



## Reconstructing edge-disjoint paths

M. Conforti<sup>a</sup>, R. Hassin<sup>b,\*</sup>,<sup>1</sup>, R. Ravi<sup>c,2</sup>

<sup>a</sup>*Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, 35131 Padova, Italy*

<sup>b</sup>*Department of Statistics and Operations Research, School of Mathematical Sciences, Tel-Aviv University, Tel Aviv 69978, Israel*

<sup>c</sup>*GSIA, Carnegie Mellon University, Pittsburgh, PA 15213, USA*

Received 18 September 2002; received in revised form 21 January 2003; accepted 27 January 2003

### Abstract

For an undirected graph  $G = (V, E)$ , the edge connectivity values between every pair of nodes of  $G$  can be succinctly recorded in a flow-equivalent tree that contains the edge connectivity value for a linear number of pairs of nodes. We generalize this result to show how we can efficiently recover a maximum set of disjoint paths between any pair of nodes of  $G$  by storing such sets for a linear number of pairs of nodes. At the heart of our result is an observation that combining two flow solutions of the same value, one between nodes  $s$  and  $r$  and the second between nodes  $r$  and  $t$ , into a feasible flow solution of value  $f$  between nodes  $s$  and  $t$ , is equivalent to solving a stable matching problem on a bipartite multigraph.

Our observation, combined with an observation of Chazelle, leads to a data structure, which takes  $O(n^{3.5})$  time to generate, that can construct the maximum number  $\lambda(u, v)$  of edge-disjoint paths between any pair  $(u, v)$  of nodes in time  $O(\alpha(n, n)\lambda(u, v)n)$  time.

© 2003 Elsevier Science B.V. All rights reserved.

*Keywords:* flow equivalent tree, stable matching

### 1. Introduction

Given an undirected graph  $G = (V, E)$  with  $|V| = n$ , let  $\lambda(s, t)$  be the  $st$ -edge connectivity of  $G$ , i.e., the maximum number of edge-disjoint  $st$ -paths. Gomory and Hu [4] showed that the edge connectivity function

$\lambda = \{\lambda(s, t) : s, t \in V\}$  has a compact tree representation, i.e., there exists a weighted spanning tree on  $V$  such that for every pair of nodes  $s, t \in V$   $\lambda(s, t)$  is the minimum weight of an edge on the (unique)  $st$ -path in this tree. This tree is known as a *flow-equivalent tree* of  $G$ .

Suppose that a set of  $\lambda(s, t)$  edge-disjoint  $st$ -paths are given for every edge  $(s, t)$  of the  $|V| - 1$  edges of the flow equivalent tree: Can we efficiently construct  $\lambda(u, v)$  edge-disjoint  $uv$ -paths for an arbitrary pair  $u, v \in V$ ? Such a question may potentially arise in applications that need to compute the maximum flow, or alternately the maximum number of edge-disjoint paths, between arbitrary pairs of vertices at several points in the course of its execution.

\* Corresponding author.

*E-mail addresses:* [conforti@math.unipd.it](mailto:conforti@math.unipd.it) (M. Conforti), [hassin@post.tau.ac.il](mailto:hassin@post.tau.ac.il) (R. Hassin), [ravi@cmu.edu](mailto:ravi@cmu.edu) (R. Ravi).

<sup>1</sup> This paper was written while the author visited GSIA, Carnegie Mellon University.

<sup>2</sup> Supported in part by an NSF CAREER Award CCR 96-25297. Ravi also acknowledges support from IBM SRC, New Delhi, for hosting a visit during January–February 1999 when this paper was completed.

In this paper we describe a compact representation of the sets of the  $\lambda(u, v)$  edge-disjoint paths for every pair  $u, v \in V$ . This representation consists of a graph with node set  $V$  and  $O(n)$  edges, where  $n = |V|$ . Each edge  $(s, t)$  in this graph is associated with  $\lambda(s, t)$  edge-disjoint  $st$ -paths. This data structure can be computed in a preprocessing step that takes time  $O(n^{3.5})$  and  $O(n^3)$  space. We then show how to construct  $\lambda(u, v)$  edge-disjoint  $uv$ -paths for an arbitrary pair  $u, v \in V$  in  $O(\alpha(n, n)\lambda(u, v)n)$  time, where  $\alpha(n, n)$  is the inverse Ackermann function.

## 2. Stable matchings

Let  $G=(P, Q, E)$ ,  $|P|=|Q|$ , be a bipartite multigraph which is complete, i.e. every pair of nodes in  $P$  and  $Q$  are adjacent. Assume further that every node  $p \in P$  ranks the edges having  $p$  as end node according to its preference and every node  $q \in Q$  also ranks the edges having  $q$  as end node, so that every edge is ranked twice, at both end nodes. A perfect matching  $M$  of  $G$  is *stable* if for every edge  $e$  in  $E \setminus M$  with end nodes  $p$  and  $q$ , either in the  $p$ -ranking,  $e$  is less desirable than the edge  $e_p \in M$  that saturates  $p$ , or in the  $q$ -ranking,  $e$  is less desirable than the edge  $e_q \in M$  that saturates  $q$ . Gale and Shapley in their seminal paper [3] (see also [5]) show that every complete bipartite simple graph has a stable matching. Their proof is algorithmic and we give below a straightforward adaptation to the multigraph case.

### StableMatch(bipartite multigraph)

1. **Start** with  $M = \emptyset$ . Initially, all nodes in  $P$  are exposed and all edges are unexplored.
2. **While** a node  $p \in P$  is exposed, explore the unexplored edge  $e$  that has highest  $p$ -ranking. Let  $q$  be the other end node of  $e$ .  
**If**  $q$  is exposed,  
**Then** set  $M = M \cup \{e\}$ .  
**Else** if the edge  $e_q \in M$  that saturates  $q$  is less preferable than  $e$  in the  $q$ -ranking, set  $M = M \cup \{e\} \setminus \{e_q\}$ .

The matching  $M$  is stable upon termination of the algorithm. At the end,  $M$  is a perfect matching. For, assume not: then all the edges incident to  $p$  are explored, for some exposed node  $p$ . Note that when an edge is explored, its end node in  $Q$  is saturated and remains saturated throughout the algorithm. So when the least desirable edge in the  $p$ -ranking is explored and its end node in  $Q$  is saturated, all nodes in  $Q$  are saturated. This is impossible since  $|P| = |Q|$  and  $p$  is exposed.

Finally, an edge is explored at most once in the algorithm, so its complexity is  $O(|E|)$ .

We remark that the above problem can be interpreted as a *multi-ethnic marriage problem*, in which  $P$  represents the set of suitors,  $Q$  the set of brides, and the edges with end nodes  $p$  and  $q$  represent the set of possible marriage ceremonies that can unite  $p$  and  $q$ . A perfect matching that is stable corresponds to a set of ceremonies  $C$  that unites all the suitors to all the brides so that no suitor  $p$  and bride  $q$  would both prefer a ceremony not in  $C$  (possibly with other partners).

## 3. Composing flow solutions

Let  $P = \{p_1, \dots, p_f\}$  be a set of  $f$  edge-disjoint  $sr$ -paths and  $Q = \{q_1, \dots, q_f\}$  be a set of  $f$  edge-disjoint  $rt$ -paths. Since each flow path has at most  $n$  edges, it is straightforward to find a set of  $f$  edge-disjoint  $st$ -paths in the graph formed by the union of the  $sr$ - and  $rt$ -paths having  $O(fn)$  edges. Using a classical flow-augmenting algorithm to find such a decomposition takes  $O(f^2n)$  time [1]. Using a method of Karger and Levine [6], this can be accomplished in time  $O(f^{3/2}n)$ .

**Theorem 3.1.** *Let  $P = \{p_1, \dots, p_f\}$  be a set of  $f$  edge-disjoint  $sr$ -paths and  $Q = \{q_1, \dots, q_f\}$ , a set of  $f$  edge-disjoint  $rt$ -paths where each flow path has at most  $n$  edges. Then, there exists a set of  $f$  edge-disjoint  $st$ -paths such that each path in this set is the concatenation of a “prefix” of a path in  $P$  and a “suffix” of a path in  $Q$ . Moreover, this set can be computed in  $O(fn)$  time.*

**Proof.** Construct the following complete bipartite multigraph  $B = (P, Q, E)$ : The node sets  $P$  and

$Q$  represent the paths in  $P = \{p_1, \dots, p_f\}$  and  $Q = \{q_1, \dots, q_f\}$ . For every edge  $g$ , that is common to paths  $p_i$  and  $q_j$ ,  $B$  contains an edge  $e$  with end nodes  $p_i$  and  $q_j$ . If, after adding all these edges the resulting bipartite multigraph is not complete, add a “dummy” edge between each pair of nonadjacent nodes in  $P$  and  $Q$ , to make it complete. The priority (from most desirable to least desirable) of the edges of  $B$  having  $p_i$  as end node is given by the order in which the edges are encountered when traversing path  $p_i$  from  $s$  to  $r$ . The “dummy” edges receive the lowest possible priority (the ranking among them is immaterial). The priority of the edges of  $B$  having  $q_j$  as end node is given by the order in which the edges are encountered when traversing path  $q_j$  from  $t$  to  $r$ . Again, the “dummy” edges receive the lowest possible priority.

From a stable perfect matching  $M$  of  $B$  one can construct the desired  $st$ -paths as follows: For every edge  $e$  in  $M$  with end nodes  $p_i$  and  $q_j$  which is not a dummy edge, traverse path  $p_i$  starting from  $s$  until  $e$  is met and then continue on  $q_j$  to  $t$ . (Edge  $e$  may or may not belong to the path thus constructed.) For every edge  $e$  in  $M$  with end nodes  $p_i$  and  $q_j$  which is a dummy edge, traverse path  $p_i$  starting from  $s$  to  $r$  and then traverse  $q_j$  from  $r$  to  $t$ .

The fact that the matching  $M$  is stable on  $B$  insures that the  $f$   $st$ -paths thus constructed are edge disjoint. Indeed, suppose for a contradiction that an edge  $g$  is used in two of these concatenated paths, which are represented by two edges in the stable matching, say  $(p^1, q^1)$  and  $(p^2, q^2)$ . These edges are witnessed by the fact that there are edges  $g^1$  common to  $p^1$  and  $q^1$  and  $g^2$  common to  $p^2$  and  $q^2$ . Since  $p^1$  and  $p^2$  are disjoint, the edge  $g$  must occur in only one of them, so assume that  $g$  occurs in  $p^1$  and  $q^2$ . Since  $g$  is in the concatenated path from  $p^1$  and  $q^1$ , it must be the case that  $g$  occurs before  $g_1$  in  $p^1$  going from  $s$  to  $r$ : This means that an edge between  $p^1$  and  $q^2$  in the auxiliary bipartite multigraph has higher priority than the edge  $(p^1, q^1)$  witnessed by  $g^1$  in the  $P$ -ranking. Similarly, since the edge  $g$  occurs in the concatenated path from  $p^2$  and  $q^2$ , it must be the case that  $g$  occurs before  $g^2$  in  $q^2$  going from  $t$  to  $r$ : This means that the edge between  $p^1$  and  $q^2$  in the auxiliary bipartite multigraph has higher priority than  $(p^2, q^2)$  witnessed by  $g^2$  in the  $Q$ -ranking. Thus, this unmatched edge  $(p^1, q^2)$  violates the definition of stability of the matching found, a contradiction.

Note however, that the concatenated paths constructed as above, while being edge disjoint, may not be simple, in which case we can delete cycles without destroying the ‘prefix-suffix’ property. This clean-up step takes time proportional to the size of the paths.  $\square$

We finally remark that every stable matching problem in a complete bipartite multigraph can be converted into a path-pairing problem of the above type between  $f$  edge-disjoint  $sr$ -paths and  $f$  edge-disjoint  $rt$ -paths.

#### 4. Augmenting flow-equivalent trees

Given the above method for composing a pair of edge-disjoint path solutions, we now show how we can maintain the maximum edge-disjoint paths solution for  $O(n)$  pairs of nodes in an  $n$ -node undirected graph, so that the maximum edge-disjoint paths solution for any arbitrary pair of nodes  $s$  and  $t$  can be recovered by applying the stable matching procedure  $O(\alpha(n, n))$  times. We exploit the natural connection that the maximum number of edge-disjoint paths in unit capacity undirected graph between a pair of nodes is equal to the value of the maximum flow between them [1], and use the flow-equivalent tree as our starting point.

Consider a pair of nodes  $s$  and  $t$  separated by  $k$  edges in a given flow-equivalent tree. To compute the maximum flow between them using the above procedure, we must use  $k$  applications of the procedure. The key to speeding this up is to add  $O(n)$  additional flow solutions in such a way that for any pair of nodes, there always exists a small number of pairs of nodes connecting them from which we can compose the required flow. Note that for any pair of nodes, the flow decomposition of a maximum flow (say  $f$ ) solution can be computed as mentioned above in time  $O(f^{1.5}n) = O(n^{2.5})$  time. This will lead to a total time complexity of  $O(n^{3.5})$  for this preprocessing step since we need to do this for  $O(n)$  pairs. In the unit capacity case, every pair of nodes can have  $O(n)$  paths in their max-flow decomposition leading to a space requirement of  $O(n^3)$  for this data structure. Next, we describe how to specify these pairs.

To do this, we use a method due to Chazelle [2]: Given an  $n$ -node edge-weighted tree, he provides an

algorithm to choose  $O(n)$  shortcut edges with weights on them such that for any path in the given tree, it is possible to compute the partial sum of the weights in the path using  $O(\alpha(n, n))$  summations involving the original and added edges (see Theorem 2 in [2]). More formally, Chazelle proved the following under the RAM model of computation.

**Theorem 4.1** (Chazelle [2]). *Let  $T$  be a free tree with  $n$  weighted edges. There exists a constant  $c > 1$  such that, for any integer  $m > cn$ , it is possible to sum up weights along an arbitrary query path of  $T$  in time  $O(\alpha(m, n))$ . The data structure is of size at most  $m$  and can be constructed in time  $O(m)$ .*

Chazelle's result is framed in a more general setting where the weight function maps the edges to a semigroup, and the partial sum in the above theorem can be replaced with the semigroup operation. We use this generalization and observe that  $(Z^+, \min)$  is a semigroup, and hence Chazelle's construction applies to deriving the minimum-weight edge along a tree path (rather than the sum) using shortcut edges. In fact, this is accomplished by weighting every shortcut edge with the minimum weight of an edge along the tree-path between its endpoints. We also maintain a maximum flow decomposition between pairs of nodes connected by shortcut edges. This enables us to reconstruct the maximum flow for any pair of nodes using  $O(\alpha(n, n))$  flow compositions. Each flow composition was argued

earlier to take time  $O(fn)$  for a flow of value  $f$  giving the claimed time of  $O(\alpha(n, n)\lambda(u, v)n)$  time to reconstruct the maximum number of flow paths  $\lambda(u, v)$  between any pair of nodes  $(u, v)$ . We thus have our main theorem.

**Theorem 4.2.** *Given an undirected unit capacity graph on  $n$  nodes, in time  $O(n^{3.5})$ , a data structure using space  $O(n^3)$  can be constructed that, given any pair of nodes, can compute the maximum number  $f$  of edge-disjoint paths between them in time  $O(\alpha(n, n)fn)$  where  $\alpha(n, n)$  is the inverse Ackermann function.*

## References

- [1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.
- [2] B. Chazelle, Computing on free trees via complexity-preserving mappings, *Algorithmica* 2 (1987) 337–361.
- [3] D. Gale, L.S. Shapley, College admissions and the stability of marriage, *Am. Math. Monthly* 69 (1962) 9–15.
- [4] R.E. Gomory, T.C. Hu, Multi-terminal network flows, *SIAM J. Appl. Math.* 9 (1961) 551–556.
- [5] D. Gusfield, R.W. Irving, *The Stable Marriage Problem: Structure and Algorithms*, The MIT Press, Cambridge, MA, 1989.
- [6] D.R. Karger, M.S. Levine, Finding maximum flows in undirected graphs seems easier than bipartite matching, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, Dallas, TX, 1998, pp. 69–78.