Tour merging via tree decomposition

A hybrid approach between heuristics and exact solutions for TSP and VRP.

Mattias Beimers - 3672565

May 26, 2015

Abstract

A hybrid approach between heuristics and exact solutions for TSP and VRP using tree decompositions.

1 Introduction

For many optimization problems calculating optimal solutions is not feasible in practical applications, because the computation time grows exponentionally with the problem size. Hence, heuristics are used to find solutions that are good, but not nescessarily optimal. To get more certainty that a solution is good, or to improve the solution even more, the heuristics are applied multiple times and the best solution is selected. Although this works well, Cook and Seymour noted in their work on the Traveling Salesman Problem (TSP) [1] that by discarding the other solutions, possibly valuable information is lost. Hence the idea emerged to merge the found approximate solutions and calculate the optimal solution on this reduced problem, using branch decompositions.

At the time of publishing the solutions found by Cook and Seymour improved on the best known results for instances with almost 25000 vertices [?]. Since then the heuristics have improved massively and outperform the approach by Cook and Seymour [3, ?]. In this paper will try if the strategy for TSP by Cook and Seymour can still improve current heuristics even more. Furthermore, we will try to extend it to work for the Vehicle Routing Problem (VRP).

The Traveling Salesman Problem (TSP) is one of the most well studied NP-hard problems, where a merchant wants to visit a number of cities and get back at his starting point in the shortest possible amount of time. We recognize the TSP problem in many practical applications, from planning a school bus route to scheduling a machine to drill holes in a circuit board. A generalized version of this problem, where there are not one but a number of merchants or trucks visiting the cities from the starting point (or depot), is whidely used in the transportation sector. This problem is known as the Vehicle Routing Problem (VRP).

We define the TSP, given a complete graph $G = (V, E)$, as finding the tour (or Hamiltonian circuit) with smallest total cost. For this paper we assume the cost c_e of an edge $e = (v, w)$ is the euclidean distance between v and w. Given additionally a demand d_v for each vertex v, a maximal capacity C of goods per truck, a number M of available trucks and a special vertex v_0 that is the depot, we can define the VRP as finding a set of at most M tours with the least total cost. Each tour has to start and end at the depot and has to have a total demand of at most C. There are many variants with additional constraints or freedoms, but these are out of the scope of this paper.

The paper is organised as follows: in section 2 we will discuss the heuristics used to generate the initial tours and routes. In section 3 we will discuss how the solutions are merged and how the treedecomposition is calculated and in section 4 we will show the dynamic programming algorithms on the computed decompositions. In section 5 we will discuss the result and finally we will conclude in section 6.

2 Heuristics

Although many different heuristics have been tried to solve the Traveling Salesman Problem, there are few that can compete with (variants of) the Lin Kernighan heuristic [2]. We will discuss the Lin Kernighan Helsgaun [3] variant in section 2.1. For the Vehicle Routing Problem the currently best heuristics are tabu search algorithms. Unfortunately they often require to finetune a lot of parameters and are focussed on specific instances of VRP, rather than giving consistent solutions for all versions [**?**, **?**]. Other heuristics like the classic savings heuristic or the sweep heuristic do give good solutions for all variants of VRP, but they can't get the results one gets with the tabu heuristics.

2.1 Lin Kernighan Helsgaun

Todo

3 Tree decomposition

Once we have generated a couple of good tours for our original graph $G' = (V, E')$, we merge these tours in one graph. All tours $E'_j \subset E'$, for $1 \leq j \leq \#tours$ as found by the heuristics are merged together into a new graph $G = (V, E)$, where $E = \bigcup E'_j)$. For this graph G we will compute a tree decomposition so that we can compute the optimal tour in this reduced problem.

3.1 Tree decomposition and width

A *tree decomposition* of a graph $G = (V, E)$ is a tree $T = (W, F)$ and a *bag*, or a set of vertices, $X_i \subset V$ for $i \in W$ such that:

1. $\bigcup_{i \in W} X_i = V$,
2. for all $(u, v) \in E$ there is an $i \in W$ with $u, v \in X_i$ and
3. for all $v \in V$, the set $W_v = \{i \in W : v \in X_i\}$ forms a connected subtree of T.

The *width* k of the tree decomposition is $\max_{i \in W} |X_i| - 1$. The *treewidth* of a graph G, is the minimum width among all tree decompositions of G.

In this paper we often work with the edge set corresponding to the bag X_i, $i \in W$, rather than the vertex set X_i itself, and to that end we define $Y_i = \{(u, v) \in E : u, v \in X_i\}$. We say that a bag contains a vertex v if $v \in X_i$ and that it contains an edge e if $e \in Y_i$.

3.2 Minimum Degree Heuristic

Calculating the optimal treewidth or the optimal tree decompositions is an NP Hard problem [**?**] so finding an optimal decomposition in reasonable time is infeasable unlesss $P = NP$. We also do not nescessarily need a tree decomposition of optimal width, we just need the width to be sufficiently small so that our DP algorithm runs fast enough. Therefore, we compute our tree decomposition with a heuristic: the minimum degree heuristic by Bodlaender and Koster [4].

Todo

4 Dynamic programming

Provided the treewidth is small enough, the optimal solution on the merged graph and its tree decomposition can be computed using dynamic programming algorithms. In the following sections we explain their details.

4.1 Traveling Salesman

Let $G = (V, E)$ be a simple graph with edge-weights c_e and $T = (W, F)$ be the tree decomposition with width $k - 1$ and X_i and Y_i as defined in section 3.1. Note that because G is the result of a number of merged tours, it is 2-connected. The main idea of the algorithm is to find a series of disjoint paths and connect them together into a hamiltonian tour of minimum weight. Such a path is encoded using vertex degrees and a matching. Every vertex can have degree 0, 1 or 2. Vertices with degree 2 are already *used* in a path, vertices with degree 1 are *endpoints* of a path and vertices with degree 0 are *free*, so not yet used in a path. For every pair of endpoints we have a *matching* $\{u, v\} : u, v \in V$ to mark which vertices mark the endpoints of one path.

We now define the function $F(X_i, \mathit{degrees}, \mathit{matching})$ to be the minimum total cost of the edges in a series of paths starting and ending in bag X_i. The endpoints of these paths should be as in the matching, all vertices in bags below X_i in the tree should be used (have a degree of 2) and the vertices in bag X_i should have degrees as given in the *degrees* parameter. A vertex' degree can be satisfied in the bag itself by choosing an edge from Y_i or by passing it on to one (or two) of the child bags. Formally this becomes

$$F(X_i, D, M) = \min_{1 \leq p \leq P_i} \Big(\sum_{j \in W:\ \mathrm{Parent}(j) = i} F(X_j, D_{p,j}, M_{p,j}) + \sum_{e \in E_p} c_e \Big)$$

for all P_i ways of dividing D and M into the $D_{p,j}$ and $M_{p,j}$ sets and the corresponding $E_p \subset Y_i$.

To find the different divisions we mainly focus on D, the set of degrees we have to 'spend'. We try all possible combinations of dividing the degrees of per vertex. For a given vertex v, assuming we have 2 degrees to spend, we first try to give it to each of the children. So for possibility p and child bag j we try to set the degree of v in $D_{p,j}$ to 2. Afterwards, assuming we have at least 1 degree to spend for v, we try to use it as an endpoint for each of the vertices in all of the child bags. So the degrees of two vertices, v and some other vertex u in $D_{p,j}$ are set to 1. We try this for all (valid) combinations of u and j. At this step we also add the matching $\{u, v\}$ to $M_{p,j}$. Finally we also try not to assign v to any of the child bags. Of course, vertices are only given to a child bag if it contains the vertex.

For the remaining degrees that are not handled by any of the child bags we calculate a subset E_p of Y_i. For non leaf bags these edges mainly glue paths from the children together in the paths as given in the D and M parameters. For the leaf bags there are of course no child bags to delegate the degrees to so it all has to be solved using the bags own edges. Note that the edge set is not allowed to introduce cycles, so in particular two endpoints in a matching are not allowed to be connected. The root bag is of course an exception, because there all paths are merged in a single cycle. If no valid edge set is found, $F(X_i, D, M) = \infty$.

The overall algorithm then consists of a top down approach where we tabulate all entries for F, starting at the root and then recursively work downwards in the tree. Then the value of each table entry is finished bottom up as the recursion returns the values for children.

4.2 Vehicle Routing

Todo

4.3 Speed

Although DP running time upperbounds of $O(n3^k2^{k^2}) - TODO???$ and $O(Mn3^k2^{\cdots})$ are terrible, in practice these limits are never reached. This is because edges. . . TODO

5 Results

Todo

6 Conclusion

Todo

References

[1] Cook, W., & Seymour, P. (2003). *Tour merging via branch-decomposition.* INFORMS Journal on Computing, 15(3), 233-248.

[2] Lin, S., & Kernighan, B. W. (1973). *An effective heuristic algorithm for the traveling-salesman problem.* Operations research, 21(2), 498-516.

[3] Helsgaun, K. (2009). *General k-opt submoves for the LinKernighan TSP heuristic.* Mathematical Programming Computation, 1(2-3), 119-163.

[4] Bodlaender, H.I. & Koster, A.M.C.A (2010). *Tree width computations I upper bounds.* Information and Compuutation 208, 259-275