COMBINATORICA 6 (2) (1986) 123—150

AN AUGMENTING PATH ALGORITHM FOR LINEAR MATROID PARITY

H. N. GABOW and M. STALLMANN

Received 15 March 1985
Revised 28 January 1986

Linear matroid parity generalizes matroid intersection and graph matching (and hence network flow, degree-constrained subgraphs, etc.). A polynomial algorithm was given by Lovász. This paper presents an algorithm that uses time $O(mn^3)$, where m is the number of elements and n is the rank. (The time is $O(mn^{2.5})$ using fast matrix multiplication; both bounds assume the uniform cost model). For graphic matroids the time is $O(mn^2)$. The algorithm is based on the method of augmenting paths used in the algorithms for all subcases of the problem.

1. Introduction

Let **M** be a matroid over a set of elements E that is partitioned into blocks of size two, called *(parity) pairs*. Thus each element e has a *mate*, denoted $\bar{e}$, such that $\{e, \bar{e}\}$ is a parity pair; clearly $\bar{\bar{e}}=e$. A *matching* M is an independent set consisting of parity pairs, i.e., $e\in M$ if and only if $\bar{e}\in M$. A *maximum matching* has the greatest number of pairs possible. The *matroid parity problem* [15] is to find a maximum matching. An important special case is the *spanning tree parity problem* [12]: Given an undirected graph with the edges partitioned into pairs, find a forest containing as many pairs as possible.

Matroid parity generalizes many well-solved problems in combinatorial optimization, in particular, matroid intersection and matching on a general graph [16]. (Recall that matroid intersection solves integral network flow, and graph matching solves the degree-constrained subgraph problem.) Applications include spanning tree parity, certain processor scheduling problems [8], and finding the pinning number of a graph [19]. Lovász [17] and Jensen and Korte [13] independently showed that the general parity problem requires exponential time, if the matroid is given by an independence oracle (the usual assumption). Lovász [17] also found a polynomial-time algorithm for parity on linear matroids, a class that includes most applications. This was an important breakthrough. But as one might expect for such a complicated problem, the algorithm is involved, its running time is a polynomial of high degree, and it does not appear to generalize to other problems, most notably the parity problem where elements have numerical weights.

First author was supported in part by the National Science Foundation under grants MCS 78-18909, MCS-8302648, and DCR-8511991. The research was done while the second author was at the University of Denver and at the University of Colorado at Boulder.

AMS subject classification (1980): 05 B 35, 68 C 05

More recent work has shown that versions of the parity problem on subclasses of linear matroids, such as gammoids, reduce to graph matching or the degree-constrained subgraph problem [8, 22]. Orlin and Vande Vate [21] give an algorithm for linear matroid parity based on duality theory.

This paper presents an augmenting path algorithm for linear matroid parity. The algorithm can be viewed as a direct generalization of the algorithms for all subcases of the parity problem, e.g., matroid intersection and graph matching. The time is $O(mn^3)$, where $m=|E|$ is the number of elements and n is the rank of the matroid. If fast matrix multiplication is used the time is $O(mn^{2.5})$. (These bounds are in the uniform cost model, where each operation is one time unit [2]. This is realistic for finite fields. However the algorithm does n matrix inversions, each on a matrix whose entries are given input numbers. So for matroids over **R** the time bounds should be multiplied by L, the length of the input). The time is $O(mn^2)$ for a naive implementation on graphic and cographic matroids (i.e., spanning tree parity). This is as efficient as the classic algorithms for cardinality matroid intersection problems on the same classes of matroids [16]. In fact when the input represents a matroid intersection problem, the algorithm reduces to the intersection algorithm of [16]. Similarly for general graph matching the algorithm reduces to that of [7]. A preliminary version of the algorithm for binary matroids was given in Stallmann's doctoral dissertation [23]. The algorithm for graphic matroids (running in time $O(mn^2)$ and space $O(m)$) has been implemented in Pascal on a VAX 11/780. We anticipate that, like its special cases, the algorithm generalizes to the weighted parity problem.

The generalization of augmenting paths to matroid parity is discussed in Section 2. Section 3 presents the algorithm. Section 4 proves its correctness and Section 5 analyzes the time and space requirements. This section closes with some notation and definitions.

If A and B are sets, $A \oplus B$ denotes their *symmetric difference*. If e is an element, $A+e$ is equivalent to $A \cup \{e\}$, and $A-e$ is $A-\{e\}$. If P and Q are paths in a graph, PQ is the *concatenation* of P and Q, and P^r is the *reverse* of P. If P is simple and passes through vertices v and w, in that order, $P(v, w)$ is the *subpath* from v to w. Paths are also interpreted as sets of vertices or edges, depending on the context.

For the basic concepts of matroids, e.g., *independence, base, circuit,* see [1, 16, 26, 27]. If A is a set of elements, its *rank* is $r(A)$ and its *span* is $sp(A)$. The *fundamental circuit* of element x in base B is denoted $C(x, B)$. For an independent set A, (e,f) is a *swap* if $f \in A$ and $sp(A+e-f)=sp(A)$ [6, 10]. A matroid **M** with elements E is *linear* if there is a vector space V (over a field F) and a mapping of E into V that preserves rank. We usually identify elements of a linear matroid with their corresponding vectors. For $F=GF(2)$, **M** is *binary*. A special case is a *graphic* matroid, where the elements correspond to the edges of a graph and the independent sets are the forests.

2. Augmenting paths

This section introduces the basic ideas of the algorithm, including augmenting paths and transforms. Facts about matroids that are used in the correctness proof are also given. The development is interpreted as a generalization of the two important special cases of matroid parity — matroid intersection and general graph matching. Recall that matroid intersection is the case of the parity problem where the matroid

$\mathbf{M}$ is the direct sum of matroids $\mathbf{M}_i$ on elements E_i, $i=0, 1$, and every parity pair contains one element from each matroid. General graph matching (on a graph G) can be viewed as a graphic matroid parity problem (on a multigraph H): A vertex v of G corresponds to two vertices v_0, v_1 of H. An edge vw of G corresponds to a parity pair consisting of edges v_0v_1, w_0w_1. (Each edge incident to v has its own copy of v_0v_1 in H).

Algorithms for matroid intersection and graph matching use the method of augmenting paths. An augmenting path adds an element to the partial solution; if this violates feasibility it deletes an element to restore feasibility and repeats the process. Eventually the added element preserves feasibility and the partial solution has been enlarged. This can be done for any matroid, in the following sense.

Definition 2.1. An *augmenting sequence for a matching* M consists of distinct pairs $a_i, \bar{a}_i \notin M$, $0 \leqq i \leqq k$, $b_i, \bar{b}_i \in M$, $0 < i \leqq k$, such that the following sets have the same span:

$$M(a_0) = M + \bar{a}_k + a_0,$$

$$M(b_i) = M(a_{i-1}) + \bar{a}_{i-1} - b_i, \quad (0 < i \leqq k),$$

$$M(a_i) = M(b_i) - \bar{b}_i + a_i, \quad (0 < i \leqq k).$$

Note that $M(a_k)$ is a matching of rank $|M|+2$.

Theorem 2.1. *A nonmaximum matching M has an augmenting sequence.*

Proof. First observe that there is a matching N with $|N|=|M|+2$, $sp(M) \subseteq sp(N)$: Let N be a matching with $|N|=|M|+2$ and $|M \oplus N|$ minimum. Suppose $sp(M) \not\subseteq sp(N)$ and derive a contradiction as follows. Choose $a \in M$ with $N+a$ independent. Choose $b \in N-M$ with $N+a+\bar{a}-b$ independent. Then $N+a+\bar{a}-b-\bar{b}$ is a matching that contradicts the minimality of N. We conclude that N is the desired matching.

Now choose e, $f \in N$ so that $M+e+f$ is independent, i.e., $sp(M+e+f) = sp(N)$. Let $a_0=e$, $\bar{a}_k=f$. Assume that a sequence $a_0, \bar{a}_0, b_1, \bar{b}_1, \ldots, a_i$ has been defined, with $a_j, \bar{a}_j \in N-M$, $b_j, \bar{b}_j \in M-N$ and $sp(M(a_j)) = sp(M(b_j)) = sp(N)$. Note this implies $M(a_i)$ is independent. If $\bar{a}_i = f$, setting $i=k$ gives the desired sequence. Otherwise choose $b_{i+1} \in M-N$ so that $M(b_{i+1}) = M(a_i) + \bar{a}_i - b_{i+1}$ is independent. Thus $sp(M(b_{i+1})) = sp(N)$. Next choose $a_{i+1} \in N-M$ so that $M(a_{i+1}) = M(b_{i+1}) - \bar{b}_{i+1} + a_{i+1}$ is independent. Thus $sp(M(a_{i+1})) = sp(N)$. Repeat this process until it halts with the desired sequence. (It eventually halts because each swap replaces an element of $M-N$ by an element of $N-M$.) ∎

It is sometimes possible to drop the "destination" element $\bar{a}_k$ from the span-conserving conditions, i.e., define $M(a_0)$ as $M+a_0$. This type of augmenting sequence is used for matroid intersection and graph matching. The above proof is easily modified for these special cases: In matroid intersection $\bar{a}_k$ is unnecessary because all swaps in $\bar{a}_k$'s matroid $\mathbf{M}_i$, $i=0$ or 1, preserve M's span in E_i. In graph matching span amounts to the vertices covered by the matching, which of course is independent of the destination vertex $\bar{a}_k$. On the other hand the destination element $\bar{a}_k$ is needed for more general problems like graphic matroid parity: Figure 2.1 shows a graphic matroid with a matching (forest) consisting of wavy edges. The augmenting sequences

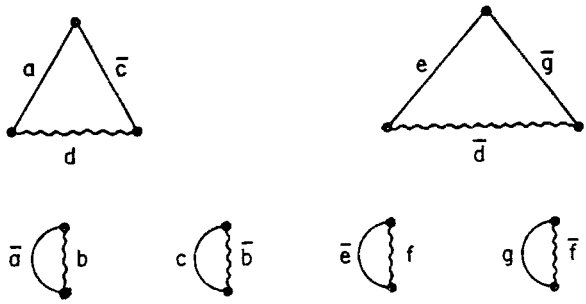


Fig. 2.1. Destination elements are needed in augmenting sequences

are $a\bar{a}b\bar{b}c\bar{c}d\bar{d}e\bar{e}f\bar{f}g\bar{g}$ and its symmetric variants. This sequence preserves the span $sp(M+a+\bar{g})$. However if $\bar{g}$ is dropped span is not preserved — $sp(M+a)\neq sp(M(e))$.

The need for destination elements makes it convenient to work with bases rather than matchings. Given a parity problem on a matroid $\mathbf{M}$, let B be a base. Form a new matroid $\mathbf{M}'$ by adding an element e' parallel to e for each $e\in B$; e' has no mate and is called a *singleton*. If M is a matching (in $\mathbf{M}$), M^* denotes a base (in $\mathbf{M}'$) formed by adding appropriate singletons to M. The parity problem becomes finding a base with as many matched pairs e, $\bar{e}$ as possible.

Definition 2.2. An *augmenting sequence for a base* M^* consists of distinct pairs b_i, $\bar{b}_i\in M$, $0<i\leqq k$, singletons $\bar{b}_0$, $b_{k+1}\in M^*$, and pairs a_i, $\bar{a}_i\notin M$, $0\leqq i\leqq k$, such that the following sets are bases:

$$M^*(a_0) = M^* - \bar{b}_0 + a_0,$$

$$M^*(b_i) = M^*(a_{i-1}) + \bar{a}_{i-1} - b_i, \quad (0 < i \leqq k+1),$$

$$M^*(a_i) = M^*(b_i) - \bar{b}_i + a_i, \quad (0 < i \leqq k).$$

$M^*(b_{k+1})$ is a base with one more matched pair than M^*. If M^* does not have as many pairs as possible it has an augmenting sequence (by an argument similar to Theorem 2.1). The algorithm of Section 3 works by repeatedly finding an augmenting sequence for the current base.

To find an augmenting sequence it is convenient to use a graphic representation of the matroid, as in the matroid intersection algorithm [16]:

Definition 2.3. The *dependence graph* $G(B)$ of a base B is a bipartite graph with vertex sets $(E-B, B)$ and edges ab that are swaps, i.e., $B+a-b$ is a base.

Figure 2.2 shows a graphic matroid with dependence graph in Figure 2.3(a). In Figure 2.2 wavy edges are parity pairs of the base; dashed edges are singletons of the base. In Figure 2.3 basic elements are on the right, nonbasic on the left.

For a linear matroid, let B be a base. For any element a denote its representation as a linear combination of elements of B as

$$a = \sum_{b\in B} c(a, b)\, b.$$

Here $c(a, b)$ are scalars and ab is an edge of $G(B)$ if and only if $c(a, b)\neq 0$.

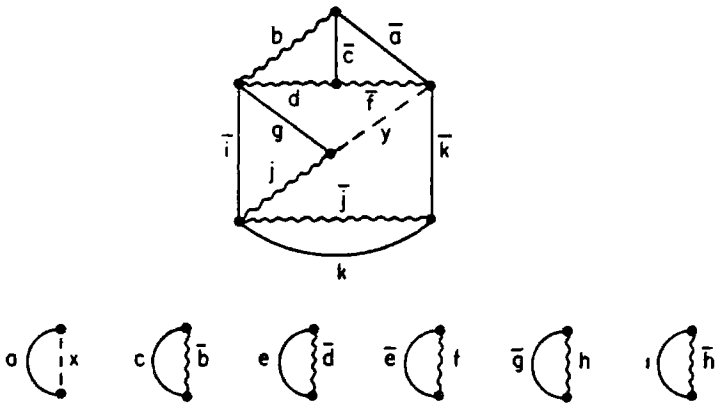


Fig. 2.2. No augmenting path is an augmenting sequence

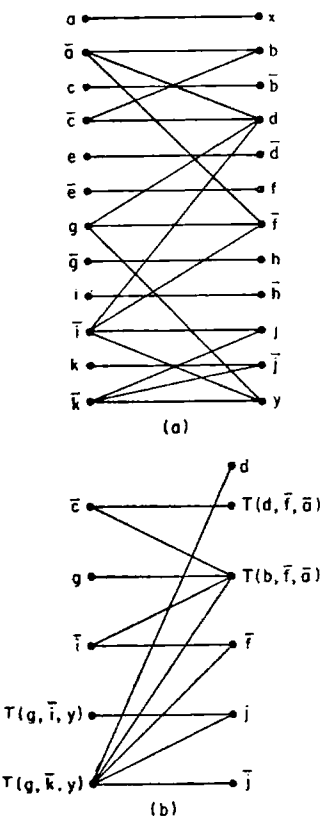


Fig. 2.3.(a) Dependence graph for Fig. 2.2
(b) Modifications in dependence graph for transforms

In a binary matroid a dependence graph completely determines the matroid. A special case of this is graphic matroids. (The dependence graph of a graphic matroid determines the matroid but not the graph itself, since nonisomorphic graphs can have isomorphic matroids [26].) The opposite extreme from binary matroids are the Vamos—Higgs matroids and their generalizations as defined by Lovász [17]. Their dependence graph is essentially complete bipartite and gives no information about the matroid. The parity problem on these matroids requires exponential time [17].

A dependence graph specifies the swaps that are valid for a base. In an augmenting sequence each swap that is done can change the valid swaps. This could conceivably force an algorithm to recompute the dependence graph in its search for an augmenting sequence. Recomputing the dependence graph is unnecessary in graph matching — a swap corresponds to two incident edges, regardless of the base. Recomputing is unnecessary in matroid intersection — a shortest augmenting sequence never changes the relevant portions of the dependence graph. However in linear matroid parity, even on binary or graphic matroids, some modification of the dependence graph seems necessary. Our algorithm adds nodes to the dependence graph, making it the union of dependence graphs for various bases. We now make these ideas more precise.

Definition 2.4. Consider a dependence graph $G(B)$.

(i) A *parity path* is a finite sequence $\bar{b}_0 a_0 \bar{a}_0 b_1 \bar{b}_1 \ldots a_{i-1}\bar{a}_{i-1} b_i \bar{b}_i \ldots$, where $\bar{b}_0$ is a singleton of B, and $\bar{a}_{i-1}b_i$, $a_i\bar{b}_i$ are all edges of $G(B)$. (The last element of the path may be a_r, $\bar{a}_r$, b_r or $\bar{b}_r$, for some r.) The reverse of such a path is also a parity path.

(ii) A set of edges $\{a_i b_i | b_i \in B,\ 1 \leqq i \leqq r\}$ in $G(B)$ is *feasible if* $B \cup \cup \{a_i | 1 \leqq i \leqq r\} - \{b_i | 1 \leqq i \leqq r\}$ is a base.

(iii) An *augmenting path* is a feasible parity path $\bar{b}_0 \ldots b_k$ ending at a singleton b_k, i.e., $B - \bar{b}_0 + a_0 + \bar{a}_0 - b_1 - \bar{b}_1 \ldots - b_k$ is a base.

Note that a parity path is a path in the graph $G(B)$ plus all "parity" edges $e\bar{e}$. It is convenient *not* to include the pairs $e\bar{e}$ as edges in the dependence graph. So throughout this paper, the term "edge" refers to an edge in the dependence graph, which represents a swap.

The algorithm modifies the original dependence graph by adding nodes to it (on both the nonbasic and basic side). Define a *generalized dependence graph* for a base B as a bipartite graph containing $G(B)$ as a subgraph, such that aside from the singletons of B, every node has a mate (on the same side). The notion of parity path makes sense in generalized dependence graphs, and we use it in that context.

Figure 2.2 has an augmenting path

$$P = x a \bar{a} b \bar{b} c \bar{c} d \bar{d} e \bar{e} f \bar{f} g \bar{g} h \bar{h} i \bar{i} j \bar{j} k \bar{k} y.$$

P is not an augmenting sequence because $M(d)$ contains the cycle $\overline{acf}$. In this example no augmenting path is also an augmenting sequence. However an augmenting sequence can be derived from P by reversing the subpath $d\bar{d}e\bar{e}f\bar{f}$: $x a \bar{a} b \bar{b} c \bar{c} f \bar{f} e \bar{e} d \bar{d} g \bar{g} h \bar{h} i \bar{i} j \bar{j} k \bar{k} y$ is an augmenting sequence. In general an augmenting sequence can be derived from an augmenting path by a number of reversals of (nested) subpaths.

The need to reverse subpaths comes about as follows. Traversing edge $\bar{a}b$ changes $\bar{c}$'s dependence graph adjacencies, adding $\bar{f}$ and removing d. Usually such a change can be ignored — edge $\overline{af}$ allows $\bar{f}$ to be reached from $\bar{a}$ directly, so it is unnecessary to use the new edge $\overline{cf}$ to reach $\bar{f}$. However in Figure 2.2 the goal is to reach g, and the direct path to g, $xa\overline{af}f\bar{e}e\bar{d}dg$, is infeasible because it introduces the cycle $gy\bar{a}b$. This cycle is introduced when $\overline{af}$ is traversed, thereby removing g from d's adjacencies. Clearly the algorithm needs a mechanism to compute d's new adjacencies.

In Section 3 we call the subpath $\bar{f}f\bar{e}e\bar{d}d$ a "blossom", and $\bar{f}, d$ its "tips". Element g is adjacent to both tips. When a path traverses a blossom, entering one tip and leaving the other, it can exit only to elements that are not adjacent to all tips. (Actually this is the case for binary matroids. Other exits are possible in general linear matroids — see Lemma 2.1.) To model this the algorithm introduces a new element called a "transform" that is adjacent to precisely the valid exits of the blossom.

The blossom tips $\bar{f}$, d above are elements in the matching. Similar reasoning shows that transforms for unmatched blossom tips are needed: Figure 2.2 has a blossom $\bar{l}i\bar{h}h\bar{g}g$ that makes the direct path from y to $\bar{f}$, $y\bar{l}i\bar{h}h\bar{g}g\bar{f}$, infeasible. The tips l, g are unmatched and need a transform to model the valid exits.

A path always enters a blossom through an element called the "bud". The bud is matched if and only if the tips are not.

Now we show how transforms are constructed in a linear matroid. In Lemma 2.1 (i) below, a_0 and a_1 play the role of unmatched tips and b plays the role of the matched bud. The transform $T(a_0, a_1, b)$ is adjacent to exactly the valid exits from the blossom, i.e., the elements that, along with the bud, can be replaced in the matching by the tips. Lemma 2.1 (ii) treats the case of matched tips.

Let $\mathbf{C}$ be a matrix whose rows and columns are indexed by different sets. The *two-rowed minor* for rows $e_i, i=0, 1$ and columns $f_i, i=0, 1$ is $c(e_0, f_0)c(e_1, f_1) - -c(e_0, f_1)c(e_1, f_0)$, and is denoted $c(e_0, e_1, f_0, f_1)$ or $c(f_0, f_1, e_0, e_1)$. In what follows B is a base of a vector space V, and $\mathbf{C}$ is the matrix whose columns correspond to elements $b \in B$, rows correspond to elements $a \in E-B$, and entries are $c(a, b)$ (as above, the coefficient of b in a's representation).

Lemma 2.1. *Consider elements* $a_0, a_1 \notin B$, $b_0, b_1 \in B$.

(i) *Suppose* a_0, a_1 *are both adjacent to* b_0. *Then* V *contains an element* $T(a_0, a_1, b_0)$ *adjacent to precisely the elements* b_1 *such that* $B+a_0-b_0+a_1-b_1$ *is a base.*

(ii) *Suppose* b_0, b_1 *are both adjacent to* a_0. *Then* V *contains a base* B' *whose dependence graph is isomorphic to that of* B, *except that* b_0 *is replaced by an element* $T(b_0, b_1, a_0)$ *that is adjacent to precisely the elements* a_1 *such that* $B+a_0-b_0+a_1-b_1$ *is a base.*

Proof. Write

$$a_i = \sum_{\beta \in B} c(a_i, \beta)\beta.$$

Thus

$$c(a_0, b_0)a_1 - c(a_1, b_0)a_0 = \sum_{\beta \in B} c(a_0, a_1, b_0, \beta)\beta. \tag{1}$$

Then $c(a_0, a_1, b_0, b_0)=0$ together with the hypothesis of (i) (or (ii)) implies that $B+a_0-b_0+a_1-b_1$ is a base exactly when $c(a_0, a_1, b_0, b_1)\neq 0$.

(i) Choosing $T(a_0, a_1, b_0)=c(a_0, b_0)a_1-c(a_1, b_0)a_0$ gives the desired adjacencies.

(ii) Define a vector $b=c(a_0, b_0)b_0/c(a_0, b_1)+b_1$. Thus $B'=B-b_1+b$ is a base, for which a_1's expansion is

$$(2)\qquad a_1 = c(a_1, b_1)b-c(a_0, a_1, b_0, b_1)b_0/c(a_0, b_1)+\sum_{\beta\in B-b_0, b_1} c(a_1, \beta)\beta.$$

The description of B's dependence graph given in the lemma follows if we let $b_0\in B'$ be $T(b_0, b_1, a_0)$ and in the isomorphism, b corresponds to b_1. ∎

Formulas (1)—(2) essentially define the transforms, by specifying their adjacencies. (Comment on terminology: "transform" refers to these formulas, viewed as linear transformations.) The algorithm uses (1)—(2) to create transforms. Note that in the construction of transforms, the only use of the third argument is to define two nonzero scalars — $c(a_i, b_0)$ in (1) and $c(a_0, b_i)$ in (2). We generalize the notion of transform in Lemma 2.1 by allowing the bud of a transform to be given by two nonzero scalars. Thus the bud need not be an element of the original matroid.

Now we state some easy consequences of (1)—(2). Assume transform $T(e_0, e_1, f_0)$ exists; the e_i are both either basic or nonbasic (and are not transforms); f_0 is given by scalars (of course if f_0 is an element, as in Lemma 2.1, the scalars are the appropriate coefficients). If e_i are basic, let $B(T(e_0, e_1, f_0))$ denote base $B-e_0+T(e_0, e_1, f_0)$ (as in Lemma 2.1). Saying that $T(e_0, e_1, f_0)$ is adjacent to f refers to adjacency with respect to $B(T(e_0, e_1, f_0))$.

Corollary 2.1. *$T(e_0, e_1, f_0)$ is adjacent to f_1 if and only if $c(e_0, e_1, f_0, f_1)\neq 0$.*

Proof. For e_i basic this is clear from (2). For e_i nonbasic it is clear from (1) if $f_1\in B$. If f_1 is a transform of basic elements, the analog of (1) for base $B(f_1)$ gives the desired conclusion. ∎

Corollary 2.2. *$T(e_0, e_1, f_0)$ is adjacent only to neighbors of e_0 or e_1. It is adjacent to any element that is adjacent to exactly one of e_0, e_1. It is not adjacent to f_0.* ∎

If bud f_0 is given by scalars, the last statement of this corollary means that the transform is not adjacent to any element whose coefficients for e_i are f_0's scalars. In a binary matroid the corollary can be strengthened: $T(e_0, e_1, f_0)$ is adjacent to precisely the elements adjacent to exactly one of e_0, e_1. The next fact shows that no information is lost in replacing a tip by a corresponding transform.

Corollary 2.3. *For a set C containing none of e_i or $T(e_0, e_1, f_0)$, $C+e_0+e_1$ is a base if and only if $C+e_1+T(e_0, e_1, f_0)$ is.* ∎

The last fact shows how these properties are propagated.

Corollary 2.4. *Let S be a sequence of all basic or all nonbasic elements. In the first case for element f define a vector $v(f)=(c(f, e)|e\in S)$; in the second case for f basic define vector $v(f)=(c(e, f)|e\in S)$. Let R be a tree whose nodes are labelled by $e\in S$ such that for any edge e_0e_1 of R, $T(e_0, e_1, f_0)$ exists. Then f_1 is not adjacent to any of these $|S|-1$ transforms if and only if $v(f_1)$ is a multiple of $v(f_0)$. (If bud f_0 is given by nonzero scalars $c(f_0, e)$ or $c(e, f_0)$, $v(f_0)$ is still defined as above).*

Proof. For $|S|=2$ this follows from Corollary 2.1. For $|S|>2$ use induction. ∎

Figure 2.3 (b) illustrates the construction of transforms for Figure 2.2. Note that $T(d, \bar{f}, \bar{a})$ models the legal exits from the blossom path. In the modified matroid the augmenting path is

$$P' = xa\bar{a}b\bar{b}c\bar{c}T(d, \bar{f}, \bar{a})\bar{d}e\bar{e}\bar{f}T(\bar{f}, b, \bar{a})T(g, \bar{k}, y)\bar{g}h\bar{h}iT(\bar{i}, g, y)j\bar{j}k\bar{k}y.$$

(This is the path found by the algorithm of Section 3.) P' is essentially the path P (given after Definition 2.4), but transforms make it both an augmenting path and an augmenting sequence.

This completes the motivation for the algorithm. Now we give some facts that form the basis of the correctness proof. First is a mechanism for proving that the augmenting path found by the algorithm is feasible. It generalizes a result of Krogdahl [14, 16] for matroid intersection. In Theorem 2.2, the term "matching" refers to an ordinary matching on a graph. Recall that an *alternating path* is a simple path whose edges are alternately matched and unmatched.

Theorem 2.2. *A matching of a dependence graph is feasible if it has no alternating cycle.*

Proof. Let M be a matching of $G(B)$. Suppose the edges of M are ordered, $M=\{m_i|1\leqq i\leqq k\}$, where $m_i=a_ib_i$. Let N_i be the subgraph of $G(B)$ induced on $\{a_j, b_j|1\leqq j\leqq i\}$. Call M *reducible* if, in some ordering of M, for all i in $1\leqq i\leqq k$, a_i or b_i is a degree one vertex in N_i. Observe that M is reducible if and only if it has no alternating cycle: The *only if* direction is clear. For the *if* direction, find the elements m_i in the order $i=k, \ldots, 1$, as follows: Inductively assume that $m_i, i=k, \ldots, j+1$ have been found. Choose m_j as an endpoint of a maximal length alternating path in the graph induced on the elements of $M-\{m_i|k\geqq i>j\}$. Such a path is not a cycle, by the hypothesis. So a_j or b_j has degree one in the graph, as desired.

Thus it suffices to show that the swaps of a reducible matching can be executed in order, i.e., for all $1\leqq i\leqq k$, $B_i=B+a_1-b_1\ldots+a_i-b_i$ is a base. This is done by induction, as follows. Assume B_{i-1} is a base. First suppose a_i has degree one in N_i. It is easy to see that $C(a_i, B_{i-1})=C(a_i, B)$. Hence a_ib_i is feasible for B_{i-1} and B_i is a base. Next suppose b_i has degree one in N_i. An easy induction shows that for any j, r, $1\leqq j\leqq r<i$, $b_i\notin C(a_r, B_j)$. Thus $b_i\in C(a_i, B_j)$ for $1\leqq j<i$, and a_ib_i is feasible for B_{i-1}. ∎

Simple examples show that the condition of the theorem is not necessary. The theorem can be used to show that a simple parity path is feasible, since the dependence graph edges of such a path form a matching. The theorem is applied in Section 4 to show that the paths constructed by the algorithm are feasible.

This section concludes with a mechanism for showing that the final matching is maximum. One approach is to use Theorem 2.1, but it is more convenient to use the Lovász duality theorem [18]. This result generalizes the duality theorems for matroid intersection and graph matching. Section 4 only uses the "easy" part of Lovász's result. To make this paper self-contained we state it and give a simple proof. A *parity partition* is a partition of the elements E of a matroid into sets of parity pairs $E_1, \ldots, E_k$ (thus $e\in E_i$ if and only if $\bar{e}\in E_i$).

Lemma 2.2. *In a (not necessarily linear) matroid, let M be a matching, $E_1, \ldots, E_k$*

a parity partition and A a set of elements. Then

$$\frac{|M|}{2} \leqq r(A) + \sum_{i=1}^{k} \left\lfloor \frac{r(E_i \cup A) - r(A)}{2} \right\rfloor.$$

Proof. It suffices to show that all but at most $r(A)$ pairs of M contribute one to the summation term. Without loss of generality assume A is independent, and add elements of M to A to get an independent set of $|M|$ elements. All but at most $r(A)$ pairs of M get both elements added to A. An added pair that is in E_i contributes two to $r(E_i \cup A)$, as desired. ∎

In a linear matroid the lemma holds if A is any subset of the vector space containing E. Lovász shows that in a linear matroid equality is achieved with the proper choice of M, A and E_i; clearly such an M is a maximum matching. Section 4 follows Lovász's approach: the matching found by the algorithm is proved to be maximum by exhibiting an appropriate parity partition. As a consequence Section 4 provides another proof of Lovász's duality theorem.

3. The algorithm

This section presents the algorithm, first intuitively and then formally. It starts with features related to matroid intersection [16] and graph matching [5, 20]- breadth-first search and blossoms. Then it discusses new features — most notably, transforms. This is followed by formal definitions. A detailed statement of the algorithm is in Figure 3.1. Figure 3.2 shows how the algorithm works on a simple example. The reader should refer to both figures throughout the discussion. It may also be helpful to refer to the very end of this section, which details how the algorithm works on Figure 3.2.

Input: a matching M and corresponding base M^*.

Output: a matching N and corresponding base N^* such that $|N| = |M| + 2$, if possible.

comment The following details of the algorithm are assumed rather than stated explicitly: When an element e gets labelled, the algorithm assigns the next highest serial number to $s(e)$ and puts e on the queue; if e was a blossom tip, all tips of its blossom are unmarked. A transform $T(t_0, t_1, b)$ is created using formulas (1)—(2) of Lemma 2.1. **end comment**

comment the main routine **end comment**

for $e \in E$ **do** $s(e) \leftarrow \infty$; $B(e) \leftarrow e$ **end do**
initialize the queue to empty
initialize G to $G(M^*)$
for e a singleton in M^* **do** label e by $(\emptyset, \emptyset)$ **end do**
$N^* \leftarrow \emptyset$

while the queue is not empty **and** $N^* = \emptyset$ **do**
    remove the first element e from the queue
    for f adjacent to e in G, $B(f) \neq B(e)$ **do**

comment choose f's in order of increasing serial number $s(f)$

end comment

if f is labelled **and** $s(f)<s(e)$ **then**
let b be the first element of $P(e)$ such that $P(f)$ contains an element in $B(b)$
if $b=\emptyset$ **then** Augment (e,f) **else** Blossom (e,f,b) **end if**

else if neither f nor $\bar{f}$ is labelled **and** $B(f)=f$ **then**
if $\bar{f}$ is adjacent to e in G **then**
comment degenerate blossom **end comment**
create $x=T(f,\bar{f},e)$ and add it to G
label x by $(e,\emptyset)$
update $B(f)$, $B(\bar{f})$ to the degenerate blossom, and mark $f,\bar{f}$ as tips
else label $\bar{f}$ by $(e,\emptyset)$ **end if**

end if
end do

end do

comment This ends the main routine. If $N^*\neq\emptyset$ then N is a larger matching, else M is maximum. **end comment**

procedure Blossom (e_0,e_1,b)

let b_i be the first element of $P(e_i)$ in $B(b)$, $i=0,1$
if $b_0=b_1$ **then** let t_i be the immediate predecessor of b in $P(e_i)$, $i=0,1$
else $t_i=\emptyset$, $i=0,1$

for g an unlabelled element, that is either not in a blossom or is a blossom tip, and is in either path $P(e_i,b_i)-t_i$ **do**
comment choose g's in order of decreasing $s(\bar{g})$ **end comment**
label g by (e_{1-i},e_i)

end do

if $t_0\neq\emptyset$ **then**
create transform $T(t_0,t_1,b)$ and add it to G
label it by (e_1,e_0)

end if

update $B(e)$ for e in the new blossom, and mark any new tips

end Blossom

procedure Augment (e,f)
$N^*\leftarrow M^*$
Path $(e,\emptyset)$
Path $(f,\emptyset)$

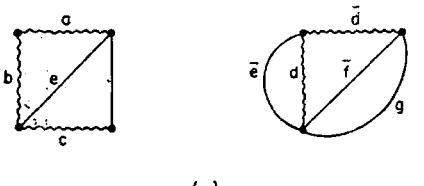
end Augment

```
procedure Path (start, finish)
    if start ≠ finish then
        if start is a transform T(x, y, z) then g ← x else g ← start end if
        N* ← N* ⊕ {g, ḡ} (N* − g if g is a singleton)
        (back, reverse) ← label (start)
        if reverse ≠ ∅ then Path (reverse, ḡ) end if
        Path (back, finish)
    end if
end Path
```

Fig. 3.1. The algorithm

An iteration of the algorithm finds an augmenting path and enlarges the matching. In what follows "the algorithm" refers to one such iteration.

The algorithm does (essentially) a breadth-first search to find a parity path joining two singletons in the dependence graph $G(M^*)$ (which gets enlarged). The search constructs a system of *search paths* $P(e)$. $P(e)$ is a parity path of even length from element e to a singleton. Search paths are defined by *back pointers:* if e has a back pointer to f then $P(e) = e\bar{e}P(f)$. (A complete definition is given below.) An element whose back pointer (and search path) is defined is called *labelled.*



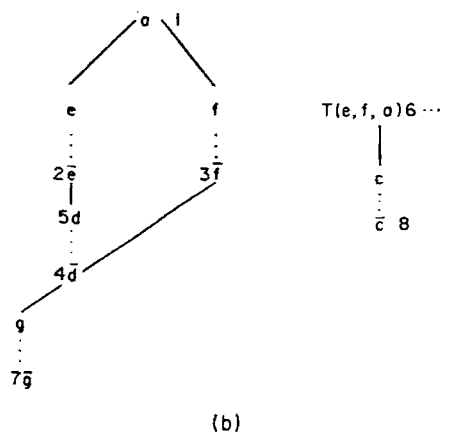


Fig. 3.2.(a) Example graphic parity problem
(b) Search paths grown by the algorithm

To do the breadth-first search the algorithm maintains a queue of labelled elements [25]. It *scans* a labelled element e by removing it from the queue and examining its dependence graph adjacencies. Suppose e is adjacent to $\bar{f}$. There are four possibilities:

(i) *neither f nor $\bar{f}$ is labelled.* Element f gets labelled with a back pointer to e, making $P(f)=f\bar{f}P(e)$. This is a *grow step*.

(ii) *$\bar{f}$ is not labelled but f is.* A search path $P(f)$ is already defined. There is no need for another so the algorithm does nothing.

(iii) *$\bar{f}$ is labelled and $P(e)$ is disjoint from $P(\bar{f})$.* The path $P(e)^r P(f)$ is a feasible parity path between two distinct singletons, i.e., an augmenting path. The algorithm augments the matching. This is an *augment step*.

(iv) *$\bar{f}$ is labelled and $P(e)$ and $P(\bar{f})$ intersect.* A *blossom step* is done. For convenience rename e and $\bar{f}$ as y_0 and z_0, so $y_0 z_0$ is an edge in the dependence graph and y_0 and z_0 are labelled. Let $P(y_0)=y_0\bar{y}_0 y_1\bar{y}_1 \ldots y_j\bar{y}_j x_1\bar{x}_1 \ldots x_i\bar{x}_i x_{i+1}$, $P(z_0)=$ $=z_0\bar{z}_0 z_1\bar{z}_1 \ldots z_k\bar{z}_k x_1\bar{x}_1 \ldots x_i\bar{x}_i x_{i+1}$. New search paths can be formed: $P(\bar{y}_s)=$ $=P(y_0, \bar{y}_s)^r P(z_0)$ (where $P(y_0, \bar{y}_s)$ is the prefix of $P(y_0)$ that ends at $\bar{y}_s$), and $P(\bar{z}_t)=$ $=P(z_0, \bar{z}_t)^r P(y_0)$. If $\bar{y}_s$ is currently unlabelled it gets a label defining the new search path. The label consists of a back pointer to z_0 and a *reverse pointer* to y_0. Similarly for $\bar{z}_t$.

A label is denoted *(back pointer, reverse pointer)*. Thus if e has the label (f, g) (assigned in a blossom step) then $P(e)=P(g, e)^r P(f)$. If e has the label $(f, \emptyset)$ (assigned in a grow step) then $P(e)=e\bar{e}P(f)$.

A *blossom* is the set $B=\{y_0, \bar{y}_0, \ldots, y_j, \bar{y}_j, z_0, \bar{z}_0, \ldots, z_k, \bar{z}_k\}$. The *bud* of B is x_1, the first element common to $P(y_0)$ and $P(z_0)$. The *tips* of B are $\bar{y}_j$ and $\bar{z}_k$, the immediate predecessors of the bud. (This description of blossoms, buds and tips is a simplification of the general case- see Definition 3.2.) In a blossom step all unlabelled elements of B get labelled except for the tips. (Comment on flowery terminology: The algorithm grows a blossom from its bud. The tips of a blossom are at the end, both in terms of how the blossom forms and how it is traversed. Some other aspects of buds and tips are not as botany would have it.)

For efficiency the algorithm maintains, for each element e, the identity $B(e)$ of the blossom containing e. (Before e enters a blossom, $B(e)=e$). When scanning an edge ef with $B(e)=B(f)$ there is no need to do a blossom step — e and f are already in the same blossom, which implies no new label can be generated.

The above scheme is similar to the one for graph matching in [7]. The matching literature calls the bud the "base" [5, 20] (or "tip" [4], "(mate of the) join" [7]). Viewing graph matching as a spanning tree parity problem, the tips of a blossom are edges parallel to the bud. This makes graph matching simpler. Now we discuss new aspects of linear matroid parity.

As mentioned in Section 2, if a tip t_0 were given a label, $P(t_0)$ would include the other tip t_1, and a subsequent grow step could produce an infeasible path. To avoid this the algorithm creates the transform $T(t_0, t_1, b)$, where b is the bud of the blossom, and labels it rather than the tips. The proposed path for t_0 is the reverse of the one for t_1. Since these paths are identical except for order, one can be chosen arbitrarily as the transform's search path.

For example consider Figure 3.2. The path $ae\bar{e}dd\bar{f}fb$ is infeasible. The tips are e, f and the bud is a, so the algorithm creates $T(e, f, a)$. This transform is not adjacent

to b but is adjacent to c, as the path $ae\bar{e}d\bar{d}\bar{f}fc$ is feasible. The choices for $P(T(e,f,a))$ are $T(e,f,a)\bar{e}d\bar{d}\bar{f}fa$ (the transform replaces e) and $T(e,f,a)\bar{f}d\bar{d}\bar{e}ea$ (it replaces f). The first interpretation is used.

The algorithm adds the transform to the dependence graph G. It does not remove the tips that the transform replaces from G. This is because either tip can occur in a search path and so can get labelled in a future blossom step. So G can have transforms for a number of different bases; it is a generalized dependence graph.

Transforms can cascade in the following sense. The tips of a transform are unlabelled, and so are elements of the given matroid $\mathbf{M}$. The bud is labelled, and can be a transform. A "first generation transform" forms from three elements in $\mathbf{M}$. A "second generation transform" forms from tips in $\mathbf{M}$ and a bud that is a first generation transform. In general a "k^{th} generation transform" forms from tips in $\mathbf{M}$ and a bud that is a $k-1^{\text{st}}$ generation transform. The algorithm adds each transform $T(e_0, e_1, f_0)$ to the dependence graph and to the corresponding coefficient matrix $\mathbf{C}$; the transform's coefficients in $\mathbf{C}$ are computed using formulas (1)—(2) of Lemma 2.1. For nonbasic transforms in the second and higher generations, formula (2) constructs the transform from a bud given by scalars, as mentioned after Lemma 2.1.

As in graph matching, blossoms nest — a blossom is composed of a number of elements and subblossoms. A consequence of nesting is that a blossom B can contain various unlabelled elements. All tips of B are unlabelled. The tips are still candidates for receiving labels. A subblossom C of B has one tip labelled (the label is assigned when C is absorbed into B). The other tips are unlabelled and are no longer candidates for labels. This remark applies recursively to subblossoms of C, etc. An *old tip* is an element that was a tip at some point but no longer is. Tips and old tips constrast with graph matching, where all elements of a blossom are labelled.

Now consider the special situation when an element e is scanned and found adjacent to two unlabelled elements $f, \bar{f}$ that form a pair. The obvious action — labelling f in a grow step — is incorrect: the dependence edge ef can make subsequent labels propagated from f infeasible. This situation actually represents the simplest of blossoms, with bud b, tips $f, \bar{f}$ and no other parity pairs. The algorithm does a *degenerate blossom step*, creating transform $T(f, \bar{f}, e)$ and labelling it with a back pointer to e. For uniformity we say that the degenerate blossom step is "for $f\bar{f}$".

Paths that include both blossom tips are not the only ones that can produce infeasibilities. For instance in Figure 3.2, $af\bar{f}\bar{d}dg$ is infeasible. This is because of a "hidden blossom" with tips $d, \bar{d}$. It is possible to detect all hidden blossoms and generate appropriate transforms. Note however that in Figure 3.2, $ae\bar{e}\bar{d}dg$ is feasible, because it does not traverse $\bar{f}$, the bud of the hidden blossom. Because of the order in which labels are generated, the feasible path is found first and becomes $P(\bar{g})$.

In a hidden blossom situation, a feasible path always exists as an alternative to the infeasible one. To ensure that feasible paths are generated first, *serial numbers* $s(e)$ are used to record the order that elements are labelled: the i^{th} element labelled has $s(e)=i$ ($s(e)=\infty$ if e is unlabelled). If $s(e)<s(f)$ then e was labelled before f and is scanned before f. The following rule ensures that all search paths are feasible: When edge ef is scanned, a blossom or augment step is done if and only if $B(e)\neq B(f)$ and these two conditions hold:

(i) $s(f)<s(e)$ (f has already been scanned);
(ii) $s(f)\leqq s(g)$ for any g adjacent to e with $B(g)\neq B(e)$.

(Note that (i) does not inadvertently rule out a blossom step for ef: If e is a transform with $s(f)<s(e)$ then e may not exist when f is scanned. In this case the blossom step must be done when e is scanned, which is consistent with (i)).

A blossom step labels elements in the reverse order of the serial numbers of their mates. (This order is the same as if all the new paths were formed in grow steps). A transform is labelled last and gets the highest serial number.

Now we give formal definitions.

Definition 3.1. A *search path* is a path created by an augment step or $P(e)$ for a labelled element e. In the latter case let $label(e)=(f, g)$. If e is not a transform then

$$\begin{aligned} P(e) &= e && \text{if } f=g=\emptyset;\\ &= e\bar{e}P(f), && \text{if } f\neq\emptyset,\ g=\emptyset;\\ &= P(g, e)^r P(f), && \text{if } f, g\neq\emptyset. \end{aligned}$$

If e is a transform $T(t_0, t_1, b)$ then

$$\begin{aligned} P(e) &= e\bar{t}_0 P(f), && \text{if } f\neq\emptyset,\ g=\emptyset;\\ &= eP(g, \bar{t}_0)^r P(f), && \text{if } f, g\neq\emptyset. \end{aligned}$$

By convention the mate of a transform is considered to be the tip mate $\bar{t}_0$, as in Definition 3.1. This allows us to consider search paths as parity paths.

Now we define blossom. A blossom step for edge e_0e_1 is illustrated in Figure 3.3 (circles represent previous blossoms). b_0 is computed as the first element of $P(e_0)$ such that $P(e_1)$ contains an element in $B(b_0)$; b_1 is the first element of $P(e_1)$ in $B(b_0)$. (If b_0 is not in a previous blossom then $B(b_0)=b_0$ and b_0 is in both paths.) The arbitrary choice of $P(e_0)$ is irrelevant — $B(b_0)$ is the first blossom common to both search paths.

Definition 3.2. Suppose the algorithm does a blossom step for edge e_0e_1, computing b_0, b_1 as above. This forms a *blossom* B. If $b_0\neq b_1$, then $B=\cup B(e)$, where the union is over all elements $e\in P(e_i, b_i)$, $i=0, 1$; the *bud* and *tips* of B are those of $B(b_0)$. If $b_0=b_1$, let t_i be the immediate predecessor of b_0 in $P(e_i)$, $i=0, 1$. Then $B=\cup B(e)+T$, where the union is over all elements $e\in P(e_i, t_i)$, $i=0, 1$, and T is the transform $T(t_0, t_1, b_0)$. The *bud* of B is b_0; the *tips* of B are t_i plus the tips of $B(t_i)$ (if t_i is already in a blossom). In either case, an *old tip* of B is an element in B that was once the tip of a blossom but is not a tip of B.

Note that the bud is *not* in the blossom. In Figure 3.3(b) the tips are the t_i shown (a special case is when t_0 or t_1 is not in a previous blossom). This illustrates how the number of tips for a blossom starts at two and can increase. The algorithm creates a transform in Figure 3.3(b). When a blossom forms all previous blossoms contained in it are considered destroyed. Hence $B(e)$, the blossom containing e, is well-defined.

We close the section with the details of how the algorithm works in Figure 3.2 Figure 3.2(a) shows a graphic matroid; a is a singleton, $\bar{b}$, $\bar{c}$ and $\bar{g}$ are not shown. Figure 3.2(b) illustrates the search paths grown by the algorithm: solid edges are dependence edges, dotted edges join paired elements, and numbers are serial numbers. The algorithm starts by giving singleton a a null label. Scanning a's adjacencies to e

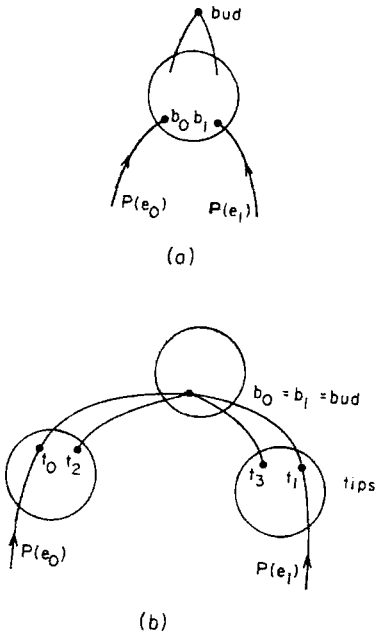


Fig. 3.3. Tips and bud in a blossom step

and f gives back pointers to a for $\bar{e}$ and $\bar{f}$. Scanning $\bar{e}$ gives a back pointer for $\bar{d}$. Scanning $\bar{f}$ gives no new labels ($\bar{d}$ is labelled, but the blossom step is postponed; the adjacency to d does not give another back pointer to $\bar{d}$). Scanning $\bar{d}$'s adjacency to $\bar{f}$ now gives the blossom step. The bud is a and the tips are e, f. d gets the label $(\bar{f}, \bar{d})$; transform $T(e, f, a)$ is created and gets the same label. The blossom contains $d, \bar{d}, e, \bar{e}, f, \bar{f}$, $T(e, f, a)$. Then $\bar{d}$'s adjacency to g gives a back pointer for $\bar{g}$. Next d is scanned. The adjacencies to $\bar{f}$ and $\bar{e}$ go to the same blossom, so nothing is done. The adjacency to g is redundant for a back pointer, so nothing is done. Next the transform is scanned. It is adjacent to c, giving a back pointer for $\bar{c}$. At this point the algorithm scans $\bar{g}$ and then other elements not shown.

4. Correctness

This section proves the algorithm correct. This amounts to two properties — every augmenting path found by the algorithm is feasible (Corollary 4.4), and the final matching is maximum (Corollary 4.5).

For feasibility we show no search path admits an alternating cycle. First we discuss blossoms and deduce that an alternating cycle contains an unlabelled element f (Lemma 4.4). Propagating labels breadth-first implies a monotonicity property of alternating paths starting at such elements f (Lemma 4.7). This prevents the path from becoming a cycle.

First we make two conventions. Blossom and augment steps create the same type of paths (more precisely, the blossom routine could be used to generate labels to define the augmenting path found by the algorithm). In this section "blossom step" refers to a labelling step done by the algorithm explicitly in the blossom routine or implicitly in the augment routine. Also it is convenient to use an ellipsis notation for paths. For example $P=\ldots e..fg\ldots$ means that the path P contains elements e, f and g, in that order, with f and g consecutive. Any assumption about what the dots stand for (e.g., possibly null sequence or nonnull) is always stated explicity.

Lemma 4.1 gives a structure of blossoms analogous to the correctness proof for graph matching [7]. First we verify that a search path $P(e)$ is well-defined. (In Definition 3.1 this may not be clear for the last case, a transform labelled in a blossom step — $\bar{t}_0$ might not be in $P(g)$.) A simple induction (similar to the proof of Lemma 4.1) shows $P(e)$ is a well-defined parity path, where any element an even distance from e has lower serial number than e unless it is an old tip. (Note that $P(e)$ always begins $e\bar{e}\ldots$. For e labelled in a blossom step this follows from the fact on serial numbers, which implies that if e is in a search path $P(x)$ before it is labelled, it occurs an odd distance from x, and hence is preceded by its mate).

Lemma 4.1. *At any point in the algorithm any search path $P(e)$ decomposes as*

$$P(e) = P(e_0, t_0) \cdots P(e_{k-1}, t_{k-1})e_k,$$

where $e_0=e$, e_k is a singleton, and for all i either

(i) *$t_i=\bar{e}_i$, $P(e_i, t_i)=e_i\bar{e}_i$; $e_i, \bar{e}_i$ are not in a blossom, and e_{i+1} is the back pointer of e_i; or*

(ii) *t_i is a tip of a blossom that contains $P(e_i, t_i)$ and has bud e_{i+1}.*

Proof. The proof is by induction on the step that labels e (considering steps in order of occurrence). Suppose e is labelled in a grow or degenerate blossom step. Thus $P(e)=e\bar{e}P(x)$. With the inductive assertion this gives the lemma.

Next suppose a blossom step for edge xy creates a blossom B. By induction write $P(x)=P(x_0, t_0)\ldots P(x_{j-1}, t_{j-1})x_j$, $P(y)=P(y_0, u_0)\ldots P(y_{k-1}, u_{k-1})y_k$.

The algorithm computes b_0, the first element of $P(x)$ such that $P(y)$ contains an element in $B(b_0)$, and b_1, the first element of $P(y)$ in $B(b_0)$. Observe that for some indices r, s, $b_0=x_r$ and $b_1=y_s$: If b_0 is in a blossom this follows easily from (ii). If b_0 is not in a blossom it occurs in both paths (since $B(b_0)=b_0$) in a subpath of type (i). Such a subpath has its first element labelled but not the second. So the same subpath is in both paths and the observation follows.

By definition $P(y)$ does not contain an element in $B(x_i)$, $i<r$. Hence $P(x, x_r)$ and $P(y, y_s)$ are disjoint, except possibly at their ends; $x_{r+1}=y_{s+1}$ and $P(x)$ and $P(y)$ are identical after that element.

If $b_0\neq b_1$ then B contains $B(f)$ for f in $P(x, x_r)$ or $P(y, y_s)$; its bud and tips are those of $B(x_r)$. If $b_0=b_1$ then B contains $B(f)$ for f in $P(x, t_{r-1})$ or $P(y, u_{s-1})$. Its bud is $x_r=y_s$. Its tips are the tips of $B(t_{r-1})$ if it is a blossom else t_{r-1}, plus the tips of $B(u_{s-1})$ if it is a blossom else u_{s-1}.

The inductive assertion for blossom steps now follows by simple verification of various cases: For $e=x$, t_0 in the new decomposition is the old t_r in the first case and t_{r-1} in the second. Similarly for $e=y$, t_0 is the old u_s or u_{s-1}. For newly labelled

elements, including the transform created (if any), t_0 is the same as for x or y. The other cases to be checked are previously labelled elements f, with either f in B or $P(f)$ containing an element of B. ∎

Here are several useful properties of search paths and blossoms. Most follow from the lemma; others are placed here for convenience.

Corollary 4.1. *Let B be a blossom with bud b, and t any tip of B.*

(i) *A search path is simple.*

(ii) *B consists of transforms and parity pairs.*

(iii) *t is unlabelled (and hence not a transform). $\bar{t}$ either was labelled in a grow step from b, or is unlabelled if a degenerate blossom step was done for t, $\bar{t}$ from b.*

(iv) *If $P(e)=\ldots ff'\ldots$ with $s(f)>s(e)$, f not an old tip, then $P(e)=P(e,f')P(f)$.*

Proof. (iv) If e is labelled in a grow or degenerate blossom step just apply the lemma to $P(e)$ when it is formed. The remaining case is e labelled in a blossom step. In the proof of the lemma, first assume $e\in P(x)$. The order of assigning labels in a blossom step shows $f\in P(y)$; now the lemmas's decomposition implies the desired one. The other possibility is that e is the transform of the blossom step. Since it is labelled last the same argument applies. ∎

Next we analyze unlabelled elements in blossoms.

Lemma 4.2. *Let e be an old tip, with B the last blossom containing e as a tip. Some tip e^+ of B gets labelled in the blossom step making e an old tip. An element adjacent to e is adjacent to e^+ or to some transform T created for two tips of $B\big(s(T)<s(e^+)\big)$.*

Proof. Lemma 4.1 shows e^+ exists (and is unique). The algorithm creates a transform each time it merges two sets of tips; hence the transform tree R of Corollary 2.4 exists. Let b be the bud of B. By Corollary 2.4, if an element f is not adjacent to any of the transforms created for tips of B, vector $v(f)$ is a multiple of $v(b)$. The latter has all coefficients nonzero (Corollary 4.1(iii)). So f adjacent to e implies it is adjacent to some transform for tips of B or to every tip of B, in particular e^+. ∎

Corollary 4.2. *If $e\in P(e^+)$ then $e=e^+$.*

Proof. The proof of Lemma 4.1 shows any search path $P(e)$ containing an element of a blossom B contains exactly one tip of B. Now the corollary follows by examining the blossom step that labels e^+. ∎

The following notation is useful for properties resulting from the order of labelling steps.

Definition 4.1. (i) For element e, e^+ is the element of Lemma 4.2 if e is an old tip, else e.

(ii) For element e, $x(e)$ is the element that is the first back pointer assigned to e, $\bar{e}$, or an associated transform (if such exists). Thus if e is labelled in a grow step from f, $x(e)=x(\bar{e})=f$. If a degenerate blossom step for e, $\bar{e}$ is done from f the same holds. For a transform $x\big(T(e_0, e_1, b)\big)=b$, the bud.

For a transform created in a nondegenerate blossom step, interpreting the mate of $T(e_0, e_1, b)$ as $\bar{e}_0$ justifies this definition (see Corollary 4.1(iii)). For succinctness we drop parentheses in compositions of functions, e.g., $s(x(e))=sx(e)$.

Lemma 4.3. *An element y with $s(y^+)<sx(e)$ is not adjacent to e.*

Proof. Let y be an element with $s(y)<sx(e)$. If e is labelled in a grow step, e and $\bar{e}$ are unlabelled when $x(e)=x(\bar{e})$ is scanned. Since y is already scanned it is not adjacent to e or $\bar{e}$. The same holds if a degenerate blossom step for e, $\bar{e}$ is done. The last case is e a transform $T(e_0, e_1, b)$. Corollary 4.1(iii) implies $x(e_i)=b=x(e)$. The previous cases show y is not adjacent to e_i. Corollary 2.2 shows y is not adjacent to e.

Finally suppose y is an old tip with $s(y^+)<sx(e)$. If e is adjacent to y, Lemma 4.2 implies it is adjacent to y^+ or a transform T, $s(T)<s(y^+)$. But the previous paragraph shows this is false. ∎

Corollary 4.3. (i) *The bud b of a blossom B is adjacent to all tips of B and no other elements of B.*

(ii) *If ef is an edge with $s(e)<s(f)<\infty$ then after ef is scanned from f, $B(e)=B(f)$. If ef is an edge with $s(e^+)<s(f^+)<\infty$ then after f^+ is scanned, $B(e)=B(f)$.*

Proof. (i) The first part follows from Corollary 4.1 (iii). For the second part first observe that any element $e\in B$ has $sx(e)\geqq s(b)$; equality holds only for e a tip of B, its mate, or a transform for tips of B. (This follows from the proof of Lemma 4.1). If $sx(e)>s(b)$ the lemma shows b is not adjacent to e. If $x(e)=b$ suppose e is not a tip. A mate of a tip (that is not itself a tip) is not adjacent to b; similarly a transform for tips of B is not adjacent to b.

(ii) First suppose e and f are labelled. A blossom step for ef is done when f is scanned. The only way for e or f — say e — not to be in the new blossom B is to have, in Definition 3.2, $b_0=b_1=e$ and e the bud of B (and the tip in $P(e)$ does not exist). But then e and f contradict (i). (This argument also implies any blossom has at least two tips, although this fact is not needed).

Next suppose e or f is an old tip. Apply the previous paragraph to two adjacent labelled elements given by Lemma 4.2, having serial numbers at most $s(f^+)$. ∎

Now we define an edge that occurs in an alternating cycle for a search path created in a blossom step. First some notation. If e is labelled in a blossom step for edge x_0x_1, denote its search path as $P(e)=P_0P_1$, where $P_1=P(x_1)$, $P_0=P(x_0, e)^r$ or for a transform e, $eP(x_0, \bar{t})^r$.

Definition 4.2. Let e be labelled in a blossom step for edge x_0x_1. An edge $g_0g_1\neq x_0x_1$ with $g_i\in P_i$ is a *cross edge for e, x_0x_1*. (Although dependence graph edges are undirected, in the notation "cross edge g_0g_1 for x_0x_1" order is significant).

Since $P(e)$ is simple the cross edge g_0g_1 has $g_i\notin P_{1-i}$. The basic significance of cross edges is that if a blossom step creates the first search path $P(e)$ with an alternating cycle A, then $P(e)$ has a cross edge: A is not alternating for P_i since these paths exist before the blossom step. (Even if e is a transform, A does not contain e since any $P(e)$ has no dependence graph edge incident to e; hence the relevant part of P_0 exists). So A has at least two edges joining P_0 and P_1. $P(e)$ has only one such edge, x_0x_1. The other edge of A is a cross edge.

Lemma 4.4. *Let g_0g_1 be a cross edge for e, x_0x_1.*

(i) *For some $i\in\{0,1\}$, $s(g_i^+)>s(x_i)$.*
(ii) *For $i\in\{0,1\}$, the above condition $s(g_i^+)>s(x_i)$ implies $s(g_{1-i}^+)>sx(g_i)$.*
(iii) *If $s(x_1)\geqq s(g_1^+)$ then $s(g_1^+)>sx(e)$.*
(iv) *If $f\in P(e)$ with $s(f^+)>s(e)$ then an element $y\in P(e)$ with $s(y^+)\geqq sx(f)$ is not adjacent to e.*

Proof. Suppose the blossom step for x_0x_1 has bud b and creates transform T, if any.

(i) Immediately before the blossom step for x_0x_1, $B(g_0)\neq B(g_1)$, since P_0 does not pass through a (previous) blossom common to $P(x_0)$, $P(x_1)$ (see the proof of Lemma 4.1). We prove (i) by showing that $s(g_i^+)\geqq s(x_i)$ for $i=0,1$ implies the opposite, $B(g_0)=B(g_1)$.

First suppose $s(g_i)\geqq s(x_i)$, $i=0,1$. Write $\{g_0,g_1\}=\{g,g'\}$ where $s(g)>s(g')$ and $\{x_0,x_1\}=\{x,x'\}$ where $s(x)>s(x')$. By Corollary 4.3 (ii) it suffices to show that g_0g_1 is scanned from g before x_0x_1 is scanned from x. The supposition implies $s(g)\geqq s(x)$. If $s(g)<s(x)$ then g is scanned before x, as desired. If $s(g)=s(x)$, i.e., $g=x$, then $s(g')<s(x')$ and gg' is scanned from g before xx', as desired.

Finally suppose $s(g_i^+)\geqq s(x_i)$, $i=0,1$, and either g_i is an old tip. This implies g_i is an old tip before the blossom step. Lemma 4.2 gives an element f_i in $B(g_i)$ (before the blossom step) with $s(f_i)\geqq s(g_i^+)$. This gives a dependence edge joining $B(g_0)$ to $B(g_1)$. It may not be a cross edge but the argument of the previous paragraph still applies. (Note that the new edge is not x_0x_1: This would imply $f_i=g_i^+=x_i$ for some i. But Corollary 4.2 shows $g_i\notin P(g_i^+)$).

(ii) From Lemma 4.3 it suffices to show $g_{1-i}^+\neq x(g_i)$. Suppose $g_i\neq T$. The hypothesis implies g_i is unlabelled and not an old tip when x_i gets labelled. So Lemma 4.1 shows $x(g_i)$ is the successor of g_i in $P(x_i)$, and hence in P_i. Since g_0g_1 is a cross edge $g_{1-i}\neq x(g_i)$. Next suppose $g_i=T$. The same inequation holds, since a transform is not adjacent to its bud (Corollary 2.2).

It remains to consider g_{1-i} an old tip. If $s(g_{1-i}^+)\geqq s(x_i)$ the conclusion is immediate, so assume the opposite. Thus g_{1-i} is an old tip when e is labelled. Proceed as in (i).

(iii) Part (i) implies $s(g_0^+)>s(x_0)$. So (ii) implies $s(g_1^+)>sx(g_0)$. It suffices to show $sx(g_0)\geqq sx(e)$. This is implicit in (ii): If $g_0\neq T$, it is unlabelled and not an old tip when x_0 gets labelled, and Lemma 4.1 shows that in $P(x_0)$, $x(g_0)$ is the successor of g_0. If $e\neq T$ a similar statement holds for e and $sx(g_0)\geqq sx(e)$, as desired. If $e=T$ then $x(e)=b$ and the same inequality holds. Finally if $g_0=T$ then $g_0\in P_0$ shows $g_0=T=e$, and the desired inequality is an equality.

(iv) If e is labelled in a blossom step, the order of labelling shows $sx(e)\geqq sx(f)\geqq s(y^+)$. Lemma 4.3 gives the desired conclusion unless both inequalities are equalities. This implies $e=T$, f is a tip and $y^+=b$. Then $y\in P(e)$ implies $y\in P(y^+)$, by Lemma 4.1. So Corollary 4.2 shows $y^+=y$. Now y is not adjacent to e since a transform is not adjacent to its bud.

Suppose e is labelled in a grow or degenerate blossom step. Since f is not an old tip when e is labelled, Lemma 4.1 shows $sx(e)\geqq sx(f)\geqq s(y^+)$. Lemma 4.3 gives the desired conclusion unless $y^+=x(e)$. Since $y\in P(e)$ and clearly $y\neq e,\bar{e}$, Corollary 4.2 implies $y^+=y$. This gives the desired conclusion since $x(e)$ is not adjacent to e in this case. ∎

The next definition describes the structure of alternating paths for search paths. Throughout this section we assume alternating paths consist of directed edges, i.e., saying ff' is in an alternating path implies elements f, f' occur in that order. We do not assume this for search paths $P(e)$. However we often consider $P(e)$ to be a set of dependence graph edges ff', writing $ff' \in P(e)$.

Definition 4.3. To say an alternating path A for $P(e)$ is *blocked* means that if A starts with edge $ff' \in P(e)$, where $s(f^+) > s(e)$, then every edge gg' of A in $P(e)$ has $s(g'^+) \leqq s(f')$. A labelled element e is *1-blocked* if any alternating path for $P(e)$ is blocked. An element e labelled in a blossom step for edge x_0x_1 is *2-blocked* if x_0 and x_1 are 1-blocked.

For motivation we state the next four results and then give the proofs.

Lemma 4.5. *If e is a 1-blocked element labelled in a grow or degenerate blossom step, an alternating path for $P(e)$ is not a cycle if it starts with edge $\bar{e}x \in P(e)$. If e is 2-blocked an alternating path for $P(e)$ is not a cycle if it starts with a cross edge.*

Lemma 4.6. *A 2-blocked element is 1-blocked.*

Lemma 4.7. *A labelled element is 1-blocked.*

Corollary 4.4. (i) *A search path does not have an alternating cycle (in the final, generalized dependence graph G).*

(ii) *An augment step constructs a base N^* with one more pair than M^*.*

Proof of Lemma 4.5. Let A be an alternating path for $P(e)$ satisfying the hypothesis of the lemma. For e 1-blocked, if A is a cycle it has the form $A = \bar{e}x \ldots gg'\bar{e}$ with last edge $g'\bar{e} \notin P(e)$. Hence $gg' \in P(e)$. Since A is blocked $s(g'^+) \leqq s(x)$. Since $x = x(\bar{e})$, Lemma 4.3 implies $s(g'^+) = s(x)$, $g'^+ = x$. This implies $g' = x$, since $g' \in P(g'^+)$. But this contradicts $g'\bar{e} \notin P(e)$.

For e 2-blocked, let e be labelled in a blossom step for x_0x_1 and let the cross edge starting A be g'_0f_1. Suppose A is a cycle. By Lemma 4.4 (i) A can be oriented so that for some $i(1) \in \{0, 1\}$, $f_1 \in P_{i(1)}$, $s(f_1^+) > s(x_{i(1)})$. After g'_0f_1, A consists of subpaths that are alternately in P_0 and P_1; let the j^{th} subpath be in $P_{i(j)}$ and denote it as $f_jf'_j \ldots g_jg'_j$ (j starts at 1). It suffices to show $s(g'^+_j)$ decreases monotonically, since this implies A is not a cycle.

Inductively assume the sequence up to $s(g'^+_{j-1})$ decreases monotonically and further, $s(f_j^+) > s(x_{i(j)})$. (The base case $j = 1$ is shown above). Hence $f'_j = x(f_j)$ and Lemma 4.4 (ii) shows $s(g'^+_{j-1}) > s(f'_j)$. Since the j^{th} subpath is blocked $s(f'_j) \cong s(g'^+_j)$. Transitivity shows $s(g'^+_{j-1}) > s(g'^+_j)$, giving the first part of the induction. It also implies $s(x_{i(j)}) \cong s(g'^+_j)$, since $s(x_{i(j)}) \cong s(f'_j)$. Hence by Lemma 4.4 (i) for cross edge g'_jf_{j+1}, $s(f^+_{j+1}) > s(x_{i(j+1)})$, giving the second part of the induction. ∎

Proof of Lemma 4.6. Let e be a 2-blocked element labelled in a blossom step for x_0x_1. Let the blossom have bud b and transform T (if any), $P(e) = P_0P_1$.

An element f with $s(f^+) > s(e)$ is in P_1, by the order of labelling in a blossom step. It suffices to show that if $ff' \ldots gg'$ is an alternating path for P_1 and $gg' \in P_1$, then there is no cross edge from g'. For then x_1 1-blocked gives the desired conclusion.

Clearly $s(x_1)>s(f')$. Since x_1 is 1-blocked $s(f')\geqq s(g'^+)$. So $s(x_1)>s(g'^+)$. A cross edge from g' implies $sx(e)<s(g'^+)$, by Lemma 4.4 (iii). It suffices to show $sx(e)\geqq s(f')$. For with $s(f')\geqq s(g'^+)$ this gives the contradiction $s(g'^+)>s(g'^+)$.

If $e\neq T$ the order of assigning labels in a blossom step shows $sx(e)\geqq s(f')$, as desired. Suppose $e=T$. Since the transform gets the highest serial number in a blossom step, $s(f^+)>s(e)$ implies f was not labelled in the blossom step. Thus f does not precede the tip in P_1. Lemma 4.1 for $P(x_1)$ shows $s(b)\geqq s(f')$. Since $b=x(e)$ this is the desired inequality. ∎

Proof of Lemma 4.7. First observe that if e is 1-blocked there is no alternating path $ff'...gg'e$ with ff', $gg'\in P(e)$, $s(f^+)>s(e)$. For 1-blocked implies $s(g'^+)\geqq s(f')$. Note $f'=x(f)$. So Lemma 4.4 (iv) shows g' is not adjacent to e.

Now prove the lemma by induction on serial number $s(e)$. First consider an element e labelled in a grow or degenerate blossom step, so $P(e)=e\bar{e}P(x)$. Let A be an alternating path for $P(e)$ that starts with an edge $ff'\in P(e)$, $s(f^+)>s(e)$. Show A is blocked by considering two cases, $f\in P(x)$ and $f=\bar{e}$.

Suppose $f\in P(x)$. Since x is 1-blocked it suffices to show A does not contain edge $\bar{e}x$ (in either direction). If it does the portion before $\bar{e}x$ is alternating for $P(x)$ — write it as $ff'...gg'$, with $gg'\in P(x)$. The preliminary observation shows g' is not adjacent to x. So it is adjacent to $\bar{e}$. But $s(g'^+)\leqq s(f')<s(x)=sx(\bar{e})$ and Lemma 4.3 show g' is not adjacent to $\bar{e}$, the desired contradiction.

Now suppose $f=\bar{e}$. Write $A=\bar{e}x...gg'...$ with $gg'\in P(e)$. Show $s(g'^+)\leqq s(x)$ as follows. Since A is simple $gg'\in P(x)-x$. If $s(g'^+)>s(x)$ the preliminary observation applied to the reversed subpath of A, $g'g...x$ gives the desired contradiction (the last edge of this path is not in $P(x)$). This completes the second case.

An element e labelled in a blossom step is trivially 1-blocked. So the induction is complete. ∎

Proof of Corollary 4.4. (i) Let e be the first element labelled such that $P(e)$ has an alternating cycle A. Deduce a contradiction as follows.

If e is labelled in a grow or degenerate blossom step, $P(e)=e\bar{e}P(x)$ and A contains edge $\bar{e}x$. Make it the first edge of A. Since e is 1-blocked A is not a cycle, by Lemma 4.5.

If e is labelled in a blossom step choose a cross edge in A. (It exists by the discussion after Definition 4.2.) Make it the first edge of A. Since e is 2-blocked A is not a cycle, by Lemma 4.5.

(ii) First observe that, although a search path can contain more than one transform with the same tip e, there is at most one of the form $T(e, t_1, b)$. (The opposite implies that $\bar{e}$ occurs twice in a search path, a contradiction. Note that if a degenerate blossom step was done for e, $\bar{e}$, then $\bar{e}$ is always interpreted as $T(\bar{e}, e, b)$). Furthermore, any $T(e, t_1, b)$ in the path has higher serial number than any transform of the form $T(t_0, e, b)$. To show this suppose a step of the algorithm creates a search path where the opposite holds. The step must be a blossom step; Lemma 4.1 shows that the new path is $P(T(t_0, e, b))$. In this path tip e is preceded by its mate $\bar{e}$; an occurrence of $T(e, t_1, b)$ gives another occurrence of $\bar{e}$, a contradiction.

Now let P be the augmenting path found by the algorithm. Construct a base B as follows: Initially B is M^*. Consider all transforms $T(t_0, t_1, b)$ in order of increasing serial number. If $T(t_0, t_1, b)\in P$ with $t_i\in M$, replace t_0 in the base by the transform.

Note that the construction is correct: It suffices to show that tips t_i are in the

base when $T(t_0, t_1, b)$ is considered, by Corollary 2.3. The preliminary observation shows that t_0 is in the base when $T(t_0, t_1, b)$ is considered, since t_0 occurs only once as the first tip of a transform. The observation also shows that t_1 is in the base when $T(t_0, t_1, b)$ is considered, since a transform containing t_1 as first tip has higher serial number.

Next observe that the dependence graph for base B is given by the subgraph of G (the algorithm's final dependence graph) that corresponds to the nodes of B. This follows by induction on the transforms added: Recall that the algorithm constructs a transform using formula (2) of Lemma 2.1, with the values $c(a_0, b_i)$ taken as the scalars that are the coefficients of a_0 with respect to base M^*. This equation shows that replacing t_0 by $T(t_0, t_1, b)$ does not change any coefficient $c(a, \beta)$ (with the obvious exceptions $\beta = t_0$, $T(t_0, t_1, b)$). Hence by induction, the transform constructed by the algorithm is the same as that given by Lemma 2.1 applied to the current base B. Thus the scalars $c(a_0, a_1, b_0, b_1)/c(a_0, b_1)$ in (2), which determine the edges in the algorithm's dependence graph, are exactly those for the transform given by Lemma 2.1 for B.

$B \oplus P$ is a base by (i) and Theorem 2.2. Now construct a base C as follows: Initially C is $B \oplus P$. Consider all transforms $T(t_0, t_1, b)$ in order of decreasing serial number. If $T(t_0, t_1, b) \in C$, replace it by t_0.

To show the construction is correct it suffices to show that when $T(t_0, t_1, b)$ is considered, t_1 is in the base but t_0 is not, by Corollary 2.3. The argument is the same as for the construction of B.

C is a base with all elements in the given matroid, and it has one more pair than M^*. It is easy to see that C is the base N^* constructed by the algorithm. ∎

The second part of the correctness proof shows the final matching is maximum, by constructing a parity partition and using Lemma 2.2. Suppose the algorithm terminates without augmenting the matching M. The partition of E (analogous to the one for graph matching) is defined as follows:

$E_0 = \{e | e$ and $\bar{e}$ are unlabelled and are not in a (degenerate) blossom$\}$;

E_i, $i = 1, \ldots, k$ is either $B \cap E$ for a blossom B, or a pair $e, \bar{e}$ (in E) with exactly one of $e, \bar{e}$ labelled.

Also set $M_i = M \cap E_i$, $i = 0, \ldots, k$. In these definitions labels and blossoms refer to those when the algorithm terminates.

The argument is similar to graph matching. The main complication is that in graph matching the matched edges of a blossom plus its base span the corresponding set of the partition. This need not be true in linear parity: an element of a blossom can be adjacent to tips of another blossom (provided it is not adjacent to the corresponding transforms). Define sets where these "bad" adjacencies can occur:

$T_0 = \emptyset$;

$T_i =$ the unlabelled elements in E_i that are not old tips, if $i > 0$.

(Here T stands for tip, not transform!) If $E_i = B \cap E$ then T_i consists of the tips of B. In this case define b_i as the bud of B. Finally define the set A (of Lemma 2.2) to contain all elements α_i, where

$\alpha_i = e$, for E_i a pair of matched elements $e, \bar{e}$ with e unlabelled;

$= \sum_{\beta \in T_i} c(b_i, \beta)\beta$, for $E_i = B \cap E$ and $T_i \subseteq M$;

$=$ undefined, otherwise.

Observe that some of the elements defined above are not necessarily in E: α_i for blossoms, and buds b_i that are transforms or singletons of M^*.

We will show that

$$r(E_i \cup A) = r(A) + |M_i| + \delta_i, \tag{1}$$

where $\delta_0 = 0$, $\delta_i = 1$ if $i > 0$ with $T_i \not\subseteq M$, and $\delta_i = -1$ if $i > 0$ with $T_i \subseteq M$. Hence

$$r(A) + \sum_{i=0}^{k} \left\lfloor \frac{r(E_i \cup A) - r(A)}{2} \right\rfloor = r(A) + \frac{|M|}{2} - |\{i \mid i > 0,\ T_i \subseteq M\}| = \frac{|M|}{2}.$$

Thus Lemma 2.2 shows the matching is maximum.

(1) is proved by exhibiting a base of $E_i \cup A$. Towards this end observe that $M_0 \cup A$ is independent; for $i > 0$ with $T_i \not\subseteq M$, $M_i \cup A$ is independent; for $i > 0$ with $T_i \subseteq M$, $M_i \cup (A - \alpha_i)$ is independent and spans α_i. We now state the last two lemmas and show how they accomplish the goal. The first leads to the desired base for two cases.

Lemma 4.8. *For any $i = 0, \ldots, k$ and $a \in E_i - M_i - T_i$, $M_i \cup A + a$ is dependent.*

The lemma shows that for $i = 0$, or $i > 0$ with $T_i \subseteq M$, $M_i \cup A$ spans $E_i \cup A$. Thus the observations on independence imply $M_0 \cup A$ is a base of $E_0 \cup A$, and $M_i \cup A - \alpha_i$ is a base of $E_i \cup A$ for $i > 0$ with $T_i \subseteq M$. This gives (1) for these cases.

Lemma 4.9. *Let $a_0, a_1 \in T_i - M_i$ for any $i > 0$. Then $M_i \cup A + a_0$ is independent; $a_0 \neq a_1$ implies $M_i \cup A + a_0 + a_1$ is dependent.*

Lemmas 4.8—9 show that for $i > 0$ and $T_i \not\subseteq M$, $M_i \cup A + a$ is a base of $E_i \cup A$ for any $a \in T_i$. This proves (1) for the last case.

Proof of Lemma 4.8. The desired dependence comes from a's representation,

$$a = \sum_{\beta \in M^*} c(a, \beta)\beta.$$

If $i = 0$ then a was never involved a labelling step. So a is only adjacent to elements of $M_i \cup \bigcup_{j>0} T_j$. ($a$ is not adjacent to an old tip by Lemma 4.2.) If $i > 0$ then a is labelled or an old tip, and the same conclusion holds. (a is not adjacent to a singleton, which is adjacent only to $\bigcup_{j>0} T_j$; it is not adjacent to M_0, as above; it is not adjacent to a labelled element or old tip in another partition, by Corollary 4.3 (ii)). So suppose $f \in M_j$, $j \neq i$, and $c(a, f) \neq 0$. If E_j is the pair $f, \bar{f}$ with $\bar{f}$ labelled then $f \in A$. The remaining case is $E_j = B \cap E$, f a tip of B. a is not adjacent to any transform of B. Corollary 2.4 shows the vector $(c(a,f) \mid f \in T_j)$ is a multiple of $(c(b_j, f) \mid f \in T_j)$. (Note that b_j can be a transform; this is the case of Corollary 2.4 where the bud is given by scalars). Hence for some scalar s,

$$\sum_{\beta \in T_j} c(a, \beta)\beta = s \sum_{\beta \in T_j} c(b_j, \beta)\beta = s\alpha_j.$$

This implies the desired dependence. ∎

Proof of Lemma 4.9. For the first part define a matched element b: If $E_i=\{a_0, \bar{a}_0\}$ with $\bar{a}_0$ labelled, b is $\bar{a}_0$'s back pointer; if $E_i=B\cap E$, b is the bud of B. Thus a_0 is adjacent to b. If b is not a transform then $c(a_0, b)\neq 0$ in a_0's representation in the base M^*. Hence $a_0\notin sp(M_i\cup A)$, as desired. If b is a transform let $b=T(e_0, e_1, f)$. Corollary 2.4 shows $v(a_0)$ is not a multiple of $v(f)$. This implies the same conclusion.

The second assertion is vacuous unless $E_i=B\cap E$. First suppose $T(a_0, a_1, b)$ is a transform generated by the algorithm. As in the previous proof, the desired dependence comes from the representation

$$T(a_0, a_1, b) = c(a_0, b)a_1 - c(a_1, b)a_0 = \sum_{\beta\in M^*} c(a_0, a_1, b, \beta)\beta.$$

The argument is exactly the same, since $T(a_0, a_1, b)$ is adjacent only to $M_i\cup\bigcup_{j>0} T_j$.

To prove that $M_i\cup A+a_0'+a_1'$ is dependent for arbitrary tips $a_j', j=0,1$ of B examine the blossom steps that built up B. Suppose a blossom step merges the tips of B_0 with those of B_1. It constructs a transform $T(a_0, a_1, b)$ for a_j a tip of B_j, $j=0, 1$. Let a_j' be another tip of B_j, $j=0,1$. Inductively assume that $M_i\cup A+a_j+a_j'$ is dependent. As shown above, $M_i\cup A+a_0+a_1$ is dependent. Together with the first part of the lemma (on independence) this implies $M_i\cup A+a_0'+a_1'$ is dependent. Now induction completes the proof. ∎

Corollary 4.5. *When the algorithm halts without augmenting the matching is maximum.* ∎

Corollaries 4.4 and 4.5 show the algorithm is correct. This section also proves a result mentioned in Section 2: In a linear matroid a matching M is maximum if and only if M^* has no augmenting path. (Since transforms do not create adjacencies (Corollary 2.2), the algorithm's augmenting path can be traced in the original dependence graph.)

5. Efficiency

This section analyzes the time and space requirements of the algorithm for linear and graphic matroids. Recall the notation $m=|E|$, $n=r(E)$. Assume (as in [17]) that a representation of the matroid is given. The size of the representation is $O(mn)$, and n bounds the size of a maximum matching.

The final dependence graph G is bipartite, with vertex sets of cardinality $O(n)$ and $O(m)$ (and hence $O(mn)$ edges). To prove this it suffices to bound the number of transforms. The number of transforms for matched tips is $O(n)$, since a blossom step creating such a transform decreases the number of blossoms with matched tips by one (and a matching has at most n elements). Similarly the number of transforms for unmatched tips is $O(m)$.

Estimate the time for the algorithm by dividing it into the time for (i) the main routine; (ii) blossom and augment steps; (iii) creating transforms; (iv) processing $B(e)$ values. Term (ii) includes computing the first common element b in the main routine. Term (iv) refers to accessing values $B(e)$ (in main and blossom) and updating them

(in blossom). Of course the four terms are taken to be mutually exclusive: (i) is the time for the main routine excluding (ii)—(iv), etc.

(i) The main routine takes time proportional to the number of edges in the dependence graph. Note that e's adjacency list will be sorted by serial number when e is scanned, if whenever an element f gets labelled, f is placed behind all other labelled elements in any adjacency list containing it. Hence the main routine is $O(mn)$.

(ii) A blossom step takes time proportional to the length of the paths $P(e_i)$, $i=0, 1$. This is $O(n)$ since alternate pairs of a search path are matched. There are $O(m)$ blossom steps since each one merges at least two blossoms (or elements). So the time is $O(mn)$. (It is not hard to see that an augment step is $O(n)$.)

(iii) From Lemma 2.1 (ii), the time to create a transform with matched tips is bounded by the number of unmatched elements, $O(m)$. (We assume an arithmetic operation takes one unit of time.) There are $O(n)$ such transforms, giving total time $O(mn)$. Similarly the time to create a transform with unmatched tips is $O(n)$ and there are $O(m)$ of them, giving $O(mn)$.

(iv) Since blossoms are merged together (and never split), accessing and updating the values $B(e)$ is the set merging problem [24]. Accessing a value is done by the *find* operation, updating by *union*. There are $O(mn)$ *finds* (in main and blossom), $O(m+n)$ *unions* (in blossom). The total time is $O(mn\ \alpha(mn, m+n))$ [24]. Since $m \geqq n$, usually $\alpha(mn, m+n)=O(1)$, for instance on graphic matroids. Even if this is not so, total time $O(mn)$ for set merging can be achieved using the algorithm of [11], since as in graph matching the union tree grows one vertex at a time.

This gives the following result.

Lemma 5.1. *Given a representation of the matroid with respect to the base M^*, the algorithm runs in time $O(mn)$ and space $O(mn)$.* ∎

The time for finding a maximum matching is dominated by the time at the beginning of each iteration to compute the representation with respect to the new base M^* (which was changed by the augment). Let $\mathbf{C}$ be the matrix with entries $c(b', b)$, where b' ranges over the elements of the new base (indexing $\mathbf{C}$'s rows) and b ranges over the elements of the original base (indexing $\mathbf{C}$'s columns). Represent elements as row vectors. An element e whose original representation is vector $\mathbf{e}$ (with entries $c(e, b)$) has new representation $\mathbf{e}'=\mathbf{e}\mathbf{C}^{-1}$. So the new representation is computed by inverting an $n\times n$ matrix and multiplying m n-vectors by an $n\times n$ matrix. Since $m \geqq n$ the time is $O(mn^2)$ using naive methods. Batching the m multiplications into groups of n, the time is $O(mM(n)/n)$ where $M(n)$ is the time to multiply two $n\times n$ matrices [2]. Using the algorithm of Coppersmith and Winograd the time is $O(mn^{1.5})$ [3].

Finding a maximum matching requires at most $n/2$ calls to the algorithm. This gives the time bound.

Theorem 5.1. *The linear matroid parity problem can be solved in time $O(mn^3)$ ($O(mn^{2.5})$ using fast matrix multiplication) and space $O(mn)$ (using uniform cost).* ∎

In graphic matroids, the adjacencies for a given base can be computed from the spanning tree — there is no need for matrix inversion. This speeds up the algorithm. Matroid parity on cographic matroids is solved by running the algorithm on the dual graphic matroid.

Theorem 5.2. *The graphic and cographic parity problems can be solved in time* $O(mn^2)$ *and space* $O(mn)$. ∎

This can be improved to $O(m)$ space (there is no need to store the dependence graph explicitly — adjacencies can be computed from the spanning tree as needed); alternatively, $O(nm \lg^6 n)$ time and $O(m \lg^4 n)$ space. Details are in [9].

References

[1] M. AIGNER, *Combinatorial Theory*, Springer Verlag, Berlin, 1979.

[2] A. V. AHO, J. E. HOPCROFT and J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Ma., 1974.

[3] D. COPPERSMITH and S. WINOGRAD, On the asymptotic complexity of matrix multiplication, *SIAM J. Comput.*, **11** (1982), 472—492.

[4] J. EDMONDS, Paths, trees and flowers, *Canadian J. of Math.*, **17** (1965), 449—467.

[5] S. EVEN and O. KARIV, An $O(n^{2.5})$ algorithm for maximum matching in general graphs, *Proc. 16th Annual IEEE Symp. on Foundations of Computer Science*, Berkeley, (1975), 100—112.

[6] H. N. GABOW, Decomposing symmetric exchanges in matroid bases, *Mathematical Programming* **10** (1976), 271—276.

[7] H. N. GABOW, An efficient implementation of Edmonds' algorithm for maximum matching on graphs, *J. ACM* **23** (1976), 221—234.

[8] H. N. GABOW and M. STALLMANN, Scheduling multi-task jobs with deadlines on one processor, *working paper*.

[9] H. N. GABOW and M. STALLMANN, Efficient algorithms for graphic matroid intersection and parity, *Automata, Languages and Programming; Lecture Notes in Computer Science* **194**, (W. Brauer, ed.), Springer-Verlag, New York, 1985, 210—220.

[10] H. N. GABOW and R. E. TARJAN, Efficient algorithms for a family of matroid intersection problems, *J. Algorithms* **5** (1984), 80—131.

[11] H. N. GABOW and R. E. TARJAN, A linear-time algorithm for a special case of disjoint set union, *J. of Comp. and Sys. Sciences* **30** (1985), 209—221.

[12] M. R. GAREY and D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, 1979.

[13] P. JENSEN and B. KORTE, Complexity of matroid property algorithms *SIAM Journal on Computing*, **11** (1982), 184—190.

[14] S. KROGDAHL, The dependence graph for bases in matroids, *Discrete Mathematics* **19** (1977), 47—59.

[15] E. L. LAWLER, Matroids with parity conditions: a new class of combinatorial optimization problems, *Electronics Research Laboratory, Berkeley, Memorandum Number* ERL—M334 (1971).

[16] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, San Francisco, 1976.

[17] L. LOVÁSZ, The matroid matching problem, *Algebraic Methods in Graph Theory*, Colloquia Mathematica Societatis János Bolyai, Szeged (Hungary), 1978.

[18] L. LOVÁSZ, Selecting independent lines from a family of lines in space, *Acta Scientarium Mathematicarum*, **42** (1980), 121—131.

[19] L. LOVÁSZ, Matroid matching and some applications, *J. Comb. Theory, B*, **28** (1980), 208—236.

[20] S. MICALI and V. V. VAZIRANI, An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs, *Proc. 20th Annual IEEE Symp. on Foundations of Computer Science*, 1980, 17—27.

[21] J. B. ORLIN and J. H. VANDE VATE, An algorithm for the linear matroid parity problem, *manuscript*.

[22] PO TONG, E. L. LAWLER and V. V. VAZIRANI, Solving the weighted parity problem for gammoids by reduction to graphic matching, *in: Combinatorial Optimization*, (W. R. Pulleyblank, ed.), Academic Press, 1984, 363—374.

[23] M. STALLMANN, *An Augmenting Paths Algorithm for the Matroid Parity Problem on Binary Matroids*, Ph.D. Thesis, University of Colorado at Boulder, 1982.

[24] R. E. TARJAN, Efficiency of a good but not linear set union algorithm, *J. ACM* **22** (1975), 215—225.
[25] R. E. TARJAN, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA., 1983.
[26] D. J. A. WELSH, *Matroid Theory*, Academic Press, New York, 1976.
[27] H. WHITNEY, On the abstract properties of linear dependence, *American Journal of Mathematics*, **57** (1935), 509—533.

Harold N. Gabow
Department of Computer Science
University of Colorado at Boulder
Boulder, CO 80309
U.S.A.

Matthias Stallmann
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206
U.S.A.