MATCHING IN NETWORKS WITH FEW CYCLES

By

David Magagnosc

and

Michael Werman

IMA Preprint Series # 419
April 1988
MATCHING IN NETWORKS WITH FEW CYCLES

DAVID MAGAGNOSC AND MICHAEL WERMAN*

Abstract. A central problem in combinatorial optimization is the problem of finding a minimum cost matching of vertices in a graph. In general, this problem takes time $O(n^3)$ to solve sequentially, where $n$ is the number of vertices in of the graph (we are dealing only with complete graphs). There is no known parallel algorithm running in deterministic polylogarithmic time. There do exist polylog randomized parallel algorithms for matching, and also a deterministic algorithm for graphs whose adjacency matrix has small permanent. In this paper we exhibit an $O(m \log m)$ sequential algorithm for bipartite matching and an $O(m\alpha)$ sequential algorithm for non-bipartite matching in complete graphs for a class of weight functions determined by a network of size $m$. The time for the bipartite algorithm may be improved to $O(m^{\log_2 n})$ when the number $n$ of points to be matched is not too large. This improves the known time bound for the case of matching on a cycle. Moreover, both algorithms are parallelizable.

1. Introduction and Definitions. A central problem in combinatorial optimization is the problem of finding a minimum cost matching of vertices in a graph. In general, this problem takes time $O(n^3)$ to solve sequentially (see [Ga86]), where $n$ is the number of vertices in of the graph (we are dealing only with complete graphs). There is no known parallel algorithm running in deterministic polylogarithmic time. There do exist polylog randomized parallel algorithms for matching ([KUW84], [GP85], [Ka86]), and also a deterministic algorithm for graphs whose adjacency matrix has small permanent ([KG87]).

In this paper we exhibit an $O(m \log m)$ sequential algorithm for bipartite matching and an $O(m)$ sequential algorithm for non-bipartite matching in complete graphs for a class of weight functions determined by a network of size $m$. The time for the bipartite algorithm may be improved to $O(m^{\log_2 n})$ when the number $n$ of points to be matched is not too large. This improves the known time bound for the case of matching on a cycle [WPMK86]. Moreover, both algorithms are parallelizable.

Let $N = (V, E, s)$ be a connected network (graph) with weight function $s: E \to \mathbb{R}_+$. $s$ determines a distance function $d: V \times V \to \mathbb{R}_+$ by $d(x, y) = \text{length of the shortest weighted path from } x \text{ to } y \text{ in } N$. Let $B$ and $R$ be multisets selected over the base set $V$ with $|B| = |R|$. Let $K_{B, R}$ be the complete bipartite graph with parts $B$ and $R$, weighted by $d$. Our algorithm determines a minimum weight perfect matching in $K_{B, R}$. Alternatively, for the non-bipartite case, let $B$ be a multiset over $V$ with $|B|$ even. Let $K_B$ be the complete graph on $B$, again, weighted by $d$. Our algorithm determines a minimum weight perfect matching in $K_B$. Note that since the graphs are complete, the permanents of the adjacency matrices are large. However, to achieve our time bounds, we assume that $N$ has few cycles, i.e. $k(N) \overset{\text{def}}{=} |E| - |V| + 1$ is small. This generalizes and improves the time bound for the known case $k(N) = 1$, i.e. one cycle ([WPMK86]).

*Institute for Mathematics and its Applications
An alternative formulation of the bipartite matching problem may be given in terms of network flow. Let $B$ be a multiset of unit sources and $R$ a multiset of unit sinks. Then a minimum weight matching is equivalent to a minimum cost flow, where the capacities are infinite. This correspondence may be exhibited by a collection $\mathcal{P} = \{[b_i, r_i]\}$ of shortest paths between matched elements $b_i$ and $r_i$.

Let $\text{len}([b, r])$ denote the length of the path $[b, r]$. The weight of a matching $\mathcal{P} = \{[b_i, r_i]\}$, weight($\mathcal{P}$), is defined to be $\sum_i \text{len}([b_i, r_i])$, and the weight of the optimum matching is

$$M(N, B, R) \overset{\text{def}}{=} \min_{\mathcal{P}} \text{weight}(\mathcal{P}).$$

For a matching $\mathcal{P}$ and edge $e \in E$, let $t(e)$ denote the number of paths in $\mathcal{P}$ traversing $e$. For a subset $S \subseteq V$, let $b(S)$ denote the blue excess in $S$, i.e. $|S \cap B| - |S \cap R|$.

Note that if there is a vertex common to $B$ and $R$ (bipartite case) or a vertex of multiplicity greater than one in $B$ (non-bipartite case), then these elements may be matched and removed in pairs without affecting the optimum matching. Thus we may assume that $B \cap R = \emptyset$ for the bipartite case and that $B$ is a set for the non-bipartite case.

2. A Structural Theorem. In this section we show that, for arbitrary $N = (V, E, s)$ and $B$ and $R$ multisubsets of $V$, there is a subtree $T \subseteq N$ for which $M(N, B, R) = M(T, B, R)$. In other words, some subtree of $N$ already contains enough information to find an optimum matching in $N$, even though the distance function defined by $T$ is generally larger for most pairs of vertices than the function defined by $N$. To prove this assertion, we begin with the simplest case: $N$ is a cycle.

**Proposition 2.1.** Let $N = (V, E, s)$ be a cycle, $B$ and $R$ multisubsets of $V$ with $|B| = |R|$. Then there exists an edge $e \in E$ so that $M(N, B, R) = M(N - e, B, R)$.

**Proof.** Let $\mathcal{P} = \{[b_i, r_i]\}$ be a minimal weight matching. If some edge of the cycle is not covered by a path in $\mathcal{P}$, then we are done. We may assume without loss of generality that $\mathcal{P}$ does not contain the configurations illustrated in Figure 2.1. In this figure, as in all subsequent figures, blue elements are represented as filled circles, red elements as empty circles, and other elements as small circles. Matching paths are indicated with dotted arrows.

This is so because occurrences of configurations 1 and 3 may be removed by replacing the indicated matchings by $b \rightarrow r'$ and $b' \rightarrow r$ without increasing the weight of the matching. Configuration 2 contradicts the assumption that no red and blue elements lie at a common vertex of $N$.

Define a major source to be a vertex $b$ on the cycle which has matching paths leaving in both directions. Similarly, a major sink is a vertex $r$ on the cycle which has matching paths arriving from both directions. Minor sources and sinks have paths leaving and arriving in one direction only. A major path is a path $[b, r]$, where both $b$ and $r$ are major. Then
Lemma 2.2. Let \( N \) be a cycle and \( \mathcal{P} \) be a minimum matching such that none of the forbidden configurations appears and every edge is traversed by some path in \( \mathcal{P} \). Then there exists a major source. Furthermore, major sources and major sinks alternate around the cycle.

Proof. Let \( xy \) be any edge on the cycle and \([b, r]\) be the longest matching path covering \( xy \). Let \( z \) be the vertex adjacent to \( b \) on the other side of \( b \) from \( r \). The matching path covering edge \( zb \) cannot be oriented in the same direction as \([b, r]\) without producing configurations 1 or 2. It cannot originate strictly on the \( r \) side of \( b \) without producing configuration 3. Thus, it begins at \( b \) and is oriented away from \( r \), so \( b \) is a major source. See Figure 2.2 (I).

Let \( b \) be any major source. Let \([b, r]\) be the longest clockwise matching path from \( b \). Let \( x \) be the next vertex of the cycle following the path \([b, r]\) and \([b', r']\) be a matching path covering \( rx \). Suppose \([b', r']\) is oriented in the same direction as \([b, r]\). \([b', r']\) cannot completely cover \([b, r]\) since \( b \) is a major source. \( b' \neq b \) since \([b, r]\) was the longest matching path from \( b \). However, \([b', r']\) cannot begin strictly on the \( r \) side of \( b \) without producing configuration 1 or 2. Thus \([b', r']\) cannot be oriented in the same direction as \([b, r]\). Finally, it cannot cross vertex \( r \) without producing configuration 3, so \( r' = r \) and so the next major vertex appearing in a clockwise direction from \( b \) is a major sink. Similarly, the next major vertex appearing in a counterclockwise direction from \( b \) is a major sink. See Figure 2.2 (II).

Lemma 2.3. For any adjacent major vertices \( b \) (a major source) and \( r \) (a major sink), there is an edge \( xy \) between \( b \) and \( r \) covered only by major paths.

Proof. Let \([b, x]\) be the longest minor path directed from \( b \) toward \( r \) and \( y \) the next vertex after \( x \). Since \( r \) is a major sink, \( x \neq r \). Suppose there is a minor path covering \( xy \). Then it must be directed in the same direction as \([b, x]\) lest it produce configuration 3 with
a major path \([b, r]\). However, this is impossible for precisely the same reasons as given in
the proof of Lemma 2.2. See Figure 2.3.

We will define a new minimum weight matching in \(N\) without altering the minor paths.
Thus we may assume for simplicity that only major paths occur. Recall that \(t(xy)\) for an
edge \(xy\) is the number of paths which traverse \(xy\). Let \(c\) be the minimum value of \(t(xy)\)
over all edges \(xy\). We define two new matchings given by altering the matching paths as
follows and as illustrated in Figure 2.4:

- **Flip\((N, B, R)\)**: matching obtained by changing \(c\) counterclockwise paths to
clockwise for each (major) source \(b\).
- **Flop\((N, B, R)\)**: matching obtained by changing \(c\) clockwise paths to counterclockwise
   for each (major) source \(b\).

**Lemma 2.4.** \(\text{weight(Flip}(N, B, R)) = \text{weight(Flop}(N, B, R)) = M(N, B, R)\).

**Proof.** Let the edge \(xy\) be covered \(l\) times in the original matching. In \(\text{Flip}(N, B, R)\)
it is covered \(l + c\) times and in \(\text{Flop}(N, B, R)\) it is covered \(l - c\) times (or vice versa).
These numbers are all non-negative by minimality of \(c\). Now the matching weight is the
weighted sum of the covering numbers of the edges, so

\[
\text{weight(Flip}(N, B, R)) + \text{weight(Flop}(N, B, R)) = 2M(N, B, R).
\]

But, \(M(N, B, R)\) is minimum, so all are equal.  

Finally, at least one of Flip and Flop uncovers an edge (namely, an edge with minimum covering number). Thus, for such an edge $e$, $M(N, B, R) = M(N - e, B, R)$, and the Proposition is proved.

In fact, this Proposition generalizes to arbitrary graphs $N = (V, E, s)$:

**Theorem 2.** Let $N = (V, E, s)$ be any connected graph, $B$ and $R$ multisubsets of $V$ with $|B| = |R|$. Then there exists a spanning subtree $T \subseteq N$ so that $M(N, B, R) = M(T, B, R)$.

**Proof.** Induction on $k(N) = |E| - |V| + 1$.

When $k(N) = 0$, $N$ is already a tree and there is nothing to prove.

Let $k(N) > 0$ and suppose the theorem holds for all graphs with smaller $k(N)$. Since $k(N) > 0$, $N$ contains a cycle $N'$. Let $\mathcal{P}$ be a minimum weight matching in $N$ in which $[b, r]$ is the minimum weight path from $b$ to $r$ in $N$. Consider the segments of the matching paths which lie on $N'$. These constitute a matching in $N'$ in their own right, and by the Proposition, this matching may be refined to one which leaves some edge $e$ of $N'$ untraversed. By linking this matching with the remaining path segments, we obtain a new minimum matching which leaves $e$ untraversed. Thus, $e$ may be deleted and the Theorem follows. This process is illustrated in Figure 2.5 for a single matching path. Initially, the path matches $b$ and $r$ by way of $b_1$, $r_1$, $b_2$, and $r_2$, and the induced matching on $N'$ is $\{[b_1, r_1], [b_2, r_2]\}$. When this matching is changed to $\{[b_1, r_2], [b_2, r_1]\}$, the original matching path matches $b$ to $r$ via $b_1$ and $r_2$. Notice that there is a cycle left over; in fact, this cannot occur when $\mathcal{P}$ is a minimum weight matching.
In particular, for each such matching path $[b,r]$, let \{ $[b_1, r_1], \ldots, [b_n, r_n]$ \} be the connected components of $[b,r] \cap N'$, directed as in $[b,r]$ ($n$ may be 0). We assume without loss of generality that $b \notin B \cap N$ and $r \notin R \cap N$ (otherwise introduce 0 weight edges). Let

$$B' = \{ b_i : [b,r] \text{ is a matching path in } \mathcal{P} \} \quad \text{and} \quad R' = \{ r_j : [b,r] \text{ is a matching path in } \mathcal{P} \}$$

as multisets. The elements of $B'$ and $R'$ are matched by the paths $[b_i, r_i]$; let $m$ be the value of this matching in $N'$.

Now let $\mathcal{P}'$ be an optimal matching of $B'$ with $R'$ in $N'$ which moreover fails to cover some edge $e$ on the cycle. This is possible by Proposition 2.1. Define a directed graph $(V', E')$ on

$$V' = B \cup B' \cup R \cup R'$$

by

I. $br \in E'$ if $b \in B$, $r \in R$ and $[b,r]$ does not share an edge with $N'$,

II. $bb' \in E'$ if $b \in B$, $b' \in B'$ and $[b,b']$ is the initial connected segment of $[b,r] \setminus N'$,

III. $r'r \in E'$ if $r' \in R'$, $r \in R$ and $[r',r]$ is the final connected segment of $[b,r] \setminus N'$,

IV. $r'b' \in E'$ if $r' \in R'$, $b' \in B'$ and $[r',b']$ is a connected segment of some $[b,r] \setminus N'$, and

V. $b'r' \in E'$ if $b' \in B'$, $r' \in R'$ and $[b',r'] \in \mathcal{P}'$.

In this graph, every vertex has indegree and outdegree at most one. Moreover, the original blue vertices have indegree zero and the original red vertices have outdegree zero. Consequently, the graph decomposes into maximal paths which begin at elements of $B$ and end
at elements of $R$ and into cycles. This defines a new matching $\hat{P}$ of $B$ to $R$ in $N - e$, where cycles are discarded.

Claim: weight($\hat{P}$) = weight($P$):

weight($P$) = weight of external path segments + $m$
weight($\hat{P}$) = weight of external path segments + weight($P'$) - weight(cycles).

But weight($P'$) $\leq m$ so weight($\hat{P}$) $\leq$ weight($P$). Equality follows from the minimality of weight($P$).

Finally, $k(N - e) = k(N) - 1$, so the result follows. □

3. The Algorithm. We now describe an algorithm for computing a minimum weight bipartite matching in a network $N = (V, E, s)$ between multisets $B, R \subseteq V$ with $|B| = |R|$. We begin with an algorithm for the simplest case, namely, the case in which $N$ is a tree. This restricted algorithm will also be used in the general case.

3.1 Trees. The matching is constructed during a post-order traversal of the tree $T = N$, with any node selected to play the role of the root. After the subtree rooted at a node $x$ is processed, there will be a certain excess of blue or red elements at vertices of the subtree left unmatched; there will never be an excess of both blue and red elements. These unmatched elements are passed to the parent in a list. When all the subtrees rooted at children of a node $x$ have been processed their parent $x$ matches the excesses from the lists received from its children as completely as possible, although otherwise arbitrarily. Since $|B| = |R|$, the algorithm terminates with no excess remaining unmatched at the root.

**Proposition 3.1.** This algorithm produces a minimum weight matching.

**Proof.** Let $xy \in T$ be any edge, $T_x$ and $T_y$ be the components of $T - xy$ containing $x$ and $y$, respectively. Recall that $b(T_x)$ is the blue excess in $T_x (|T_x \cap B| - |T_x \cap R|)$; similarly, $b(T_y)$ is the blue excess in $T_y$. Then $b(T_x) = -b(T_y)$. Let $P$ be any matching. Then the number of times $t(xy)$ that the edge $xy$ is traversed is at least $|b(T_x)|$, and the contribution to the weight of the matching is at least $|b(T_x)s(xy)|$. Assume that $x$ is the parent of $y$ in $T$. Then $y$ passes only blue elements or red elements to $x$. Thus there are no bidirectional edges. Consequently, the minimum possible number of traversals for each edge is achieved and the matching has the minimum possible weight. □

This algorithm runs in time and space $O(|T|)$ irrespective of $|B| = |R|$ as follows: each node maintains its excess in a stack with entries consisting of pairs (position, number), where the position refers to a vertex of $T$ at which elements of $B \cup R$ lie and number indicates the number of such elements at that vertex which have yet to be matched. Furthermore, the position entries in the stack are distinct. The parent vertex’s operations consist of concatenation and of popping and matching. Since each matching operation
exhausts all residents of at least one node and the concatenation joins stacks with distinct positions, the time and space bounds follow, even though explicit enumeration of matched pairs require \(|B|\) time and space. An explicit description of the distinct matching paths might require \(O(n^2)\) space, although it is of course encoded by the tree together with the matching.

3.2 Reduced Networks and Local Chain Functions. Now let \(N = (V, E, s)\) be any connected graph, \(B, R \subseteq V\) equinumerous multisets. There is a set of \(k(N) = |E| - |V| + 1\) edges whose removal leaves a spanning tree \(T\) so that \(M(T, B, R) = M(N, B, R)\) by Theorem 2. Our purpose is to discover such a set of edges and then to find the minimum matching using the tree algorithm described above.

Our algorithm proceeds with the following major steps:

I. Compute a chain decomposition \(C\) of \(N\) and the reduced network \(N^*\).

II. For each chain in \(C\), compute a local chain function.

III. For each spanning tree \(T^*\) of \(N^*\), compute the minimum matching value \(M(T, A, B)\) given that the edges in \(N - T\) do not lie in the chains corresponding to edges of \(T^*\) (using the local chain functions computed in II).

IV. Using the optimum tree, compute a minimum matching.

Define a chain to be a path \(P = v_0, \ldots, v_m\) so that \(\deg(v_i) = 2\) for \(i = 1, \ldots, m - 1\). For each chain \(P\), no more than one of its edges will be removed from \(N\) to form \(T\). Define a chain decomposition \(C\) to be a collection of chains in \(N\) so that every edge of \(N\) belongs to precisely one chain in \(C\). The reduced network \(N^* = (V^*, E^*)\) is the graph whose vertices are the endpoints of the chains in \(C\) and whose edges are the chains themselves.

Let \(V^*\) be the set of all vertices of \(N\) with degree not equal to two. (In case \(N\) is a cycle, let \(V^*\) consist of any single vertex.) Then the chains \(E^*\) may be constructed by beginning at a vertex of \(V^*\) and building a path until a second vertex in \(V^*\) is reached. Repeating the process until all edges have been traversed completes the construction. Note that this process in fact constructs a chain decomposition of \(N\) into maximal chains. With a depth-first search (or other standard) implementation, the construction of the reduced graph \(N^*\) take \(O(|N|)\) time and space.

The local function for a chain \(P\) reflects the contribution of \(P\) to the value of a maximum matching in a tree \(T\) formed by cutting an edge of \(P\) (along with other chains). This function depends on the particular edge of \(P\) which is cut. Let \(P = v_0, \ldots, v_m\) be a chain. For each edge \(e = v_i v_{i+1}\), define the triple \((K^*_e, K^*_e, w^e)\), where \(K^*_e\) is the blue excess \(b(\{v_0, \ldots, v_i\})\) and \(K^*_e\) is the blue excess \(b(\{v_{i+1}, \ldots, v_m\})\). \(w^e\) is the contribution of edges of \(P\) to the value of a minimum matching when the edge \(e\) is cut.

We make the following observations:

1) \(w\) is independent of the choice of minimum matching,

2) \(K^*_s + K^*_t\) is constant, so \(K^*_t\) is functional in \(K^*_s\),
3) $w$ is also functional in $K_s$, and
4) the function $K_s \mapsto w(K_s) + dK_s$ is concave upwards for any $d$.

Proof.
1) Claim: $s^e = \sum_{e' \in P} w(e')|K_s^{e'} - K_s^e|$:

The blue excess between $e'$ and $e$ is $K_s^{e'} - K_s^e$. Since no matching paths cross $e$, the minimum contribution of $e'$ is $w(e')|K_s^{e'} - K_s^e|$ which is easily achieved (cf. the tree algorithm).

2) $K_s^e + K_s^e$ is the total blue excess on $P$ (i.e. $b(P)$).
3) Immediate from the claim in 1).
4) Let $K$ and $K'$ be consecutive values of $K_s$ in size.

Then

$$w(K') - w(K) = \sum_{e \in P} s(e)(|K_s^e - K'| - |K_s^e - K|)$$

$$= \sum_{e \in P \atop K_s^e \leq K < K'} s(e)(K' - K_s^e - K + K_s^e)$$

$$+ \sum_{e \in P \atop K < K' \leq K_s^e} s(e)(K_s^e - K' - K_s^e + K)$$

$$w(K') - w(K) = (K' - K) \left[ \sum_{e \in P} s(e) - \sum_{e \in P} s(e) \right]$$

and

$$(w(K') + dK') - (w(K) + dK) = (K' - K) \left[ d + \sum_{K_s^e \leq K} w(e) - \sum_{K_s^e \leq K} w(e) \right]$$

As $K$ and $K'$ increase, the sum for $K_s^e \leq K$ increases and the sum for $K' \leq K_s^e$ decreases. Therefore, $((w(K') + dK') - (w(K) + dK))/(K' - K)$ is increasing, proving the observation.

Local chain information is illustrated in Figure 3.1, in which the matching shown is an optimum matching for $K_s = 1$. The shaded area represents the sum in the proof of observation 1); note that the height of the shaded area at any point is equal to the number of matching paths traversing that point. The total weight of the segments comprising $\sigma_0$ is used below in the computation of the local chain function.

We assume that $w$ is defined over the entire real line by means of linear extensions beyond the range of occurring $K_s$ and linear interpolation in between. For integer values between consecutive values of $K_s$, this is equivalent to the introduction of 0 weight edges separating multiple occurrences of blue or red elements at a single vertex of $P$. 

9
To compute the local chain triplets and hence the functions \( w(K_s) \) and \( K_l(K_s) \):

I. Compute the excesses \( K_s \) by traversing the path \( P \),

II. Compute \( K^c_l = b(P) - K^c_s \) (observation 2),

III. Sort the \( K_s \) that occur in step I: \( K_0, K_1, \ldots, K_l \), and compute the values

\[
\sigma_i = \sum_{e \in P, K_s^c = K_i} s(e)
\]

\[
\Delta = -\sum_{i=1}^{l} \sigma_i,
\]

IV. For the minimum \( K_0 \) compute

\[
w(K_0) = \sum_{e \in P} s(e)(K^c_s - K_0),
\]

and

V. For each \( 1 \leq i \leq l \) compute

\[
w(K_i) = w(K_{i-1}) + \Delta \cdot (K_i - K_{i-1})
\]

\[
\Delta \leftarrow \Delta + \sigma_{i-1} + \sigma_i.
\]

Each step can be done in linear time except for the sort which takes time \( O(l \log l) \leq O(m \log m) \). Thus the computation of the local chain functions for all of the chains in \( \mathcal{C} \) takes time \( O(|E| \log |E|) \) and space \( O(|E|) \). Note that the values to be sorted are bounded by \( \pm |B| \); if \( |B| = O(|E|^{\log |E|}) \), this time can be improved to \( O(|E| \log |E| |B|) = O(|E|^{\frac{\log |B|}{\log |E|}}) \) by using a radix sort.
3.3 Optimization. The algorithm next finds the optimal matching for each spanning tree of the reduced network. The set of all spanning trees can be constructed by considering all subsets of edges in $E^*$ of size $|V^*| - 1$ and checking whether they form trees. Alternatively, we can use the Matrix Tree Theorem to guide the construction of the spanning trees.

Any spanning tree omits at most one edge (a cut) from each chain in $C$. The uncut chains form a spanning tree in $N^*$. Conversely, given a spanning tree $T^*$ in $N^*$, we wish to determine how to break the chains in $N^* \setminus T^*$ so that the resulting tree has minimum matching value (over all such breaks). Consider any matching $\mathcal{P}$. If each matching path in $\mathcal{P}$ is broken at the vertices $V^*$ of the reduced graph, two types of subpaths are produced: 1) subpaths which begin and/or end within a chain, and 2) subpaths which traverse an entire chain. These latter subpaths must traverse chains in $T^*$. When paths of the second type are removed, some edge on each uncut chain must be exposed (perhaps an artificial 0 weight edge). Such an edge will be referred to as a pseudocut. Cuts and pseudocuts are illustrated in Figure 3.2. The cuts are edges $e_1$ and $e_3$, whose removal produces the tree $T$. Edge $e_2$ is a pseudocut. Notice that the only matching path which traverses $e_2$ traverses the entire chain.

![Figure 3.2](image)

Fix a set of cuts, determining a tree $T$, and pseudocuts having local chain parameters $\{K^*_s\}$. All subpaths of type 1) lie within the chains entirely to one side of cuts and pseudocuts. Then the contribution of type 1) subpaths to the matching value is counted in the $w(K^*_s)$ for a total of $\sum_{e^* \in E^*} w^*(K^*_s)$. The contribution of each type 2) subpath on $e^*$ is $\text{len}(e^*)$. Let $e^*$ be an uncut chain and let $e = xy$ be the pseudocut for $e^*$. Then
the number of (type 2) subpaths traversing $e^*$ is the same as the number of matching paths crossing $e$ which is $|b(T_x)|$ (as for trees). Thus, the total contribution of subpaths traversing all of $e^*$ is $\text{len}(e^*)|b(T_x)|$.

For each reduced vertex $v^*$ define $K(v^*)$ to be the blue excess over all vertices in the 
\textit{star} at $v^*$, namely the chains incident with $v^*$ truncated at the cuts and pseudocuts. That is,

$$K(v^*) = \sum_{e^* = v^* w^*} K_{e^*} + \sum_{e^* = w^* v^*} K_{e^*}$$

$$= \sum_{e^* = v^* w^*} K_{e^*}^v + \sum_{e^* = w^* v^*} (b(e^*) - K_{e^*}^w).$$

Then $b(T_x) = \sum_{v^* \in T_x} K(v^*)$. Thus, the total weight of the matching, given cuts and pseudocuts with local parameters $K_s$ is

$$W(K_s) = \sum_{e^* \in E^*} w(e^*) (K_{e^*}^v)$$

$$+ \sum_{e^* \in T^*} \text{len}(e^*) |\sum_{v^* \in T_x} K(v^*)|,$$

where in the second sum, the pseudocut associated with $e^*$ is $e = xy$. In Figure 3.3, the short crossing lines represent the cuts and pseudocuts and the dotted circles represent the elements of $B$ and $R$ counted by $K(v^*)$. Then the circles around all reduced vertices to the left of $e$ account for all blue and red elements in the left subtree determined by $e$. Note: non-optimal choices of pseudocuts may produce matching paths which are not the shortest distance paths for $T$ (the spanning tree fixed by the cuts). However, the optimum choice is always correct.

![Figure 3.3](image)

To optimize $W(K_S)$ by determining the optimum placement of cuts (and pseudocuts), we replace each absolute value $|\sum K(v^*)|$ by $\sum K(v^*)$ and $-\sum K(v^*)$ in all possible
combinations (2^|T^*| of them). Then, \( W(K_s) \) becomes

\[
F(K_s) = \sum_{e^*} f^{e^*}(K_s^{e^*}), \quad \text{where}
\]

\[
f^{e^*}(K_s^{e^*}) = (w^{e^*}(K_s^{e^*}) + d^{e^*} \cdot K_s^{e^*}),
\]

the \( d^{e^*} \) being real numbers depending on the choice of resolutions of the absolute values. We will minimize this function over the integers subject to constraints

\[
A \cdot K_s \leq b
\]

given by the sign assumptions from the resolution of the absolute values and by the range of allowable values for the local parameters \( K_s \). Note that the entries in the \(|T^*| \times |E^*|\) matrix \( A \) are all \(-1, 0, \) or \(1\). Note also that the functions \( w^{e^*}(K_s^{e^*}) + d^{e^*} K_s^{e^*} \) are concave upwards, and remain so even when defined for non-integers by linear interpolation. Thus, \( F(K_s) \) is concave upwards as a function from \( \mathbb{R}^{|E^*|} \rightarrow \mathbb{R} \).

We solve this optimization problem in two steps: first, we find a real optimum; then we examine all integer points close to the real optimum. We will show that one of these integer points must necessarily be the integer optimum (infeasibility is also detected here).

If the unconstrained or global optimum fails to lie within the constraint polytope, then the constrained optimum must lie on the boundary since the function decreases monotonically toward the global optimum. By repeating this observation for the lower dimensional faces of the polytope, we see that the constrained optimum is the global optimum on the affine hull of one of the faces. Thus we need only find all such optima for these affine hulls.

In order to find the global optimum on a hyperplane, given that this optimum lies within known bounds, we use the following observations (\( l \) being the dimension of the hyperplane):

1) If the optimum is known to lie in a box containing at most one integral point, then the optimum can be found in a time which is a function depending on \(|E^*|\) by linear programming on the piecewise linear functions in the 2^{|E^*|} unit boxes intersecting the bounding box.

2) Otherwise, a large dimension can be halved and the optimum \( x \) found on the bisecting hyperplane \( H \) (recursively). Then the global optimum must lie on the side of \( H \) indicated by the decreasing direction from \( x \) orthogonal to \( H \). This leads to the following recursion (\( \hat{b}_i \) means that the element \( b_i \) is to be deleted from the sequence):

\[
T(b_1, \ldots, b_l) \leq T(b_1, \ldots, \hat{b}_i, \ldots, b_l) + S(|E^*|) + T(b_1, \ldots, b_i - 1, \ldots, b_l),
\]

where \( k \) is the dimension of the hyperplane, \( 1 \leq l' \leq l \), \( T(b_1, \ldots, b_l) \) is the time required for a bounding box of size \( 2^{b_1} \times \cdots \times 2^{b_l} \), and \( S(|E^*|) \) is the time
required to solve 1). This leads to a time bound of

\[ T(b_1, \ldots, b_i) \leq S(|E^*|) \left( 2 \prod_{i=1}^{l} (b_i + 1) - 1 \right). \]

To show that the integer optimum must lie close to the real optimum, we use the following Theorem:

**Theorem (see [Sc80]).** Let \( Ax \leq b \) define a polytope \( P \), where \( A \) is an integral \( m \times n \) matrix. Suppose that there is an integral point in \( P \). Then for any real point \( x \in P \), there exists an integral point \( y \in P \), with \( \|x - y\|_\infty \leq n\Delta \), where \( \Delta \) is the maximum subdeterminant of \( A \).

Here, \( \|\cdots\|_\infty \) is the maximum of the magnitudes of the coordinates of \( \cdots \).

**Lemma 3.2.** Let \( y \) be an integer optimum and \( x \) be a real optimum to the convex programming problem \((*)\). Then there exists an integer optimum \( y' \) with \( \|x - y'\|_\infty \leq \Delta \cdot |E^*| \).

**Proof.** Consider the rectilinear box \( B \) determined by \( x \) and \( y \). Let \( c \cdot x \) be the linear function defined on \( B \) by interpolating each \( f^* \) between its value at \( x \) and its value at \( y \). Then by the Theorem, there is an integer minimum \( y' \) for \( c \) so that \( \|y' - x\|_\infty \leq \Delta \cdot |E^*| \).

By convexity, \( F(y') \leq c \cdot y' \leq c \cdot y = F(y) \). Thus, \( y' \) is an integer optimum to \((*)\) close to \( x \).

Thus, to find the integer optimum, we examine all integral points \( y \) with \( \|y - x\|_\infty \leq \Delta \cdot |E^*| \). Since the entries in \( A \) are all \( 0, \pm 1 \), we get the bound \( \Delta \leq n^{n/2} \) by volume bounds and Hadamard matrices.

**3.4 Complexity.** The time required to compute each phase of the algorithm is summarized below:

- \( T_1 = |E| \) to compute \( N^* \).
- \( T_2 = |E| \log |E| \) to compute local chain functions.
- \( T_3 = (|E^*|) \) to enumerate all spanning trees.
- \( T_4 = |E^*| \) to determine the function \( W(K_s) \) for each spanning tree.
- \( T_5 = 2|V^*|^{-1} \) to enumerate the resolutions of absolute values in \( W(K_S) \).
- \( T_6 = |E^*| \) to determine the function \( F(K_s) \) for each resolution.

\[ S(|E^*|) = 2|E^*| |E^*|^{3.5} (|V^*||E^*| + |E^*| \log |B|)^2 \]
for solving each "small" linear program ([Ka84]).

\[ T_7 = T(b_1, \ldots , b_t) \leq 2 \cdot S(|E^*|) \cdot (\log |B|)^{|V^*| - 1} \]

for the real convex optimization.

\[ T_8 = (\Delta \cdot |E^*|)^{|V^*|} \leq |V^*|(|V^*|^2/2) |E^*| |V^*| \]

for local search for integer optimization.

\[ T_9 = |V| \]

to compute minimum matching in optimum tree.

Collecting all of these terms and factors, we find that the time to compute \( M(N, B, R) \) is

\[
O(T_1 + T_2 + T_3 \cdot (T_4 + T_5 \cdot (T_6 + T_7 + T_8)) + T_9) \\
= O(|E| \log |E| + (\log |B|)^{|V^*| + 1} \cdot f(|V^*|, |E^*|))
\]

for some function \( f(|V^*|, |E^*|) \), and the space required is \( O(|E|) \).

This time bound can be improved by the following observations. Suppose that \( N = (V, E, s) \) is biconnected. Then in particular, \( N \) has no vertices of degree 1. By the construction of the reduced network via chains of maximal length, each vertex \( v^* \) in \( N^* \) has degree at least 3. Thus, \( 3|V^*| \leq 2|E^*| \) and so (for \( k = k(N^*) = |E| - |V| + 1 \))

\[
|V^*| \leq 2(k - 1) \quad \text{and} \\
|E^*| \leq 3(k - 1),
\]

so that the size of the reduced network is linear in the parameter \( k \). Consequently, the time bound reduces to

\[
O(|N| \log |N| + (\log |B|)^{2k-1} \cdot f(k)) \quad (**)
\]

or if \( |B| = O(|N|^\log |N|) \),

\[
O(|N| \frac{\log |B|}{\log |N|} + (\log |B|)^{2k-1} \cdot f(k)) \quad (**)
\]

for \( f \) some function of \( k \).

In order to improve the running time for graphs which are not biconnected, we collapse each biconnected component to a single vertex. The resulting graph is now a tree. We solve the problem on this tree to determine matching paths which extend between biconnected components, all of which requires linear time. Finally, we solve the remaining problems on the biconnected components (the blue sets for a biconnected component now are its original blue elements, together with entry points of matching paths constructed in the tree; the red sets are similar). Thus, the time bound \((**\)) applies to general graphs as well.
4. The Non-Bipartite Algorithm. Let \( N = (V, E, s) \) be a graph, \( B \) a set of vertices with \(|B|\) even. We wish to find a matching \( P = \{ [b_1, b_2] \} \) which minimizes the total path length. Non-bipartite matching turns out to be simpler and faster than bipartite matching. Notice that \( B \) is no longer a multiset — multiple elements at a single vertex may simply be canceled two at a time without affecting the minimum value.

Several ideas and results carry over from the bipartite case, particularly Theorem 2 (there exists a spanning subtree with the same minimum matching weight), finding minimum matchings in trees, as well as the observations regarding biconnectivity. Hence, we will assume that \( N \) is biconnected.

Again, our algorithm proceeds with the following major steps:

I. Compute a chain decomposition \( C \) of \( N \) and the reduced network \( N^* \).

II. For each chain in \( C \), compute a local chain function.

III. For each spanning tree \( T^* \) of \( N^* \), compute the minimum matching value \( M(T, A, B) \) given that the edges in \( N - T \) do not lie in the chains corresponding to edges of \( T^* \) (using the local chain functions computed in II).

IV. Using the optimum tree, compute a minimum matching.

The excesses \( K_s^* \) are now all either 0 or 1 since the tree algorithm matches elements within a chain as completely as possible. As a result, the local chain functions can now be computed in \( O(|E|) \) time; no sort is required. Given a spanning tree \( T^* \) in \( N^* \) and cuts and pseudocuts compatible with \( T^* \), the matching value function is

\[
W(K_s) = \sum_{e_s^* \in E^*} w^*(K_s^{e_s^*}) + \sum_{e_s^* \in T^*} \text{len}(e^*) \cdot \left[ \sum_{v^* \in T^*} K(v^*) \mod 2 \right].
\]

This can now be optimized by exhaustive search through all \( 2^{|E^*|} \leq 2^{3k-3} \) possible assignments of 0 or 1 to the \( K_s^{e_s^*} \). Then the total time required by the algorithm is

\[ O(|N| + f(k)) \]

for some function \( f \) of \( k \), and the space required is \( O(|N|) \).

5. The Parallel Implementation. This is a rare example of a minimum weight matching problem (in a complete finitely weighted graph) which is solvable quickly and deterministically in parallel, for small \( k(N) \). The only parts of the algorithm which take more than polylogarithmic time sequentially are the biconnected component decomposition, which is solvable in time \( O(\log |N|) \) with \( O(|N|) \) processors by Tarjan and Vishkin ([TV84]), and the construction of the reduced network and local chain functions, both of which may be solved in time \( O(\log |N|) \) with \( O(|N|) \) processors using standard techniques, among which sorting ([AKS83]) appears only in the bipartite case.
REFERENCES


