Improving the Held and Karp Approach with Constraint Programming*

Pascal Benchimol[1], Jean-Charles Régin[2],
Louis-Martin Rousseau[1], Michel Rueher[2], and Willem-Jan van Hoeve[3]

[1] CIRRELT, École Polytechnique de Montréal
[2] Université de Nice - Sophia Antipolis / CNRS
[3] Tepper School of Business, Carnegie Mellon University

1 Introduction

Held and Karp have proposed, in the early 1970s, a relaxation for the Traveling Salesman Problem (TSP) as well as a branch-and-bound procedure that can solve small to modest-size instances to optimality [4, 5]. It has been shown that the Held-Karp relaxation produces very tight bounds in practice, and this relaxation is therefore applied in TSP solvers such as Concorde [1]. In this short paper we show that the Held-Karp approach can benefit from well-known techniques in Constraint Programming (CP) such as domain filtering and constraint propagation. Namely, we show that filtering algorithms developed for the weighted spanning tree constraint [3, 8] can be adapted to the context of the Held and Karp procedure. In addition to the adaptation of existing algorithms, we introduce a special-purpose filtering algorithm based on the underlying mechanisms used in Prim's algorithm [7]. Finally, we explored two different branching schemes to close the integrality gap. Our initial experimental results indicate that the addition of the CP techniques to the Held-Karp method can be very effective.

The paper is organized as follows: section 2 describes the Held-Karp approach while section 3 gives some insights on the Constraint Programming techniques and branching scheme used. In section 4 we demonstrate, through preliminary experiments, the impact of using CP in combination with Held and Karp based branch-and-bound on small to modest-size instances from the TSPlib.

2 The Held-Karp Approach

Let $G = (V, E)$ be a complete graph with vertex set $\{1, 2, \ldots, n\}$. We let c_{ij} denote the cost of edge $(i, j) \in E$. The cost function extends to any subset of edges by summing their costs. The Traveling Salesman Problem (TSP) asks for a closed tour in G, visiting each vertex exactly once, with minimum cost.

[4, 5] introduced the so-called *1-tree* as a relaxation for the TSP. A 1-tree is defined as a tree on the set of vertices $\{2, \ldots, n\}$, together with two distinct

* This work was partially supported by the European Community's 7th Framework Programme (FP7/2007-2013). It was started when L.-M. Rousseau and W.-J. van Hoeve were visiting the University of Nice-Sophia Antipolis (June/July 2009).

edges incident to vertex 1. The degree of a vertex is the set of edges in the 1-tree incident to that vertex, and we denote it by $\deg(i)$ for $i \in V$. To see that the 1-tree is a relaxation for the TSP, observe that every tour in the graph is a 1-tree, and if a minimum-weight 1-tree is a tour, it is an (optimal) solution to the TSP. Note that the 1-tree is a tour if and onlng y if all the degree of vertices is two.

The iterative approach proposed by [4, 5], uses Lagrangian relaxation to produce a sequence of connected graphs which increasingly resemble tours. We start by computing an initial minimum-weight 1-tree, by computing a minimum-spanning tree on $G \setminus \{1\}$, and adding the two edges with lowest cost incident to vertex 1. If the optimal 1-tree is a tour, we have found an optimal tour. Otherwise, the degree constraint one some of the vertices must be violated, i.e., it is not equal to two. In that case, we proceed by penalizing the degree of such vertices to be different from two by perturbing the edge costs of the graph, as follows. For each vertex $i \in V$, a 'node potential' π_i is introduced, Then, for each edge $(i, j) \in E$, the edge weight $\tilde{c}_{ij}$ is defined as $\tilde{c}_{ij} = c_{ij} + \pi_i + \pi_j$. [4] show that the optimal TSP tour is invariant under these changes, but the optimal 1-tree is not. Once choice for the node potentials is to define $\pi_i = (2 - \deg(i)) \cdot C$, for a fixed constant C. The Held-Karp procedure re-iterates by solving the 1-tree problem and perturbing the edge costs until it reaches a fixed point or meets a stopping criterion. The best lower bound, i.e., the maximum among all choices of the node potentials, is known as the Held-Karp bound and will be denoted by HK.

The overall Held-Karp approach solves the TSP through branch-and-bound, a technique that has been widely used on this problem (see [2] for a survey). A good upper bound, UB, can be computed easily with any of the popular heuristics that have been devised for this problem, e.g., [6].

3 Improving the Approach Using CP

In this section we describe the different refinements introduced to the original Held-Karp approach [4, 5], which consist of two filtering procedures based on the weighted minimum spanning tree (or 1-tree), and one based on the underlying structure of Prim's algorithm.

In the following procedures let T be a minimum 1-tree of G computed by the Held and Karp relaxation described above. For a subset of edges $S \subseteq E$, we let $w(S)$ denote $\sum_{e \in S} c_e$ and $T(e)$ be the minimum 1-tree where e is forced into T.

We note that the filtering in subsection 3.1 has been applied to the weighted minimum spanning tree constraint in [3, 8], and the filtering in subsection 3.2 has been applied to the weighted minimum spanning tree constraint in [3].

3.1 Removing Edges Based on Marginal Costs

The *marginal cost* of an edge e in T is defined as $c'_e = w(T(e)) - w(T)$, that is, the marginal increase of the weight of the minimum 1-tree if e is forced in the 1-tree.

The following algorithm can compute, in $O(mn)$, the marginal costs for edges $e \notin T$. Each non-tree edge $e = (i, j)$ links two nodes i, j, and defines a unique i-j path, say P^e, in T. The replacement cost of (i, j) is defined by $c_e - max(c_a | a \in P^e)$, that is the cost of (i, j) minus the cost of largest edge on the path from i to j in the 1-tree T. Finding P^e can be achieved through DFS in $O(n)$ for all the $O(m)$ edges not in T. If $\text{HK} + c'_e > \text{UB}$, then e can be safely removed from E.

3.2 Forcing Edges Based on Replacement Costs

Conversely, it is possible to compute the *replacement cost* of an edge $e \in T$ as the increase the Held-Karp bound would incur if e would be removed from E, which we define by $c^r_e = w(T \setminus e) - w(T)$.

This computation can be performed for all edges $e \in T$, with the following algorithm: a) set all $c^r_e = \infty \ \forall e \in T$ b) for all $e = (i, j) \notin T$ identify the i-j path P^e in T which joins the end-points of e. Update all edges $a \in P^e$ such that $c^r_a = min(c^r_a, c_e - c_a)$. This computation can be performed in $O(mn)$, or, at no extra cost if performed together with the computation of marginal costs. If $\text{HK} + c^r_e - c_e > \text{UB}$, then e is a mandatory edge in T.

We note that such filtering has been applied to the weighted minimum spanning tree constraint by [3, 8].

3.3 Forcing Edges Based during MST Computation

Recall that Prim's algorithm computes the minimum spanning tree in G (which is easily transformed into a 1-tree) in the following manner. Starting from any node i, it first partitions the graph into disjoints subsets $S = \{i\}$ and $\bar{S} = V \setminus i$ and creates an empty tree T. Then it iteratively adds to T the minimum edge $(i, j) \in (S, \bar{S})$, defined as the set of edges where $i \in S$ and $j \in \bar{S}$, and moves j from $\bar{S}$ to S.

Since we are using MST computations as part of a Held-Karp relaxation to the TSP, we know that there should be at least 2 edges in each possible $(S, \bar{S})$ of V (this property defines one of well known subtour elimination constraints of the TSP). Therefore, whenever we encounter a set $(S, \bar{S})$ that contains only two edges during the computation of the MST with Prim's algorithm, we can force these edges to be mandatory in T.

3.4 Tuning the Propagation Level

The proposed filtering procedures are quite expensive computationally, therefore it is interesting to investigate the amount of propagation that we wish to impose during the search. A first implementation consists in calling each filtering algorithm (as defined in sections 3.1, 3.2 and 3.3) only once before choosing a new branching variable. A second approach would be to repeat these rounds of propagation until none of these procedures is able to delete nor force any edge, that is reaching a fixed point. Finally, if reaching a fixed point allows to reduce the overall search effort, a more efficient propagation mechanism could be developed in order to speed up its computation.

3.5 Choosing the Branching Criterion

Once the initial Held-Karp bound has been computed and the filtering has been performed it is necessary to apply a branching procedure in order to identify the optimal TSP solution. We have investigated two orthogonal branching schemes, both based on the 1-tree associated to the best Held-Karp bound, say T. These strategies consist in selecting, at each branch-and-bound node, one edge e and splitting the search in two subproblems, one where e is forced in the solution and one where it is forbidden. In the strategy *out* we pick $e \in T$ and first branch on the subproblem where it is forbidden while in the strategy *in* we choose $e \notin T$ and first try to force it in the solution.

Since there are $O(n)$ edges in T and $O(n^2)$ edges not in T, the first strategy will tend to create search trees which are narrower but also deeper than the second one. However, since the quality of the HK improves rapidly as we go down the search tree, it is generally possible to cut uninteresting branches before we get to deep. Preliminary experiments, not reported here, have confirmed that strategy *out* is generally more effective than strategy *in*.

Table 1. Results on TSPlib instances

	original HK		1-round		fixpoint	
	time	BnB	time	BnB	time	BnB
burma14	0.1	28	0	0	0	0
ulysses16	0.16	32	0	0	0	0
gr17	0.14	34	0	0	0.01	0
gr21	0.16	42	0	0	0.01	0
ulysses22	0.19	0	0	0	0.01	0
gr24	0.23	44	0.01	0	0.03	0
fri26	0.36	48	0.01	2	0.01	2
bayg29	0.35	54	0.04	6	0.07	6
bays29	0.33	88	0.05	10	0.1	10
dantzig42	0.65	92	0.09	4	0.17	4
swiss42	0.79	112	0.09	8	0.09	8
att48	1.7	140	0.21	18	0.23	15
gr48	94	13554	5.18	2481	7.38	3661
hk48	1.37	94	0.17	4	0.16	4
eil51	15.9	2440	0.39	131	0.84	426
berlin52	0.63	80	0.02	0	0.02	0
brazil58	13	878	1.09	319	1.02	296
st70	236	13418	1.21	183	1.1	152
eil76	15	596	1.03	125	0.88	99
rat99	134	2510	5.44	592	4.88	502
kroD100	16500	206416	11	7236	50.83	4842
rd100	67	782	0.76	0	0.73	0
eil101	187	3692	8.17	1039	9.59	1236
lin105	31	204	1.81	4	1.85	4
pr107	41	442	4.65	45	4.49	48

4 Experimental Results

To evaluate the benefits of using CP within the Held-Karp branch-and-bound algorithm, we ran experiments on several instances of the TSPlib. We report both the number of branching nodes and CPU time required solve each instance, with different propagation levels: no propagation ('original HK'), calling each filtering algorithm once ('1-round'), and propagation until we reach a fixed point ('fixpoint'). To eliminate the impact of the upper bound can have on search tree, we ran these experiments using the optimal value of each instance as its UB.

Table 1 clearly shows the impact of CP filtering techniques on the original Held-Karp algorithm. In fact the reduction of the graph not only considerably reduces the search effort (BnB nodes) but also sufficiently accelerates the computation of 1-trees inside the Held-Karp relaxation to completely absorb the extra computations required by the filtering mechanisms. This can be seen as the proportional reduction in CPU times largely exceeds the reduction in search nodes.

Finally, we cannot conclude that the extra effort required to reach the fixed point is worthwhile, as it is sometimes better and sometimes worse than a single round of filtering. Results on these preliminary tests seem to show that more than one round of computation is most often useless, as the first round of filtering was sufficient to reach the fixed point in about 99.5% of the search nodes. More tests are thus required before investigating more sophisticated propagation mechanisms.

References

[1] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton (2006)

[2] Balas, E., Toth, P.: Branch and Bound Methods. In: Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.) The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, ch. 10. Wiley, Chichester (1985)

[3] Dooms, G., Katriel, I.: The "not-too-heavy spanning tree" constraint. In: Van Hentenryck, P., Wolsey, L.A. (eds.) CPAIOR 2007. LNCS, vol. 4510, pp. 59–70. Springer, Heidelberg (2007)

[4] Held, M., Karp, R.M.: The Traveling-Salesman Problem and Minimum Spanning Trees. Operations Research 18, 1138–1162 (1970)

[5] Held, M., Karp, R.M.: The Traveling-Salesman Problem and Minimum Spanning Trees: Part II. Mathematical Programming 1, 6–25 (1971)

[6] Helsgaun, K.: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. European Journal of Operational Research 126(1), 106–130 (2000)

[7] Prim, R.C.: Shortest connection networks and some generalizations. Bell System Tech. J. 36, 1389–1401 (1957)

[8] Régin, J.-C.: Simpler and Incremental Consistency Checking and Arc Consistency Filtering Algorithms for the Weighted Spanning Tree Constraint. In: Perron, L., Trick, M.A. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 233–247. Springer, Heidelberg (2008)