Tour merging via tree decomposition

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

Mattias Beimers - 3672565

June 9, 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 often 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 the merged graph, 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 other heuristics have improved massively and outperform the approach by Cook and Seymour [3, ?]. In this thesis, we 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 widely 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 a tour, or cycle, that visits all cities exactly once with smallest total cost. For this thesis 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 maximum 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 can satisfy a total demand of at most C. Each vertex has to be visited by a tour exactly once. There are many other variants of the VRP with additional constraints or freedoms, but these are out of the scope of this thesis.

To solve the TSP and VRP we apply the following stragegy: We start by calculating an initial set of solutions using heuristics, this gives us a set of promising edges $E \subset E'$. After that we merge the solutions into a subgraph $G = (V, E)$. On this graph we (hopefully) find a tree decomposition with small width k. With that decomposition we can calculate the optimal solution in G using a dynamic programming algorithm that has a running time exponential in k but linear in the number of vertices. This solution often improves on each of the solutions of the heuristic (TODO: OR NOT, WAIT FOR RESULTS!).

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 set of good tours for our original graph $G' = (V, E')$, we merge these tours in one graph. All tours $E'_j \subset E'$, for $0 \leq j < \#tours$ 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. Before we show how we compute this decomposition in Section 3.2, we first give the defenition of a tree decomposition.

3.1 Tree decomposition and width

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T = (W, F), X)$, where T is a tree and $X = \{X_i \subset V : i \in W\}$ a set of *bags*, satisfying:

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.

Throughout this thesis we often work with the edge set corresponding to the vertex $i \in W$, rather than the vertex set X_i itself. 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.

Bodlaender and Koster [4] evaluated a number of construction heuristics. We chose the Minimum Degree Heuristic, originally designed by Markowitz [5], which is a simple but effective heuristic. It quickly obtains results close to the optimum and is easy to implement. The algorithm consists of the following steps:

1. Take the vertex $v \in V$ with minimum degree and add it to W.
2. Create a bag X_v with v and all it's neighbours.
3. Turn all the neighbours of v into a clique and remove v from V.
4. Add an edge (v, w) to F, where w is the neighbour of v with the smallest degree.
5. Repeat step 1 to 4 untill all vertices are processed and $V = \emptyset$.

To complete the tree decomposition we choose the first vertex of W to be the root of the tree. Finally we remove the last two vertices from W, because their bags only contain 1 or 2 vertices and are fully contained in another bag. We can do this because the graph G is obtained by merging tours, and therefore every vertex is guaranteed to have at least two neighbours. For the VRP we also add the depot vertex to every bag.

4 Dynamic programming

Provided the width is small enough, the optimal solution for the merged graph can be computed using a dynamic programming algorithm on the tree decomposition of the graph. In the following sections we explain the details of the algorithms.

4.1 Traveling Salesman

Let $G = (V, E)$ be a simple graph with edge-weights c_e and $(T = (W, F), X)$ 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. We say that a bag X_j is below a bag X_i (in the tree) if i is on the path from j to the root of T. 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. A series of these paths start and end in a bag, and visit all vertices in bags below that bag in the tree. Such a series of paths 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 any of the paths. For every pair of endpoints we have an edge $\{u, v\} : u, v \in V$ in the *matching* to mark which vertices are the endpoints of a path.

We now define the function $F(X_i, D, M)$ to be the minimum total cost of the edges in a series of paths starting and ending in bag X_i, where D is a set of degrees for the vertices in X_i and M a matching. If there is an edge $\{u, v\} \in M$ then there should be a path that starts in u and ends in v. All vertices in X_i itself should have degrees as given in the degrees parameter, and the vertices that occur only in the bags below X_i in the tree should all be used (have a degree of 2).

One way of looking at this is to see the set of degrees D as an instruction to a specific part of the tree (the bag X_i and all bags below in the tree) to deliver a set of edges, together forming a series of disjoint paths, such that all the degrees of vertices in this bag match with the degrees in D and that all vertices that occur only in bags below X_i in the tree are used. Of course, we do not just want any set of edges, we want the edges that can do it with the minimum cost. To get the cost of a tour through the entire graph, we can now call $F(X_0, D_0 = \{(v, 2), \text{ for } v \in X_0\}, \emptyset)$. The root of the tree is the special case where we allow the paths to form a (single) cycle. Therefore if we give the instruction to the root bag X_0 to give us a set of edges such that all the vertices inside X_0 itself have degree 2 (as required by the set D_0) and all vertices in bags below the root have degree 2 as well (by specification of the function F), we actually give the instruction to find the weight ofa set of edges that visits all the vertices of G in a single cycle. And because this set of edges should have minimum cost, this gives us the cost of the TSP tour through G.

Of course, for a good tree decomposition not all the edges are contained in a single bag. The main problem for a non-leaf bag X_i now is not how to find a subset of edges in Y_i that satisfy all the requirements for the degrees (and the matching), but how to divide the degrees over it's children so that they (recursively) can find the right edge sets that satisfy their part of the requirements. Selecting some edges from Y_i is mostly used to stitch the different paths from the child bags together so that the complete series of paths meets the requirements of D and M.

To find the different ways of dividing the requirements for a bag over it's children, we focus solely on the degrees. We also have to decide on the edges in the matching of the children, but we will find them as we set the requirements on the degrees. We try all possible combinations of dividing the degrees per vertex. For a given vertex v, assuming we are required to give it a degree of 2, 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 to give v a degree of at least 1, we try to use it as an endpoint coupled with 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 edge $\{u, v\}$ to $M_{p,j}$. Finally we try not to assign v to any of the child bags, so that we can give its degree with one of Y_i's edges. 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 required by 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 an edge of a matching are not allowed to be connected. This is the reason why we need to give the matching as parameter to F, without it we do not have sufficient information to determine which vertices can and which vertices cannot be connected. The root bag is of course an exception, because there all paths are merged in a single cycle.

In summary, a vertex' degree can be satisfied by passing it on to one (or two) of the child bags or in the bag itself by choosing an edge from Y_i. Formally this becomes

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

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

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. L., & Koster, A. M. (2010). *Treewidth computations I. Upper bounds.* Information and Computation, 208(3), 259-275.

[5] Markowitz, H. M. (1957). *The elimination form of the inverse and its application to linear programming.* Management Science, 3(3), 255-269.