k-Opt approach, since to perform one single edge swap the algorithm has to check all possible edge sets of size k,

resulting in a runtime of $\mathcal{O}(|V|^k)$, which is too slow for typical real-world graphs and an appropriate k.

This problem motivates the Lin-Kernighan algorithm [22] for the TSP unifying k-Opt swaps by dynamically choosing k during runtime. The main idea is to start with 2-Opt swaps and only consider larger ones if needed. The currently available implementation of the Lin-Kernighan algorithm [16], called LKH, is one of the best-performing methods to heuristically solve the TSP in practice (see e.g. [29, 24] for recent developments).

One of the main difference to k-Opt is that LKH focuses on a certain subset of possible moves called *sequential* k-Opt swaps, being a k-Opt swap, which can be decomposed into $k-1$ many 2-Opt swaps, applied one after another each sharing an edge with the previous one. Additionally, the set of potential edge candidates is typically restricted to heuristically promising ones, for example only edges connecting nearest neighbours (see [16] for more details).

Motivated by the impressive results for the TSP, we propose to adapt LKH to spanning trees and use it on all found solutions of a branch-and-bound CP approach, in the sense of [9], to tighten the cuts, while still keeping the search complete.

3 Improving Solutions with LKH

As it is not possible to apply the state-of-the-art LKH implementation [16] to our problem due to its inherent design for tours, we decided to re-implement a bare-bones version of it using the mentioned sequential k-Opt swaps and the nearest-neighbour heuristic to use it directly in our CP setting. To further simplify the implementation we set an upper bound on the largest allowed k.

Since sequential k-Opt swaps can be decomposed into 2-Opt swaps (see [16]) we have to only find an efficient way to perform 2-Opt swaps. The main challenge here is to decide which one of the two possible swapping moves (see fig. 1) has to be performed to keep connectivity and the vertex degrees. Using proposition 1 the problem simplifies to checking the vertex distances in the tree. Given a first fixed edge $\{v, w\}$ of an edge pair candidate $\{v, w\}$ and $\{x, y\}$, the question is whether x or y is closer to v (see fig. 2).

To efficiently perform this check, we can use any graph traversal algorithm (like BFS or DFS) starting from v without visiting w. By proposition 1 every traversed edge $\{x, y\}$, where x is reached before y, can be swapped using $\{v, y\}$ and $\{w, x\}$. The same procedure for w is reversed, resulting in the opposite swap $\{v, x\}$ and $\{w, y\}$. In total, this yields a linear runtime to identify all valid swaps for a fixed edge.

As mentioned above, instead of trying all these edge pairs we only perform promising ones using the following nearest-neighbour heuristic. We first sort the neighbours of each vertex by increasing distance, which can be performed once at the beginning of the branch-and-bound approach. Then we only apply an edge swap, if one of the new edges, say $\{v, x\}$, connects two *close* vertices v and x, where close means being one of the nearest neighbours, typically restricted to

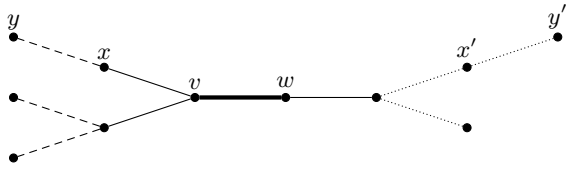


Fig. 2: Fixing the edge $\{v, w\}$ as the first part of an edge pair determines the correct way of swapping it together with any non-incident second edge. For all the (dashed) edges, which are reachable from v without using w, the correct way of swapping is to connect w with the vertex x reached first and v to the vertex y after it. For all the (dotted) edges reachable from w without using v, the opposite swap has to be performed using $\{w, y'\}$ and $\{v, x'\}$, connecting v to the vertex x' reached first.

3-5 neighbours. To prevent checking an edge pair twice, we can enforce $v < x$ with some fixed order on the vertices as a condition for any edge pair candidate. We want to emphasize that this local optimisation does not change the vertex degrees.

To sum up, our proposed approach is the following. Exactly as [6] we start with an initial greedy solution, but then apply a branch-and-bound search. Every time a solution is found, including the greedy one, it is locally optimised with LKH to reduce its total cost, hence yielding a better cut.

This significantly speeds up the branch-and-bound search, as can be seen in the experiments, since on the one hand, we can skip a lot of intermediate solutions and on the other hand, the improved cuts strengthen the filtering of the used propagators.

4 Experiments

In the following section, we compare our new approach with the state-of-the-art bottom-up CP approach from [6] and to a branch-and-bound search without our additional local optimisation on two benchmark datasets `DE` and `ANDINST` [3]. We implemented our approach in Java 8 on top of the CP library Choco 4.0.6 [27] with Choco-graph 4.2.4 [7] and made it publicly available[1]. All experiments are run on a Debian Linux 9 workstation with an Intel® Xeon® X5675 CPU and 128GB of total RAM. A time limit of 3 hours is used.

In our approach, we set the highest k for any k-Opt swap, as well as the number of nearest neighbours to 3. These parameters are a tradeoff between runtime and efficiency of the local optimisation. We determined these empirically, as they seemed to be the most appropriate for our test datasets. Due to space limitations, we omit a discussion about the parameter selection.

Motivated by our applications we also record the time to reach a solution of cost 1% close to the optimum, i.e. smaller than or equal to 1.01 times the optimal

[1] `https://github.com/mthiessen/CP-LKH-DCMST`

value. For the bottom-up approach, this almost always coincides with the overall runtime, because the bottom-up approach does not generate any intermediate solutions, besides the typically bad initial one.

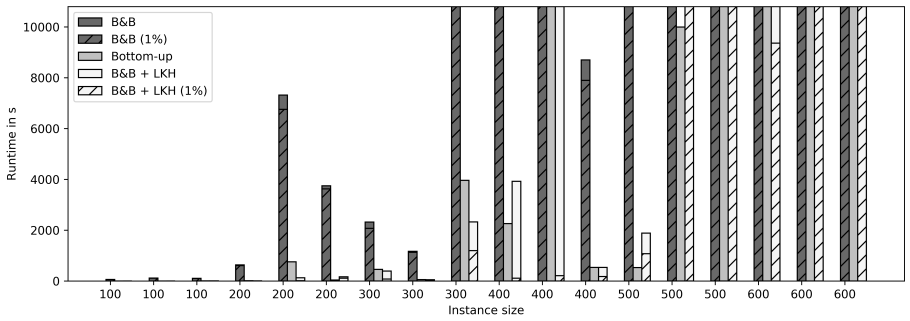


Fig. 3: Runtimes on `DE` instances

In fig. 3 we depict the runtimes on the `DE` instances. The striped parts of the branch-and-bound approaches indicate the time to reach a solution value 1% close to optimal. Our adapted approach is much faster than the basic branch-and-bound. We also observe that the time to reach 1% optimality almost coincides with the overall runtime for the basic branch-and-bound, while drops significantly for our adapted approach.

On most of the instances, the runtime of the bottom-up approach is comparable to our approach. In three instances the bottom-up approach is significantly better, but on the other hand on three other instances, our approach performs better. If we look at the runtime to reach 1% the situation improves. On all but two instances our approach finds such solutions faster than the bottom-up approach. This indicates that our approach is preferable over the bottom-up approach in a dynamic environment, such as in our mentioned applications.

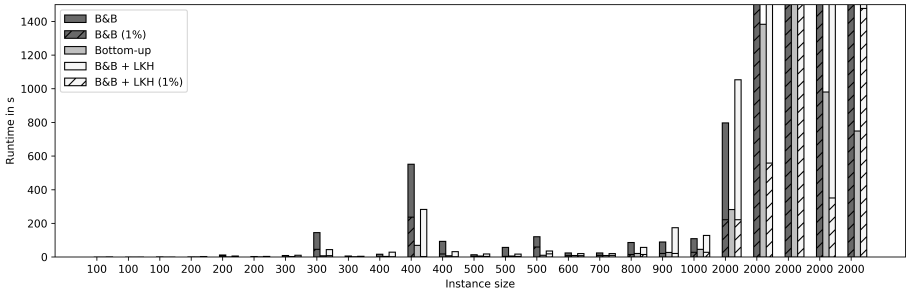


Fig. 4: Runtimes on `ANDINST` instances

On the ANDINST dataset (see fig. 4), our approach is better than the basic branch-and-bound on most instances. Unfortunately, our approach is slower than the basic branch-and-bound on some larger instances. We are convinced that a more sophisticated implementation of LKH on trees would resolve this issue.

If we compare our approach directly to the bottom-up approach, the latter is always faster than ours in finding an optimum solution. This is because the lower bound used in the bottom-up approach is very close to the optimum for the ANDINST dataset and so only a few bottom-up steps, typically 1-2, have to be performed.

Again the situation is a lot better if we compare the runtimes to reach 1%. Here our approach on all but the last instance beats the bottom-up, often with a large margin.

Overall the empirical results suggest preferring our adapted branch-and-bound approach over the basic one. The situation is much more diverse in the comparison of the bottom-up approach to ours, where most of the time the results are either similar or in favour of the bottom-up approach. Nevertheless, taking the time to reach 1% into account shows the benefits of using our adapted approach.

5 Conclusion

We have extended a CP based branch-and-bound approach to the degree constrained minimum spanning tree problem with an adaptation of the LKH local search heuristic. In this adaptation, the heuristic deals with trees instead of tours. The branch-and-bound search is significantly enhanced by this adaptation as shown in the experiments, which makes our approach competitive with the state-of-the-art bottom-up CP approach on some well-known benchmarks. We have discussed and shown empirically that our branch-and-bound approach is preferable in a dynamic environment, where close to optimal solutions are of interest.

This study is a proof-of-concept and indicates the high potential of an adapted branch-and-bound approach since there is a lot to improve. Especially, apart from improving the implementation of the adapted LKH, we plan to focus on Euclidean instances and build propagators especially for these cases. We also plan to generate nogoods and adapt our approach to deal with dynamic environments where edges are frequently changing. Lastly, generalising the idea of improving intermediate solutions by heuristics to related problems, such as the TSP, seems to be a promising research direction.

Acknowledgement. The authors thank Keld Helsgaun from the Roskilde University, Denmark for helpful discussions.

References

1. V. Annamalai, S. K. S. Gupta, and L. Schwiebert. On tree-based convergecasting in wireless sensor networks. In *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.*, volume 3, pages 1942–1947 vol.3, March 2003.
2. 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.
3. Alexandre Salles da Cunha and Abilio Lucena. Lower and upper bounds for the degree-constrained minimum spanning tree problem. *Networks: An International Journal*, 50(1):55–66, 2007.
4. Grégoire Dooms, Yves Deville, and Pierre Dupont. Cp (graph): Introducing a graph computation domain in constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 211–225. Springer, 2005.
5. O. Durmaz Incel, A. Ghosh, B. Krishnamachari, and K. Chintalapudi. Fast data collection in tree-based wireless sensor networks. *IEEE Transactions on Mobile Computing*, 11(1):86–99, Jan 2012.
6. Jean-Guillaume Fages, Xavier Lorca, and Louis-Martin Rousseau. The salesman and the tree: the importance of search in CP. *Constraints*, 21(2):145–162, 2016.
7. Jean-Guillaume Fages, Charles Prud'homme, and Xavier Lorca. *Choco Graph Documentation*. COSLING S.A.S., IMT Atlantique, 2018.
8. Hamid Farhadi, HyunYong Lee, and Akihiro Nakao. Software-defined networking: A survey. *Computer Networks*, 81:79–95, 2015.
9. Filippo Focacci, François Laburthe, and Andrea Lodi. Local search and constraint programming. In *Handbook of metaheuristics*, pages 369–403. Springer, 2003.
10. Filippo Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*, volume 1713 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 1999.
11. Filippo Focacci, Andrea Lodi, and Michela Milano. Embedding relaxations in global constraints for solving TSP and TSPTW. *Ann. Math. Artif. Intell.*, 34(4):291–311, 2002.
12. Filippo Focacci, Andrea Lodi, and Michela Milano. Optimization-oriented global constraints. *Constraints*, 7(3-4):351–365, 2002.
13. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
14. Luís Gouveia and Pedro Moura. Spanning trees with node degree dependent costs and knapsack reformulations. *Electronic Notes in Discrete Mathematics*, 36:985–992, 2010.
15. Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical programming*, 1(1):6–25, 1971.
16. Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.
17. Nicolas Isoart and Jean-Charles Régin. Integration of structural constraints into tsp models. In *International Conference on Principles and Practice of Constraint Programming*, pages 284–299. Springer, 2019.
18. Christophe Lecoutre, Lakhdar Saïs, Sébastien Tabary, and Vincent Vidal. Reasoning from last conflict (s) in constraint programming. *Artificial Intelligence*, 173(18):1592–1614, 2009.

19. Sang-Un Lee. A degree-constrained minimum spanning tree algorithm using k-opt. *Journal of the Korea Society of Computer and Information*, 20(5):31–39, 2015.
20. C. Liang, Y. Huang, and J. Lin. An energy efficient routing scheme in wireless sensor networks. In *22nd International Conference on Advanced Information Networking and Applications - Workshops (aina workshops 2008)*, pages 916–921, March 2008.
21. Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.
22. Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.
23. Haike Liu, Jilin Li, and Kai Yang. A dynamic method of constructing mst based on openflow protocol. *International Journal of Computer and Communication Engineering*, 5(6):398–408, 2016.
24. Paul McMenemy, Nadarajen Veerapen, Jason Adair, and Gabriela Ochoa. Rigorous performance analysis of state-of-the-art tsp heuristic solvers. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pages 99–114. Springer, 2019.
25. Lina Perelman and Saurabh Amin. A network interdiction model for analyzing the vulnerability of water distribution systems. In *Proceedings of the 3rd International Conference on High Confidence Networked Systems*, HiCoNS '14, pages 135–144, New York, NY, USA, 2014. ACM.
26. L. Prete, F. Farina, M. Campanella, and A. Biancini. Energy efficient minimum spanning tree in openflow networks. In *2012 European Workshop on Software Defined Networking*, pages 36–41, Oct 2012.
27. Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC - LS2N CNRS UMR 6241, COSLING S.A.S., 2017.
28. Jean-Charles Régin, Louis-Martin Rousseau, Michel Rueher, and Willem-Jan van Hoeve. The weighted spanning tree constraint revisited. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 287–291. Springer, 2010.
29. Renato Tinós, Keld Helsgaun, and Darrell Whitley. Efficient recombination in the lin-kernighan-helsgaun traveling salesman heuristic. In *International Conference on Parallel Problem Solving from Nature*, pages 95–107. Springer, 2018.
30. Risha Vashist and Suniti Dutt. Minimum spanning tree based improved routing protocol for heterogeneous wireless sensor network. *International Journal of Computer Applications*, 103:29–33, 2014.