OPTIMIZATION ON GRAPHS WITH RESTRICTED WEIGHTS

By

David Magagnosc

and

Michael Werman

IMA Preprint Series # 431

July, 1988

OPTIMIZATION ON GRAPHS WITH RESTRICTED WEIGHTS

DAVID MAGAGNOSC AND MICHAEL WERMAN*

Abstract. Many combinatorial optimization problems on graphs involve weighted edges. In this paper we solve several such problems on complete graphs with restricted weights. In particular, we give linear time algorithms for the traveling salesman problem, finding minimum spanning trees, and the all pairs shortest path problem (linear preprocessing, constant queries). We also indicate how to test the applicability of our algorithms. This extends the known results for bipartite and non-bipartite matching.

1. Introduction and Definitions. Many combinatorial optimization problems on graphs involve weighted edges. In this paper we solve several such problems on complete graphs with restricted weights. In particular, we give linear time algorithms for the traveling salesman problem, finding minimum spanning trees, and the all pairs shortest path problem (linear preprocessing, constant queries). We also indicate how to test the applicability of our algorithms. This extends the results of [MW88] which treats bipartite and non-bipartite matching.

Let $N = (V, E, s)$, $n = |V|$ be a graph with non-negative weight function $s : E \to \mathbf{R}_+$. s determines a distance function $d : V \times V \to \mathbf{R}_+$ by $d(x, y)$ equals the shortest path distance from x to y in N. Let G be the complete graph on $V(G) \subseteq V$, weighted by d. This G is the graph for which the optimization problems are to be solved. (For any G, there is an N which represents G if and only if G satisfies the triangle inequality.) Denote by $[x, y]$ a shortest x-y path in N. Let $k \stackrel{def}{=} |E| - |V| + 1$. There has been some work on computing such sparse networks for given graphs so that shortest paths in N approximate the original weight functions of a given graph G [Ch86], [DFS87], [PS88]; in these cases our algorithms will give fast approximations if N is sparse enough.

When k is small relative to n, the network contains few chains, i.e. maximal paths $C = x_0, \ldots, x_l$ whose interior vertices have degree two. A chain decomposition $\mathcal{C}$ is a collection of chains so that every edge of N belongs to precisely one chain in $\mathcal{C}$. Such a decomposition may be found easily by a graph traversal. The reduced network $N^* = (V^*, E^*)$ is the graph whose vertices are the endpoints of the chains in $\mathcal{C}$ and whose edges are the chains themselves. We will often refer to these as reduced vertices and edges, denoted by x^*, e^*, etc. Note that $k = |E^*| - |V^*| + 1$ as well. In the case of a isolated cycle, any vertex may be selected to be the reduced vertex, and the resulting reduced network is a single loop. When N is biconnected, every reduced vertex has degree at least 3, so that $3|V^*| \leq 2|E^*|$ and

$$|V^*| \leq 2k - 2 \quad \text{and} \quad |E^*| \leq 3k - 3.$$

*Institute for Mathematics and its Applications, University of Minnesota, Minneapolis, Minnesota 55455

Otherwise, the following "biconnectivity trick" is frequently applicable: let $C_1, \ldots, C_p$ be the biconnected components of N. These form a tree T, where two components are joined by an edge in T if they share a vertex or are adjacent along a bridge. The trick consists of solving part of the problem separately for each biconnected component (total size linear in k) and for the tree.

Certain configurations of N may be assumed not to appear (without loss of generality). 0-weight edges may be contracted. Non-graph vertices of degree two may be eliminated, joining its neighbors by an edge whose weight is the sum of the weights of its two incident edges. Finally, leaves in the tree of biconnected components which contain no graph vertices may be pruned (iteratively), so that we may assume that there are shortest paths passing through every component. All of these reductions may be performed while computing the biconnected components, which can be done in linear time.

Note that all of the algorithms may be easily parallelized in NC.

2. All Pairs Shortest Paths. We compute all pairs shortest paths in N in the following sense: we preprocess the network in time $O(n + k^2 \log k)$ in such a way that queries for any pair can be answered in constant time. Note that for a pair $\{x, y\}$ with x, y in different chains any x-y path contains a reduced vertex on each chain (could be the same vertex). Thus, if z_1 and z_2 are the endpoints of the chain containing z,

$$d(x,y) = \min_{i,j \in \{1,2\}} \{ d(x,x_i) + d(x_i,y_j) + d(y_j,y) \}.$$

If x and y belong to the same chain, then

$$d(x,y) = \min\{ |d(x_1,x) - d(x_1,y)|, d(x,x_1) + d(x_1,x_2) + d(x_2,y), \\ d(x,x_2) + d(x_2,x_1) + d(x_1,y) \}$$

Thus, it suffices to compute the local chain functions $d(x_1,x)$ and $d(x_2,x)$ and to be able to compute any pair shortest path in the reduced network in constant time.

If the network N is biconnected, then as observed in the introduction, $|N^*| = O(k)$, and all pairs shortest paths may be precomputed in time $O(k^2 \log k)$ [Ta83]. Otherwise, the reduced network has the structure of a tree T whose vertices are the biconnected components $C_1, \ldots, C_p$ and whose edges reflect the adjacencies among the C_i (two components which meet at an articulation vertex are connected by a 0-weight edge in the tree). Denote by v_j^i the reduced vertex in C_i which is closest to C_j and assume without loss of generality that C_1 has degree 1 in T. Let r be the articulation vertex of C_1 in N^*. Using depth first search and the all pairs shortest paths computed within each C_i, we compute $d(C_i, r)$ for every component C_i, where $d(C_i, r) = d(v_1^i, r)$. This takes $O(n)$ time. Finally, we set up a data structure in $O(n)$ time and space to find the lowest common ancestor to any two vertices in T^* (rooted at C_1) in constant time [SV88].

In order to answer shortest path queries in constant time, we use the precomputed information as follows. Let x and y be the reduced vertices whose shortest path distance is to be found and C_x and C_y their biconnected components. If $C_x = C_y$, then we have already computed $d(x, y)$. Otherwise, let C_z be the lowest common ancestor of C_x and C_y in T^*. Then (see Figure 1)

$$\begin{aligned} d(x,y) = d(x, v_1^x) + d(C_x, r) + d(C_y, r) + d(y, v_1^y) \\ - 2d(C_z, r) - d(v_x^z, v_1^z) - d(v_y^z, v_1^z) \\ + d(v_x^z, v_y^z). \end{aligned}$$

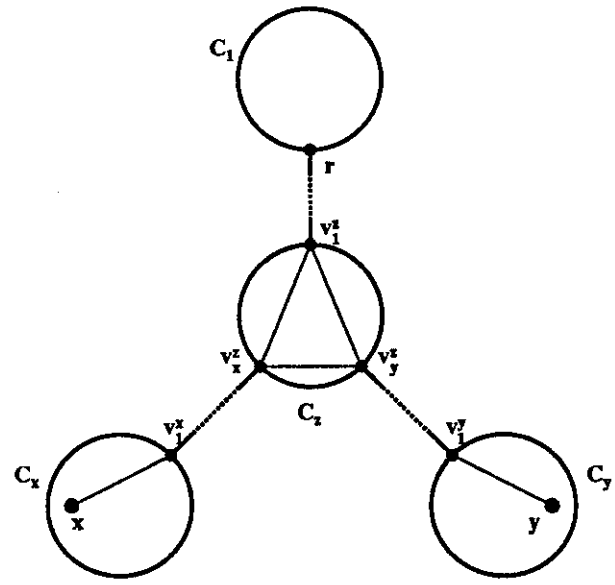

Figure 1

3. Minimum Spanning Tree. In the case in which all network vertices are also graph vertices, the situation is particularly simple:

PROPOSITION 3.1.. *Let $V(G) = V(N)$. Then any minimum spanning tree for N (as a graph) is a minimum spanning tree for G.*

Proof. Let T be a minimum spanning tree for N. Suppose that T is not a minimum spanning tree for G. Then there is an edge $e = xy$ in G and an edge $e' = x'y'$ in T so that $T' = (T + e) \setminus e'$ has smaller weight in G. But $(T + [x, y]) \setminus x'y'$ is a connected spanning subgraph of N with weight less than the weight of T, a contradiction. □

The minimum spanning tree algorithm has three distinct parts. First, non-graph vertices are eliminated by reassigning their potential contribution to new edges. Then, by Proposition 3.1, it uses standard minimum spanning tree algorithms to construct a minimum tree in the reduced network. Finally, a single edge of each chain excluded from this

tree is deleted to form a minimum spanning tree for the original graph. Let $e = xy$ be a minimum weight edge connecting an $x \in N \setminus G$ to a $y \in G$. Let $z_1, \ldots, z_l$ be by remaining neighbors of x. Then define a new network N' on $V(N) \setminus x$ by inserting all edges yz_i with weight $s(yz_i) = s(xy) + s(xz_i)$. An example of such a reduction is illustrated in Figure 2. Note that this new network contains one fewer non-graph vertex and one fewer edge, and so has the same value of k. By iterating the process, the algorithm constructs a network with no non-graph vertices.

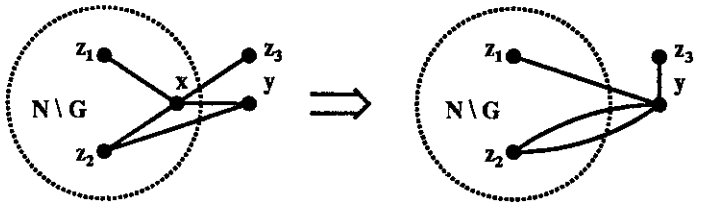

Figure 2

PROPOSITION 3.2.. *The weight of a minimum spanning tree for G with respect to N' is equal to the weight of a minimum spanning tree for G with respect to N.*

Proof. Let T be a minimum spanning tree for G with respect to N using paths $\{[x_i, y_i]\}$. Suppose that some path $[z_1, z_2]$ includes x but not y. Since z_1 and z_2 are adjacent in T, one, say z_1, is nearest to y in T. Replace $[z_1, x]$ by $[y, x] = yx$ to change $[z_1, z_2]$ to $[y, z_2]$. By minimality of xy, the resulting tree has the same weight as T. So, for any path $\ldots z, x, y \ldots$ in N, the same result is achieved by $\ldots z, y \ldots$ in N' with the same weight. □

By interating this process, we generate a network with no non-graph vertices and equal minimum spanning tree . This takes time $O(n + k \log k)$ using a heap containing edges from graph to non-graph vertices.

We have reached the situation of Proposition 3.1 and must now merely find a minimum spanning tree for N'. Note that any spanning tree omits at most one edge from each chain and that this edge is optimally taken to be the edge of maximum weight in the chain. Thus, we may compute the reduced network, weighting each of its edges equal to the maximum weight edge in the corresponding chain. Any minimum spanning tree algorithm may be used to compute a minimum spanning tree T^* for the reduced network (time $O(k \log k)$). Finally, to explicitly construct the tree, remove a maximum weight edge for each chain not in T^*, for a total of $O(n + k \log k)$. Note that if few $N \setminus G$-G edge reductions need to be done and faster general matching algorithms are employed, this bound may be reduced slightly, in the extreme case, down to $O(n + k)$ [Ta83].

4. The Traveling Salesman Problem. As an example of a difficult problem solvable within the framework of networks with few cycles, we give an algorithm for the travel-

ing salesman problem which runs in time $O(n+f(k))$ for some function f of k. Specifically, we wish to find a minimum weight cycle in G which visits every vertex. In terms of the network, this is equivalent to finding a minimum weight (not necessarily simple) cycle in N which visits every graph vertex.

To begin with, we analyze the behavior of the cycle in N with respect to the chains:

PROPOSITION 4.1. *There exists a minimum weight cycle $\mathcal{K}$ in N which does not traverse any edge of N more than once in each direction.*

Proof. Otherwise, suppose that some edge $e = xy$ is traversed twice in the same direction. Then $\mathcal{K}$ breaks into a sequence of four subpaths $\mathcal{K}_1 e \mathcal{K}_2 e$ ($\mathcal{K}_1$ or $\mathcal{K}_2$ may themselves traverse e). Then the new path $\mathcal{K}_1 \mathcal{K}_2^{-1}$ visits the same vertices as $\mathcal{K}$ while traversing e two fewer times and traversing every other edge the same number of times. This may be repeated as necessary to eliminate all double traversals. □

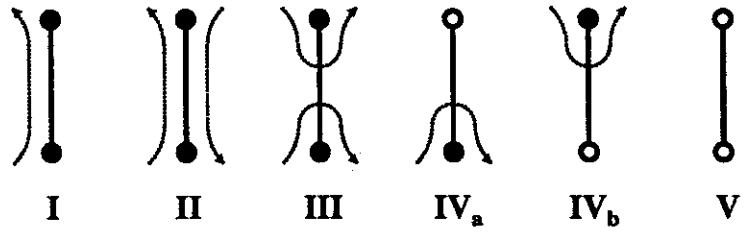

Figure 3

We consider for the remainder of this section only such cycles. As a consequence of the Proposition, Figure 3 shows all possible ways in which a chain may be traversed by such a $\mathcal{K}$. Here, the empty circles represent vertices which are not visited by $\mathcal{K}$ and the filled circles represent visited vertices, whether or not they are graph vertices as well. Let $C = x_0, \ldots, x_l$ be a chain, $x_i x_{i+1}$ be its maximum weight edge and $s(C)$ be the weight of the entire chain. Then the optimal contribution of the chain to the weight of $\mathcal{K}$ for each class of chain is

$$\begin{aligned} I: &\quad s(C) \\ II: &\quad 2s(C) \\ III: &\quad 2(s(C) - s(x_i x_{i+1})) \\ IV_a: &\quad 2(s(C) - s(x_{l-1} x_l)) \\ IV_b: &\quad 2(s(C) - s(x_0 x_1)) \\ V: &\quad 0 \end{aligned}$$

Consider now the reduced network $N^* = (V^*, E^*)$. A vertex (edge) will be call *essential* if it is (corresponds to a chain containing) a graph vertex. A labeling is a function $L : E^* \to \{ I, II, III, IV_a, IV_b, V \}$ An edge e will be said to be of *class* X if $L(e) = X$. A vertex will be called *active* if it corresponds to a solid vertex in Figure 4.1 (the cycle will

visit it). Edges of class I and II will also be called *active*. Define the graph $N' \subseteq N$ to be the graph of active vertices and edges. Then

PROPOSITION 4.2. *There is a many-to-one correspondence between cycles not traversing any edge more than once in each direction and labelings L with the following properties:*

i) N' is connected,

ii) all essential vertices are active,

iii) edges of class V are not essential,

iv) edges of class II are precisely the bridges of N', and

v) the biconnected components of N' (class I edges only) are Eulerian.

Proof. Given a labeling satisfying the properties, we construct a cycle as follows. Piece together eulerian cycles on the biconnected components of N' at articulation vertices in N' and bridges (class II edges). At the active vertices, detour into class III and IV edges to visit any graph vertices on those edges. By i), iv), and v) the eulerian cycles may be pieced together. By ii) all essential vertices are visited, and by iii) all graph vertices on essential edges are visited as well.

On the other hand, any cycle $\mathcal{K}$ in N determines a labeling L. Properties ii) and iii) follow because all graph vertices are visited by $\mathcal{K}$. Property i) holds since $\mathcal{K}$ is connected. Suppose that iv) fails for some class II edge $e^* = x^*y^*$, that is, e^* is not a bridge. Write $\mathcal{K}$ as $\mathcal{K}_1 e^* \mathcal{K}_2 (e^*)^{-1}$. Since e^* is not a bridge, $\mathcal{K}_1$ and $\mathcal{K}_2$ have a vertex z^* in common. Let $\mathcal{K}_i = \mathcal{K}_{i1} z^* \mathcal{K}_{i2}$ and make e^* class III. Then the cycle

$$\mathcal{K}' = \mathcal{K}_{11}\mathcal{K}_{22} \text{ (class } III \text{ detour on } e^*) \; \mathcal{K}_{21}\mathcal{K}_{12} \text{ (class } III \text{ detour on } e^*)$$

has lower weight than $\mathcal{K}$, a contradiction. Suppose that iv) fails for a bridge e^* of N', that is, e^* is of class I. Then $\mathcal{K}$ could not be a cycle. Thus, property iv) must hold. Finally, property v) follows from iv) and the observation that every class I edges is traversed exactly once. □

There is a certain amount of choice on class III edges in constructing $\mathcal{K}$ from a labeling L. Observe that all but one edge is traversed twice (no non-graph vertices are interior to chains). This edge should be chosen to have maximum weight in order to minimize the weight of $\mathcal{K}$. Similarly, detours for class IV_a and IV_b edges will extend across the chain omitting only the last edge.

This gives rise to a simple algorithm for finding a minimum weight cycle. Preprocess each chain to determine its contribution in each class and to determine if it is essential. Construct the reduced network N^*. Form each possible labeling $L : E^* \to \{I, \ldots, V\}$ and verify the properties i)–v), where the bridges and biconnected components may be determined as in Tarjan and Vishkin [TV84]. For each labeling satisfying all properties, compute the weight of the associated cycle. Select the minimum and construct the cycle

itself. This requires time $O(|N| + |E^*| \cdot 6^{|E^*|})$ and $O(|N|)$ space. The biconnectivity trick is applicable, resulting in an $O(|N| + k \cdot 6^k)$ algorithm, since bridges must necessarily be of class II.

5. Testing Applicability. Given a complete weighted graph $G = (V, E, s)$ there exists a network $N = (V, E', s)$ with $E' \subseteq E$ so that the shortest x-y path in N has weight $s(xy)$ as long as s satisfies the triangle inequality in G (E' can be taken to be E). We wish to compute the minimum such network, i.e. the network contained in any other such network. The fact that such a minimum network exists will follow from the algorithm:

1) Sort the edges by weight.
2) Compute a minimum spanning tree T of G.
3) Initialize $N = T$ and compute $d(x, y)$ as the shortest x-y path weight in N.
4) For each edge $e = xy$ in $G \setminus T$ do (in weight order):
   4a) If $d(x, y) < s(e)$, stop. G fails the triangle inequality.
   4b) If $d(x, y) = s(e)$, continue. e is redundant.
   4c) If $d(x, y) > s(e)$, add e to N and update d.

Step 1 takes time $O(n^2 \log n)$. Steps 2 and 3 take time $O(n^2)$. Steps 4a and 4b take constant time per edge, for a total of $O(n^2)$. Step 4c is executed exactly k times; each iteration can be performed in $O(n^2)$ time by

$$d_{new}(x', y') = \min \{ d(x', y'), d(x', x) + s(e) + d(y, y'), d(x', y) + s(e) + d(x, y') \}$$

for a grand total of $O(n^2(k + \log n))$.

To show correctness we must verify that the edges used are necessary and sufficient. Let N be some minimal network (as few edges as possible). Suppose $e = xy \in T \setminus N$. Let

$$x = x_0, x_1, \ldots, x_l = y$$

be a shortest path in N. Since T is minimal, $s(e) \leq s(x_i x_{i+1})$ for each i. But $l \geq 2$, a contradiction. Edges discarded in 4b are clearly unnecessary. Suppose $e = xy \notin N$ but e was included in step 4c, e being the first such edge encountered. Then since all later edges have weight at least as great as e, e is itself the unique shortest path from x to y.

If k is not $O(n)$, this algorithm take more than $O(n^3)$ time. However, we may always find the minimum network in $O(n^3)$ time as follows (assuming integral edge weights):

1) Let $s'(e) = s(e) - \frac{1}{|V|}$ for each $e \in E$.
2) Compute all pairs shortest paths in G with respect to s', and let $d(x, y)$ be the minimum distances.
3) For each edge $e = xy$,
   3a) if $d(x, y) < s(e) - 1$, G fails triangle inequality,

3b) otherwise, $e \in N \Leftrightarrow d(x,y) < s'(e)$.

The s' length of a path $x_0, \ldots, x_l$ is equal to the s length minus $l/|V|$. Thus, the minimum path length from x to y with respect to s is just $\lceil d(x,y) \rceil$. Moreover, among all shortest paths, we find the longest, thus detecting redundancy.

REFERENCES

[Ch86] P. Chew, "There is a Planar Graph Almost as Good as the Complete Graph," *Proc. 2nd Symp. on Computational Geometry,* 1986, pp. 169–177.

[DFS87] D. P. Dobkin, S. J. Friedman, and K. J. Supowit, "Delaunay Graphs are Almost as Good as Complete Graphs," *28th IEEE Symp. on Found. of Comp. Sci.,* 1987, pp. 20–26.

[MW88] D. Magagnosc and M. Werman, "Matching in Networks with Few Cycles," *IMA Preprint Series #419,* Institute for Mathematics and its Applications, University of Minnesota, 1988.

[PS88] D. Peleg and A. A. Schäffer, "Graph Spanners," preprint.

[SV88] B. Schieber and U. Vishkin, "On Finding Lowest Common Ancestors: Simplification and Parallelization," *SIAM Journal of Computing,* to appear, and preprint.

[Ta83] R. E. Tarjan, *Data Structures and Network Algorithms,* CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, Philadelphia, 1983.

[TV84] R. E. Tarjan and U. Vishkin, "Finding Biconnected Components and Computing Tree Functions in Logarithmic Parallel Time," *25th IEEE Symp. on Found. of Comp. Sci.,* 1984, pp. 12–20.