Algebraic Algorithms for Matching and Matroid Problems

Nicholas J. A. Harvey
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

Abstract

We present new algebraic approaches for several well-known combinatorial problems, including non-bipartite matching, matroid intersection, and some of their generalizations. Our work yields new randomized algorithms that are the most efficient known. For non-bipartite matching, we obtain a simple, purely algebraic algorithm with running time $O(n^\omega)$ where n is the number of vertices and ω is the matrix multiplication exponent. This resolves the central open problem of Mucha and Sankowski (2004). For matroid intersection, our algorithm has running time $O(nr^{\omega-1})$ for matroids with n elements and rank r that satisfy some natural conditions.

1 Introduction

The non-bipartite matching problem — finding a largest set of disjoint edges in a graph — is a fundamental problem that has played a pivotal role in the development of graph theory, combinatorial optimization, and computer science [49]. For example, Edmonds' seminal work on matchings [15, 16] inspired the definition of the class P, and launched the field of polyhedral combinatorics. The matching theory book [38] gives an extensive treatment of this subject, and uses matchings as a touchstone to develop much of the theory of combinatorial optimization.

The matroid intersection problem — finding a largest common independent set in two given matroids — is another fundamental optimization problem, originating in the pioneering work of Edmonds [18, 20]. This work led to significant developments concerning integral polyhedra [48], submodular functions [22], and convex analysis [43]. Algorithmically, matroid intersection is a powerful tool that has been used in various areas such as approximation algorithms [6, 32, 29], mixed matrix theory [42], and network coding [31].

1.1 Matching algorithms

The literature for non-bipartite matching algorithms is quite lengthy. Table 1 provides a brief summary; further discussion can be found in [48, §24.4]. As one can see, there was little progress from 1975 until 2004, when an exciting development of Mucha and Sankowski [41] gave a randomized algorithm to construct a maximum matching in time $O(n^\omega)$, where $\omega < 2.38$ is the exponent indicating the time to multiply two $n \times n$

Authors	Year	Running Time
Edmonds [16]	1965	$O(n^2 m)$
Even and Kariv [21]	1975	$O(\min\{n^{2.5}, \sqrt{n}m\log n\})$
Micali and Vazirani [39]	1980	$O(\sqrt{n}m)$
Rabin and Vazirani [46]	1989	$O(n^{\omega+1})$
Goldberg and Karzanov [30]	2004	$O(\sqrt{n}m\log(n^2/m)/\log n)$
Mucha and Sankowski [41]	2004	$O(n^\omega)$
Sankowski [47]	2005	$O(n^\omega)$
This paper		$O(n^\omega)$

Table 1: *A summary of algorithms for the non-bipartite matching problem. The quantities n and m respectively denote the number of vertices and edges in the graph.*

matrices [10]. A highly readable exposition of their algorithm is in Mucha's thesis [40].

Unfortunately, most of the algorithms mentioned above are quite complicated; the algorithms of Edmonds and Rabin-Vazirani are perhaps the only exceptions. For example, the Micali-Vazirani algorithm was not formally proven correct until much later [53]. The Mucha-Sankowski algorithm relies on a non-trivial structural decomposition of graphs called the "canonical partition", and uses sophisticated dynamic connectivity data structures to maintain this decomposition online. Mucha writes [40, §6]:

> [The non-bipartite] algorithm is quite complicated and heavily relies on graph-theoretic results and techniques. It would be nice to have a strictly algebraic, and possibly simpler, matching algorithm for general graphs.

Interestingly, for the special case of bipartite graphs, Mucha and Sankowski give a simple algorithm that amounts to performing Gaussian elimination lazily. Unfortunately, this technique seems to break down for general graphs, leading to the conjecture that there is no $O(n^\omega)$ matching algorithm for non-bipartite graphs that uses only lazy computation techniques [40, §3.4].

1.2 Matroid intersection algorithms

The discussion of matroids in this section is necessarily informal since we defer the formal definition of matroids until Section 4. Generally speaking, algorithms involving matroids fall into two classes.

Oracle algorithms. These algorithms access the matroid via an oracle which answers queries about its structure.

Linear matroid algorithms. These algorithms assume that a matroid is given as input to the algorithm as an explicit matrix which represents the matroid.

Linear matroid algorithms only apply to a subclass of matroids known as *linear matroids*, but most useful matroids indeed lie in this class.

Authors	Year	Running Time
Cunningham [11]	1986	$O(nr^2 \log r)$
Gabow and Xu [23, 24]	1989	$O(nr^{1.62})$
This paper		$O(nr^{\omega-1})$

Table 2: *A summary of linear matroid algorithms for the matroid intersection problem. The quantities n and r respectively denote the number of columns and rows of the given matrix.*

Authors	Year	Number of Oracle Queries
Edmonds [17][1]	1968	not stated
Aigner and Dowling [2]	1971	$O(nr^2)$
Tomizawa and Iri [51]	1974	not stated
Lawler [35]	1975	$O(nr^2)$
Edmonds [20]	1979	not stated
Cunningham [11]	1986	$O(nr^{1.5})$

Table 3: *A summary of oracle algorithms for the matroid intersection problem. The quantities n and r respectively denote the number of elements and rank of the matroid; they are analogous to the quantities n and r mentioned in Table 2.*

Table 2 and Table 3 provide a brief summary of the existing algorithms for matroid intersection. It should be noted that the Gabow-Xu algorithm achieves the running time of $O(nr^{1.62})$ via use of the $O(n^{2.38})$ matrix multiplication algorithm of Coppersmith and Winograd [10]. However, this bound seems somewhat unnatural: for square matrices their running time is $O(n^{2.62})$, although one would hope for a running time of $O(n^{2.38})$.

1.3 Generalizations

Several variants and generalizations of matchings and matroid intersection have been considered, notably matroid matching [36, 38, 48]. Matroid matching problems on general graphs require exponential time in the oracle model, although sophisticated polynomial-time algorithms do exist for linear matroids. On the other hand, *bipartite* matroid matching problems are tractable: they are polynomial-time reducible to matroid intersection [18, Theorem 81] [22].

Another generalization of matroid intersections and non-bipartite matchings are *basic path-matchings*, introduced by Cunningham and Geelen [13, 14, 26]. Their work shows integrality of related polyhedra and shows that one can optimize over these polyhedra using the ellipsoid method [14]. Later work [12] used algebraic techniques together with a matroid intersection algorithm to compute a basic path-matching.

[1] Edmonds [17] gives an efficient algorithm for the matroid partition problem. As was shown by Edmonds [18] [19], this implies an efficient algorithm for the matroid intersection problem.

1.4 Our results

In this paper, we present new algebraic approaches for several of the problems mentioned above.

Non-bipartite matching. We present a purely algebraic, randomized algorithm for constructing a maximum matching in $O(n^{\omega})$ time. The algorithm is conceptually simple — it uses lazy updates, and does not require sophisticated data structures or subroutines other than a black-box algorithm for matrix multiplication/inversion. Therefore our work resolves the central open question of Mucha and Sankowski [41], and refutes the conjecture [40] that no such lazy algorithm exists.

Our algorithm is based on a simple, but subtle, divide-and-conquer approach. The key insight is: adding an edge to the matching involves modifying two symmetric entries of a certain matrix. (See Section 3 for further details.) These entries may be quite far apart in the matrix, so a lazy updating scheme that only updates "nearby" matrix entries will fail. We overcome this difficulty by traversing the matrix in a novel manner such that symmetric locations are nearby in our traversal, even if they are far apart in the matrix. Our new approach has an important consequence: it easily extends to various generalizations of the non-bipartite matching problem such as path-matchings. It is not clear whether previous algorithms [39, 41] also admit such extensions.

Matroid intersection. We present a linear matroid algorithm for the matroid intersection problem that uses only $O(nr^{\omega-1})$ time. This running time is essentially optimal because computing the rank of a $n \times r$ matrix reduces to matroid intersection of the matrix with itself, and $O(nr^{\omega-1})$ is the best running time for any rank-computation algorithm that we know of. In other words, we show that finding a maximum independent set in *two* matroids requires asymptotically the same time as finding a maximum independent set in just *one* matroid.

Whereas most existing matroid algorithms use augmenting path techniques, ours uses an algebraic approach. Several previous matroid algorithms also use algebraic techniques [4, 37, 44]. This approach requires that the given matroids are linear, and additionally requires that the two matroids can be represented as matrices over the same field. These assumption will be discussed further in Section 4.

Bipartite Matroid Matching. We show a surprising result: the Mucha-Sankowski bipartite graph matching algorithm [41], with minor modification, can actually solve a much more general problem, namely bipartite matroid matching. The runtime of the resulting algorithm remains only $O(n^{\omega})$ time. Our contribution is to construct a new matrix to give as input to that algorithm, and to prove some important properties of that matrix. Our algorithm improves on the $O(n^2 r^{1.62}) = O(n^{3.62})$ time bound which one can obtain by existing matroid intersection algorithms [24], and the standard reduction from bipartite matroid matching to matroid intersection [22].

Basic Path-Matching. We present a matrix which characterizes solvable instances of the basic path-matching problem. This extends Geelen's algebraic framework for ordinary path-matching problems, which do not involve matroids [26]. This allows us to extend our non-bipartite matching algorithm to a $O(n^{\omega})$ algorithm for constructing basic path-matchings.

2 Preliminaries

2.1 Notation

The set of integers $\{1, \ldots, n\}$ is denoted $[n]$. If J is a set, $J + i$ denotes $J \cup \{i\}$. If M is a matrix, a submatrix containing rows S and columns T is denoted $M[S, T]$. A submatrix containing all rows (columns) is denoted $M[*, T]$ ($M[S, *]$). A submatrix $M[S, T]$ is sometimes written as $M_{S,T}$ when this enhances legibility. The i^{th} row (column) of M is denoted $M_{i,*}$ ($M_{*,i}$). An entry of M is denoted $M_{i,j}$. The submatrix obtained by deleting row i and column j (row-set I and column-set J) from M is denoted $M_{\mathrm{del}(i,j)}$ ($M_{\mathrm{del}(I,J)}$). A submatrix containing rows $\{a, \ldots, b\}$ and columns $\{c, \ldots, d\}$ is denoted $M_{a:b,\, c:d}$. When a matrix has been decomposed into blocks such as $\left(\begin{smallmatrix} W & X \\ Y & Z \end{smallmatrix}\right)$, we will refer to the blocks using compass directions, e.g., W is the "north-west" submatrix.

2.2 Assumptions and Conventions

We assume a randomized computational model, in which algorithms have access to a stream of independent, unbiased coin flips. All algorithms presented in this paper are all randomized, even if this is not stated explicitly. Furthermore, our computational model assumes that arithmetic operations all require a single time step.

The value ω is a real number defined as the infimum of all values c such that multiplying two $n \times n$ matrices requires $O(n^c)$ time. Thus, strictly speaking, it is not accurate to say that matrix multiplication requires $O(n^\omega)$ time. This justifies the following notational convention: we will implicitly ignore $\mathrm{polylog}(n)$ factors in expressions of the form $O(n^\omega)$.

2.3 Facts from Linear Algebra

We will use the following basic facts from linear algebra. Proofs are given in Appendix A.

Let $\mathbb{F}$ be a field, let $\mathbb{F}[x_1, \ldots, x_m]$ be the ring of polynomials over $\mathbb{F}$ in indeterminates $\{x_1, \ldots, x_m\}$, and let $\mathbb{F}(x_1, \ldots, x_m)$ be the field of rational functions over $\mathbb{F}$ in these indeterminates. A matrix with entries in $\mathbb{F}[x_1, \ldots, x_m]$ or $\mathbb{F}(x_1, \ldots, x_m)$ will be called a *matrix of indeterminates*. A matrix M of indeterminates is said to be non-singular if its determinant is not the zero function. In this case, M^{-1} exists and it is a matrix whose entries are in $\mathbb{F}(x_1, \ldots, x_m)$. The entries of M^{-1} are given by:

$$(M^{-1})_{i,j} \;=\; (-1)^{i+j} \,\cdot\, \det M_{\mathrm{del}(j,i)} \,/\, \det M,$$

which is a special case of Fact 1 below. Given a matrix of indeterminates, our algorithms will typically substitute values in $\mathbb{F}$ for the indeterminates. So for much of the discussion below, it suffices to consider ordinary numeric matrices over $\mathbb{F}$.

Fact 1 (Jacobi's Determinant Identity). *Let M be a non-singular matrix with row-set and column-set C. Then, for any equicardinal sets $I, J \subseteq C$, we have*

$$\det M[I, J] \;=\; \det M \,\cdot\, \det M^{-1}[C \setminus J, C \setminus I] \,\cdot\, (-1)^{\sum_{i \in I} i + \sum_{j \in J} j}.$$

Fact 2 (Schur Complement)**.** *Let M be a square matrix of the form $M = \left(\begin{smallmatrix} W & X \\ Y & Z \end{smallmatrix}\right)$ where Z is square. If Z is non-singular, the matrix $W - XZ^{-1}Y$ is known as the Schur complement of Z in M. The Schur complement satisfies the following useful property:*

$$\det M \;=\; \det Z \,\cdot\, \det\bigl(W - XZ^{-1}Y\bigr).$$

Additionally, the rank of the Schur complement equals the rank of M minus the size of Z.

Fact 3. *Let $M = \left(\begin{smallmatrix} W & X \\ Y & Z \end{smallmatrix}\right)$ have inverse $M^{-1} = \left(\begin{smallmatrix} \hat{W} & \hat{X} \\ \hat{Y} & \hat{Z} \end{smallmatrix}\right)$. Then W is non-singular iff $\hat{Z}$ is, and $W^{-1} = \hat{W} - \hat{X}\hat{Z}^{-1}\hat{Y}$.*

Fact 4 (Sherman-Morrison Formula)**.** *Let u and v be vectors and c a non-zero scalar. The matrix $\tilde{M} = M + cuv^{\mathsf{T}}$ is called a rank-1 update of M. Assume that M is non-singular and let $\alpha = c^{-1} + v^{\mathsf{T}}M^{-1}u$. The inverse of $\tilde{M}$ exists iff $\alpha \neq 0$, and equals*

$$\tilde{M}^{-1} \;=\; M^{-1} \;-\; \alpha^{-1}\bigl(M^{-1}\,u\bigr)\,\bigl(v^{\mathsf{T}}\,M^{-1}\bigr),$$

which is itself a rank-1 update of M^{-1}.

A matrix M is called *skew-symmetric* if $M = -M^{\mathsf{T}}$. Note that the diagonal entries of a skew-symmetric matrix are necessarily zero.

Fact 5. *Let M be an $n \times n$ skew-symmetric matrix. If M is non-singular then M^{-1} is also skew-symmetric.*

Algorithms. Lastly, let us consider the algorithmic efficiency of operations on matrices with entries in a field $\mathbb{F}$. As mentioned above, we assume that two $n \times n$ matrices can be multiplied in $O(n^{\omega})$ time. This same time bound suffices for determinant computation, rank computation, and inversion (if the matrix is non-singular) [1].

Consider now the problem of rectangular matrix multiplication. For example, one could multiply an $r \times n$ matrix A by a $n \times r$ matrix B, where $r < n$. This can be accomplished by dividing A and B into blocks of size $r \times r$, multiplying the i^{th} block of A by the i^{th} block of B via an $O(r^{\omega})$ time algorithm, then finally adding these results together. Since $\lceil n/r \rceil$ multiplications are performed, the total time required is $O(nr^{\omega-1})$. This basic technique will frequently be used in the subsequent sections. More sophisticated rectangular matrix multiplication algorithms do exist [9], but they will not be considered herein.

3 Non-Bipartite Matching

Tutte matrix. Let $G = (S, E)$ be a graph with $n = |S|$. For each edge $\{i, j\} \in E$, associate an indeterminate $t_{\{i,j\}}$. The *Tutte matrix* T for G is an $n \times n$ matrix of indeterminates where $T_{i,j} = \pm t_{\{i,j\}}$ and the signs are chosen such that T is skew-symmetric. Tutte [52] showed that T is non-singular iff G has a perfect matching (see, e.g., Godsil [28]). However, this does not directly imply an efficient algorithm to test if G has a perfect matching: the determinant of T is a polynomial which may have exponential size, so computing it symbolically is inefficient.

Fortunately, Lovász [37] showed that the rank of T is preserved with high probability after randomly substituting non-zero values for the $t_{\{i,j\}}$'s from a sufficiently large field, say of size $\Theta(n^2)$. After this numeric substitution, the determinant of the resulting matrix can be computed in $O(n^\omega)$ time. This immediately implies an efficient algorithm to *test* if a graph has a perfect matching. The remainder of this section considers the problem of *constructing* a perfect matching, if one exists.

A Self-Reducibility Algorithm. Lovász's observation yields the following simple algorithm to construct a perfect matching in $O(n^{\omega+2})$ time. For each edge $\{i,j\}$, temporarily delete it and test if the resulting graph still has a perfect matching. If so, delete the edge permanently; otherwise, restore the edge. The test used in this algorithm is performed by setting $t_{\{i,j\}} = 0$ and checking whether the determinant of the resulting matrix is non-zero.

Rabin and Vazirani's Improvement. Two definitions are needed. The *inverse Tutte matrix* is $N := T^{-1}$, and an edge $e = \{i,j\}$ is called *allowed* if e is contained in a perfect matching. Rabin and Vazirani [46] showed the following useful lemma.

Lemma 3.1. *Assuming that G has a perfect matching, edge e is allowed iff $N_{i,j} \neq 0$.*

We observed the following simple proof of their lemma. Actually, this argument is precisely equation (2) of Tutte [52], as was pointed out to us by J. F. Geelen.

Proof. Edge e is allowed iff $G[S \setminus \{i,j\}]$ has a perfect matching. (Note that the two vertices i and j are deleted here, not just the edge $\{i,j\}$.) $G[S \setminus \{i,j\}]$ has a perfect matching iff $\det T_{\mathrm{del}(\{i,j\},\{i,j\})}$ is non-zero (by Tutte's theorem). This determinant is

$$\begin{aligned}\det T_{\mathrm{del}(\{i,j\},\{i,j\})} &= \pm \det T \cdot \det N[\{i,j\}, \{i,j\}] \\ &= \pm \det T \cdot (N_{i,j})^2.\end{aligned}$$

The first equality follows from Fact 1. The second follows since Fact 5 shows that N is skew-symmetric, so its diagonal entries are zero. These observations prove the lemma. ■

This lemma yields a more efficient self-reducibility algorithm to construct a perfect matching. First compute $N = T^{-1}$, thereby identifying all allowed edges. Next, add one allowed edge $\{i,j\}$ to the matching, then recurse on the subgraph $G[S \setminus \{i,j\}]$. This algorithm performs a matrix inversion in each recursive step, and therefore uses $O(n^{\omega+1})$ time in total.

Rank-1 Updates. The bottleneck of the Rabin-Vazirani algorithm is recomputing N from scratch in each recursive step. Mucha and Sankowski showed that this is unnecessary; instead, N can be updated using rank-1 updates. To see this, suppose that edge $\{i,j\}$ is added to the matching. The algorithm recurses on the subgraph $G[S \setminus \{i,j\}]$, and must compute the inverse Tutte matrix N' for this subproblem. One might naively expect that N' is $N_{\mathrm{del}(\{i,j\},\{i,j\})}$, but this is not the case. Instead, N' can be determined from Fact 3: take $M = T$, $W = T_{\mathrm{del}(\{i,j\},\{i,j\})}$ and $\hat{Z} = N[\{i,j\}, \{i,j\}]$. Then

$$N' = (T_{\mathrm{del}(\{i,j\},\{i,j\})})^{-1} = W^{-1} = \hat{W} - \hat{X}\hat{Z}^{-1}\hat{Y}.$$

Algorithm 1: *The divide-and-conquer approach to construct a perfect matching.*

```
FindPerfectMatching($G$)
  Construct $T$ and assign random values to the indeterminates
  Compute $N = T^{-1}$
  FindAllowedEdges($S$), where $S$ is the vertex set of $G$

FindAllowedEdges($S$)
  If $|S| > 2$ then
    Partition $S$ arbitrarily into $\alpha$ equal-sized parts $S_1, \ldots, S_\alpha$
    For each unordered pair $\{S_a, S_b\}$ of parts
      FindAllowedEdges($S_a \cup S_b$)
      Update $N$ (if necessary)
  Else
    This is a base case: $S$ consists of two vertices $i$ and $j$
    If $T_{i,j} \neq 0$ and $N_{i,j} \neq 0$ (i.e., edge $\{i,j\}$ is allowed) then
      Add $\{i,j\}$ to the matching and update $N$
```

As observed above, the matrix $N[\{i,j\},\{i,j\}] = \hat{Z}$ is skew-symmetric. Therefore, for some scalar c and column vectors u_i, u_j, v_i, v_j, we have

$$\begin{aligned}\hat{X}\hat{Z}^{-1}\hat{Y} &= \begin{pmatrix} | & | \\ u_i & u_j \\ | & | \end{pmatrix} \cdot \begin{pmatrix} 0 & c \\ -c & 0 \end{pmatrix} \cdot \begin{pmatrix} \text{---} & v_i^\mathsf{T} & \text{---} \\ \text{---} & v_j^\mathsf{T} & \text{---} \end{pmatrix} \\ &= \begin{pmatrix} | & | \\ -c\cdot u_j & c\cdot u_i \\ | & | \end{pmatrix} \cdot \begin{pmatrix} \text{---} & v_i^\mathsf{T} & \text{---} \\ \text{---} & v_j^\mathsf{T} & \text{---} \end{pmatrix} \\ &= -c u_j v_i^\mathsf{T} + c u_i v_j^\mathsf{T}. \end{aligned} \tag{3.1}$$

Thus N' can be computed from N by two rank-1 updates, whose parameters are simple submatrices of N. This computation requires only $O(n^2)$ time.

Modifying the Rabin-Vazirani algorithm to use rank-1 updates, one obtains a simple, $O(n^3)$ time algorithm for constructing perfect matchings. Furthermore, this algorithm uses only naive matrix multiplication. The key question is: how can fast matrix multiplication be used to improve this algorithm?

Our recursive approach. We now describe an algorithm that achieves running time $O(n^\omega)$ via a simple divide-and-conquer approach. The pseudocode in Algorithm 1 outlines our algorithm, but for now we postpone the discussion of how to update N. The constant α will be specified later and has value at least 3. By standard arguments, one may assume that $|S|$ is a multiple of α.

A crucial observation is that Algorithm 1 considers each pair of vertices in at least one base case. The proof is an easy inductive argument: fix a pair of vertices $\{i,j\}$, and note that at each level of the recursion, at least one unordered pair of parts $\{S_a, S_b\}$ has $\{i,j\} \subseteq S_a \cup S_b$. The correctness of Algorithm 1 follows from correctness of the Rabin-Vazirani algorithm: both algorithms simply search for allowed edges, then update the matrix N after one is found. The only difference is that our algorithm considers edges in an unusual order.

Algorithm 2: *The naive scheme to update N during a base case of Algorithm 1.*

```
Set $U[*,\{i,j\}] = N[*,\{i,j\}]$
Set $V[\{i,j\},*] = N[\{i,j\},*]$
Set $C_{i,j} = -1/N_{j,i}$ and $C_{j,i} = -1/N_{i,j}$
Set $N = N + C_{i,j}\, U_{*,i}\, V_{j,*} + C_{j,i}\, U_{*,j}\, V_{i,*}$
Append $i$ and $j$ to $\pi_c$, and append $j$ and $i$ to $\pi_r$
```

Analysis. Let us suppose for now that the updating scheme requires only $O(s^{\omega})$ time for a subproblem with s vertices; this will be demonstrated later. For a subproblem with s vertices, Algorithm 1 recurses on $\binom{\alpha}{2}$ subproblems, each with $\frac{2s}{\alpha}$ vertices. After solving each subproblem, the algorithm performs an update. The total time required satisfies the recurrence

$$h(s) \;=\; \binom{\alpha}{2} \cdot h\Big(\frac{s}{\alpha/2}\Big) \;+\; O\Big(\binom{\alpha}{2} \cdot s^{\omega}\Big). \tag{3.2}$$

By standard arguments, the solution of this recurrence is $h(n) = O(n^{\omega})$ if α is a constant chosen such that $\log_{\alpha/2}\binom{\alpha}{2} < \omega$. Since $\log_{\alpha/2}\binom{\alpha}{2} < 2 + \frac{1}{\log_2 \alpha - 1}$, there exists an appropriate choice of α, assuming that $\omega > 2$. Assuming that $\omega = 2.38$, the choice $\alpha = 13$ is appropriate.

We now describe a slight variant of the algorithm which is preferable for implementations, and also admits an tighter analysis. The key observation is that Algorithm 1 may recurse into the same subproblem multiple times, and this is completely unnecessary. This issue can be avoided via dynamic programming: simply maintain a bit vector indicating which subproblems have already been solved. (Note that the queries and updates to the bit vector do not depend on the actual input, only on n.) Let us analyze this scheme with $\alpha = 4$. At level i of the recursion, the size of a subproblem is $n2^{-i}$ and the number of subproblems is $\binom{2^{i+1}}{2} \leq 2^{2i+1}$. The total time to apply updates at level i is $O\big((n2^{-i})^{\omega} \cdot 2^{2i}\big) = O(n^{\omega} 2^{-(\omega-2)i})$. Summing over all levels yields a bound of $O(n^{\omega})$ if $\omega > 2$ and $O(n^2 \log n)$ if $\omega = 2$. In contrast, the recurrence of Eq. (3.2) leads to a bound of $O(n^{2+\epsilon})$ for any $\epsilon > 0$, in the case that $\omega = 2$.

Naive Updates. We now describe the scheme for updating the matrix N in Algorithm 1. To begin, imagine a naive scheme which uses rank-1 updates to completely update N in the base cases, as in Eq. (3.1), and does not update N after each recursive call. There are $O(n)$ updates, each requiring $O(n^2)$ time, and therefore the resulting algorithm uses time $O(n^3)$ in total.

Ultimately we will define a more efficient updating scheme. Before doing so, let us modify the naive scheme by defining some additional memory areas which will store the parameters of the updates. Consider a single rank-1 update performed by the naive scheme (cf. Eq. (3.1)) when edge $\{i,j\}$ is added to the matching. The parameters of the update are a scalar $c = -1/N_{j,i}$, a column-vector $u = N_{*,i}$ and a row-vector $v^{\mathsf{T}} = N_{j,*}$. As indicated in Algorithm 2, the parameters of all updates are stored in three additional $n \times n$ matrices U, V and C, which are initially zero. The algorithm also maintains two lists π_c and π_r which specify, for each k, which column of U and

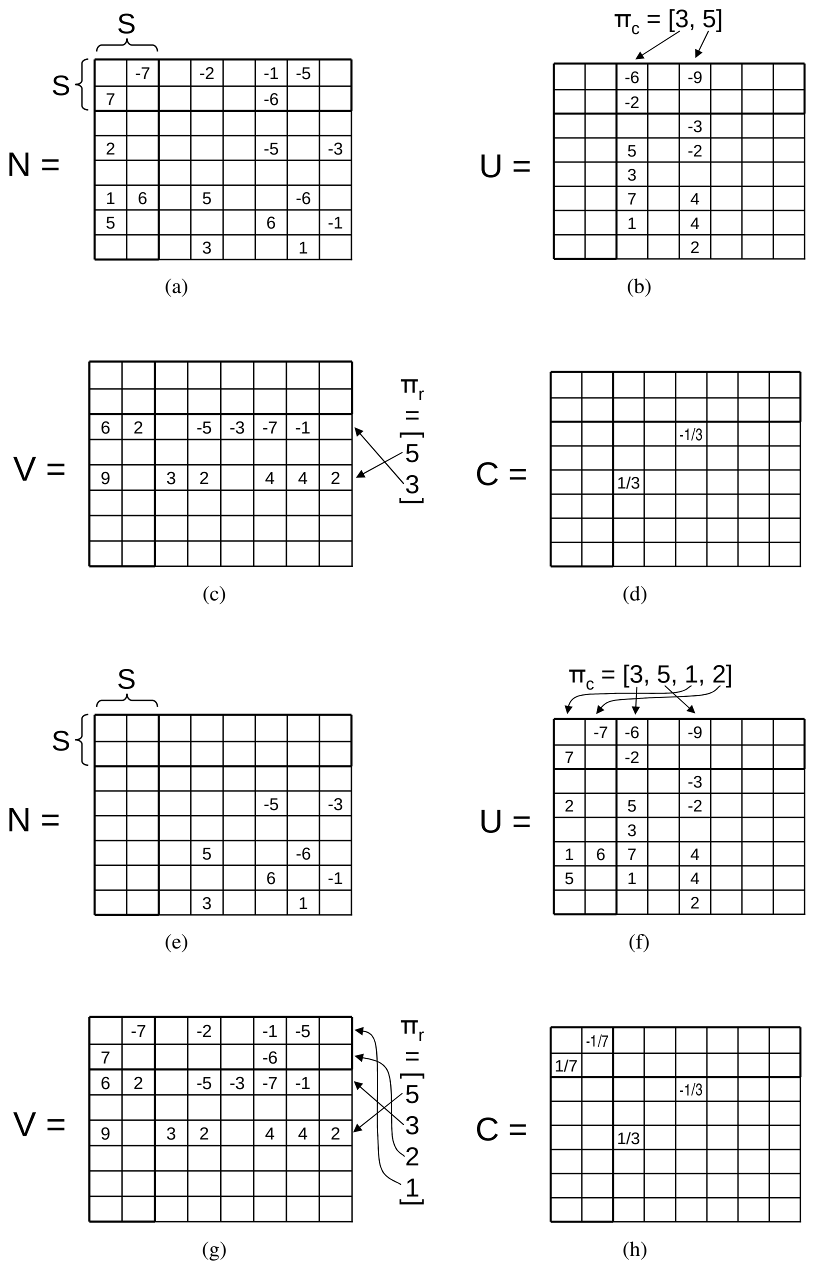


Figure 1: An illustration of the matrices used in the algorithm. *(a)-(d)* The algorithm has already added the edge $\{3, 5\}$ to the matching. This means that there has been a rank-1 update involving column 3 and row 5, and another one involving column 5 and row 3. Thus columns 3 and 5 of U are non-zero, as are rows 3 and 5 of V. The vectors π_c and π_r indicate the order in which the updates were generated. The algorithm has recursed into the set $S = \{1, 2\}$. Since $N_{1,2} \neq 0$, the edge $\{1, 2\}$ is allowed and therefore added to the matching. *(e)-(h)* The matrices after applying the updating scheme of Algorithm 2.

row of V store the parameters of the k^{th} rank-1 update. Figure 1 illustrates the updating procedure.

The updates performed by Algorithm 2 have a property that will be useful later: when vertex i is matched, $N_{*,i}$ and $N_{i,*}$ are set to zero. To see this, note that for any k, the entry $N_{k,i}$ is set to

$$N_{k,i} \;-\; \tfrac{1}{N_{j,i}} N_{k,i} N_{j,i} \;-\; \tfrac{1}{N_{i,j}} N_{k,j} N_{i,i} \;=\; 0; \tag{3.3}$$

the equality holds since $N_{i,i} = 0$ by skew-symmetry.

The reader may have noted that $V = -U^{\mathsf{T}}$ throughout the algorithm, so storing both matrices seems unnecessary. We include both matrices in the discussion here since later sections will discuss algorithms in which the matrices U and V are unrelated.

Efficient Updates. We now describe the efficient scheme which only updates the portions of the matrix which will be needed soon. The recursion of Algorithm 1 gives a convenient way to decide which portions should be updated. The idea is simple: whenever a recursive subproblem finishes executing, it fully updates the submatrix of N corresponding to its parent subproblem. As will be explained shortly, this update requires only a constant number of matrix multiplications/inversions involving matrices of size at most s, which is the number of vertices in the current subproblem. This justifies our earlier assumption that the updating scheme requires $O(s^{\omega})$ time after each recursive call.

To describe the efficient updating scheme more formally, we need some terminology. At any point of the algorithm, we say that a submatrix (of N, U, etc.) is *clean* if its entries are identical to those that the naive scheme would have computed at this point of the algorithm. The efficient updating scheme maintains the following invariant.

Invariant: When each recursive subproblem begins or completes, the parent's submatrices of N, U and V are clean. The matrix C is always clean.

When a base case performs an update, our efficient scheme behaves similarly to Algorithm 2. The key difference is that it need not update large portions of the matrices. Instead, it only updates the 2×2 submatrices corresponding to this base case: $U[\{i,j\},\{i,j\}]$, $V[\{i,j\},\{i,j\}]$, etc. This requires only $O(1)$ time and is sufficient to maintain the invariant for the moment. The remainder of the update work will be performed later (by the recursive ancestors).

After each child subproblem completes, we must perform additional updates in order to maintain the invariant. For notational convenience, we will assume that the parent subproblem is in fact the root of the recursion. The matrices N, U, and V can be decomposed as:

$$N = \begin{pmatrix} \boldsymbol{N_{\mathrm{NW}}} & N_{\mathrm{NE}} \\ N_{\mathrm{SW}} & N_{\mathrm{SE}} \end{pmatrix} \qquad U = \begin{pmatrix} \boldsymbol{U_{\mathrm{NW}}} & \boldsymbol{U_{\mathrm{NE}}} \\ U_{\mathrm{SW}} & \boldsymbol{U_{\mathrm{SE}}} \end{pmatrix} \qquad V = \begin{pmatrix} \boldsymbol{V_{\mathrm{NW}}} & V_{\mathrm{NE}} \\ \boldsymbol{V_{\mathrm{SW}}} & \boldsymbol{V_{\mathrm{SE}}} \end{pmatrix}$$

where the north-west submatrices correspond to the child subproblem that has just completed. The matrix C is decomposed analogously. The submatrices shown in bold are clean. For N_{NW}, U_{NW} and V_{NW}, this follows from our invariant: the child subproblem that just completed has updated them. For U_{NE}, U_{SE}, V_{SW} and V_{SE}, this follows because the invariant ensures that they were clean before executing the child subproblem, and they are not modified during the child subproblem.

We now explain how to update the dirty submatrices. First, consider U_{SW}. Ideally, one would just copy into U_{SW} the columns from N_{SW} corresponding to new updates generated during the child subproblem. The difficulty is that these new updates have dependencies: columns of N_{SW} involved in the j^{th} update should have been modified by the i^{th} update (if $i < j$), but this work was postponed. The following lemma gives the key to resolving these dependencies.

Lemma 3.2. *Let X and Y be $n \times n$ matrices where Y is strictly upper triangular. Define a sequence of matrices by $X^{(0)} = X$ and $X^{(i)} = X^{(i-1)} + X^{(i-1)}_{*,i} \cdot Y_{i,*}$ for $1 \leq i \leq n$. Let $X \circledast Y$ denote $X^{(n)}$. Then $X \circledast Y = X \cdot (I - Y)^{-1}$.*

We use this lemma as follows. Let $X = N_{\mathrm{SW}}$ and let $Y = C_{\mathrm{NW}} \cdot V_{\mathrm{NW}}$. Next, permute columns of X and rows of Y using π_c and π_r so that $X_{*,i}$ and $Y_{i,*}$ correspond to the i^{th} new update generated during the child subproblem. The rows of Y that don't correspond to new updates are set to zero. Note that Y is strictly upper triangular since Eq. (3.3) shows $N_{*,i}$ is set to zero when column i participates in an update (i.e., when vertex i is matched). Therefore X and Y satisfy[1] the hypotheses of Lemma 3.2. The matrix $X \circledast Y$ is, by definition, the result of sequentially applying all new updates to N_{SW}. So, to make N_{SW} and U_{SW} clean, we do the following. First, set $N_{\mathrm{SW}} = X \circledast Y$. Next, the columns from N_{SW} corresponding to new updates are copied into U_{SW} and set to zero.

A symmetric argument shows how to make N_{NE} and V_{NE} clean. It remains to apply the new updates to N_{SE}. This is straightforward since the parameters of these updates have now been fully computed. Let $\tilde{U}$, $\tilde{C}$ and $\tilde{V}$ denote the submatrices of U_{SW}, C_{NW} and V_{NE} corresponding to the new updates. We make N_{SE} clean by setting $N_{\mathrm{SE}} = N_{\mathrm{SE}} + \tilde{U}\tilde{C}\tilde{V}$. All submatrices of the parent subproblem are now clean, and therefore the invariant has been restored. Notice that this update procedure requires only a constant number of matrix multiplications/inversions involving matrices of size s, where s is the number of vertices in the parent subproblem. Thus $O(s^{\omega})$ time suffices.

3.1 Extensions

The algorithm presented above is a Monte Carlo algorithm for finding a perfect matching. By existing techniques [40, 46], the algorithm can extended to construct a maximum cardinality matching, without increasing the asymptotic running time. The first step is to find a full-rank principal submatrix of the Tutte matrix. Next, our perfect matching algorithm is applied to this submatrix.

Another possible improvement to the algorithm is to make it Las Vegas instead of Monte Carlo. The key idea is to efficiently construct an optimum dual solution, i.e., the Gallai-Edmonds decomposition. Karloff [33] showed that this can be done by algebraic techniques, and Cheriyan [7] showed that $O(n^{\omega})$ time suffices.

[1]Actually N_{SW} is only square if $\alpha = 4$, but X and Y can be made square by padding them with zeros.

4 Matroid Intersection

Matroids are combinatorial objects first introduced by Whitney [55] and others in the 1930s. Many excellent texts contain an introduction to the subject [8, 36, 45, 54]. Schrijver [48] and Murota [42] contain more technical material relating to the use of matroids in combinatorial optimization.

4.1 General Definitions

A *matroid* is a combinatorial object defined on a finite ground set S. There are several important ancillary objects relating to matroids, any one of which can be used to define matroids. Below we list those objects that play a role in this paper, and we use "base families" as the central definition.

Base family. This non-empty family $\mathcal{B} \subseteq 2^S$ satisfies the axiom:

> Let $B_1, B_2 \in \mathcal{B}$. For each $x \in B_1 \setminus B_2$, there exists $y \in B_2 \setminus B_1$ such that $B_1 - x + y \in \mathcal{B}$.

A matroid can be defined as a pair $\mathbf{M} = (S, \mathcal{B})$, where $\mathcal{B}$ is a base family over S. A member of $\mathcal{B}$ is called a *base*. It follows from the axiom above that all bases are equicardinal. This cardinality is called the *rank of the matroid* $\mathbf{M}$.

Independent set family. This family $\mathcal{I} \subseteq 2^S$ is defined as

$$\mathcal{I} \;=\; \{\, I \,:\, I \subseteq B \text{ for some } B \in \mathcal{B} \,\}\,.$$

A member of $\mathcal{I}$ is called an *independent set*. Any subset of an independent set is clearly also independent, and a maximum-cardinality independent set is clearly a base.

Rank function. This function, $r : 2^S \to \mathbb{N}$, is defined as

$$r(T) \;=\; \max_{I \in \mathcal{I},\, I \subseteq T} |I|.$$

A maximizer of this expression is called a *base for T* in $\mathbf{M}$. A set I is independent iff $r(I) = |I|$.

Since all of the objects listed above can be used to characterize matroids, we sometimes write $\mathbf{M} = (S, \mathcal{I})$, or $\mathbf{M} = (S, \mathcal{I}, \mathcal{B})$, etc. To emphasize the matroid associated to one of these objects, we often write $\mathcal{B}_{\mathbf{M}}$, $r_{\mathbf{M}}$, etc.

A *linear representation over* $\mathbb{F}$ of a matroid $\mathbf{M} = (S, \mathcal{I})$ is a matrix Q over $\mathbb{F}$ with columns indexed by S, satisfying the condition that $Q[*, I]$ has full column-rank iff $I \in \mathcal{I}$. There do exist matroids which do not have a linear representation over any field. However, many interesting matroids can be represented over some field; such matroids are called *linear matroids*.

Let $\mathbf{M}_1 = (S_1, \mathcal{B}_1)$ and $\mathbf{M}_2 = (S_2, \mathcal{B}_2)$ be two matroids where $S_1 \cap S_2 = \emptyset$. Their *direct sum*, denoted $M_1 \oplus M_2$, has ground set $S_1 \cup S_2$ and base family

$$\mathcal{B} \;=\; \{\, B_1 \cup B_2 \,:\, B_1 \in \mathcal{B}_1 \quad \text{and} \quad B_2 \in \mathcal{B}_2 \,\}\,.$$

The *free matroid* on a set S, denoted $\mathbf{F}(S)$, is defined to be $(S, \mathcal{B})$ where $\mathcal{B} = \{S\}$.

Let $\mathbf{M} = (S, \mathcal{B})$ be a matroid. Given a set $T \subseteq S$, we may define the *contraction* of $\mathbf{M}$ by T. The contracted matroid, denoted $\mathbf{M}/T$, has ground set $S \setminus T$. To define this matroid, first fix a base B_T for T in $\mathbf{M}$. (So $B_T \subseteq T$ and $r_{\mathbf{M}}(T) = r_{\mathbf{M}}(B_T)$.) The base family of $\mathbf{M}/T$ is defined as:

$$B \in \mathcal{B}_{\mathbf{M}/T} \iff B \cup B_T \in \mathcal{B}_{\mathbf{M}}.$$

The rank function of $\mathbf{M}/T$ satisfies: $r_{\mathbf{M}/T}(X) = r_{\mathbf{M}}(X \cup T) - r_{\mathbf{M}}(T)$.

4.2 Assumptions

To specify a matroid requires space that is exponential in the size of the ground set [34] [54, §16.6]. In this case, many matroid problems trivially have an algorithm whose running time is polynomial in the input length. This observation motivates the use of the oracle model for matroid algorithms. However, most of the matroids arising in practice actually can be stored in space that is polynomial in the size of the ground set. The broadest such class is the class of linear matroids, mentioned above.

The algebraic approach used in this paper works only for linear matroids, as do some existing algorithms [4, 5, 37, 44]. One additional assumption is needed, as in this previous work. We assume that the given pair of matroids are represented as matrices over the same field. Although there exist matroids for which this assumption cannot be satisfied (e.g., the Fano and non-Fano matroids), this assumption is valid for the vast majority of matroids arising in applications. For example, the regular matroids are those that are representable over all fields; this class includes the graphic, cographic and partition matroids. Many classes of matroids are representable over all but finitely many fields; these include the uniform, matching, and transversal matroids, as well as deltoids and gammoids [48]. Our results apply to any two matroids from the union of these classes.

4.3 Matroid intersection.

Let two matroids $\mathbf{M}_1 = (S, \mathcal{B}_{\mathbf{M}_1})$ and $\mathbf{M}_2 = (S, \mathcal{B}_{\mathbf{M}_2})$ be given. A set $B \subseteq S$ is called a *common base* if $B \in \mathcal{B}_{\mathbf{M}_1} \cap \mathcal{B}_{\mathbf{M}_2}$. A *common independent set* (or an *intersection*) is a set $I \in \mathcal{I}_{\mathbf{M}_1} \cap \mathcal{I}_{\mathbf{M}_2}$. The *matroid intersection problem* is to find a common base. Alternatively, one can define matroid intersection to be the problem of finding a maximum cardinality intersection. In the context of the matroid intersection problem, it is convenient to use the shorthand $\mathcal{B}_1$ instead of $\mathcal{B}_{\mathbf{M}_1}$, and r_1 instead of $r_{\mathbf{M}_1}$, etc.

Any subset of a maximum cardinality intersection is called an *extensible* set. If J is extensible, $i \notin J$, and $J + i$ is also extensible then element i is called *allowed* (relative to J). Let $\lambda(J)$ denote the maximum cardinality of an intersection in $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$.

The general idea of our algorithm is similar to the matching algorithm in Section 3. High-level pseudocode is presented in Algorithm 3. The crucial step is to decide if element i is allowed. The following section explains how this is done.

Algorithm 3: *A general overview of our algorithm for constructing a common base of two matroids* $\mathbf{M}_1 = (S, \mathcal{B}_1)$ *and* $\mathbf{M}_2 = (S, \mathcal{B}_2)$*.*

```
MatroidIntersection($\mathbf{M}_1$, $\mathbf{M}_2$)
  Set $J = \emptyset$.
  For each $i \in S$, do
    Invariant: $J$ is extensible.
    Test if $i$ is allowed (relative to $J$).
    If so, set $J := J + i$.
```

4.4 Formulation using Linear Algebra

Suppose that each $\mathbf{M}_i$ is a linear matroid representable over a common field $\mathbb{F}$. Let Q_1 be an $r \times n$ matrix whose columns represent $\mathbf{M}_1$ over $\mathbb{F}$ and let Q_2 be a $n \times r$ matrix whose rows represent $\mathbf{M}_2$ over $\mathbb{F}$. For notational convenience, we will let Q_1^J denote $Q_1[*, J]$ and Q_2^J denote $Q_2[J, *]$. Let T be a diagonal matrix where $T_{i,i}$ is an indeterminate t_i. For convenience, let $T(J)$ denote $T_{\mathrm{del}(J,J)}$. For each $J \subseteq S$, we define the matrix

$$Z(J) \;:=\; \begin{pmatrix} & Q_1^J & Q_1^{\bar{J}} \\ Q_2^J & & \\ Q_2^{\bar{J}} & & T(J) \end{pmatrix}. \tag{4.1}$$

Theorem 4.1. *For any* $J \subseteq S$*, we have* $\operatorname{rank} Z(J) = n + r_1(J) + r_2(J) - |J| + \lambda(J)$*.*

For the special case $J = \emptyset$, this result was stated by Geelen [27] and follows easily from the connection between matroid intersection and the Cauchy-Binet formula, as noted by Tomizawa and Iri [51]. Building on Theorem 4.1, we obtain the following result which is crucial to our algorithm. Let us now assume that both $\mathbf{M}_1$ and $\mathbf{M}_2$ have rank r. That is, $r := r_1(S) = r_2(S)$.

Theorem 4.2. *Suppose that* $\lambda(\emptyset) = r$*, i.e.,* $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *have a common base. Then* $Z(J)$ *is non-singular iff* J *is an extensible intersection.*

The preceding theorems lead to the following lemma which characterizes allowed elements. Here, we identify the elements of $S \setminus J$ with the rows and columns of the submatrix of $T(J)$ in $Z(J)$.

Lemma 4.3. *Suppose that* $J \subseteq S$ *is an extensible intersection and that* $i \in S \setminus J$*. The element* i *is allowed iff* $(Z(J)^{-1})_{i,i} \neq t_i^{-1}$*.*

Proof. By Theorem 4.2, our hypotheses imply that $Z(J)$ is non-singular. By linearity of the determinant, $\det Z(J+i) = \det Z(J) - t_i \cdot \det Z(J)_{\mathrm{del}(i,i)}$. By Fact 1, $(Z(J)^{-1})_{i,i} = \det Z(J)_{\mathrm{del}(i,i)} / \det Z(J)$, so we have $\det Z(J+i) = \det Z(J) \cdot (1 - t_i \cdot (Z(J)^{-1})_{i,i})$. Thus $\det Z(J+i) \neq 0 \iff Z(J)^{-1}_{i,i} \neq t_i^{-1}$. By Theorem 4.2, this holds iff element i is allowed. ■

For simplicity, let $Z = Z(\emptyset)$. The structure of Z will play a key role in our algorithm for matroid intersection below. Let Y denote the Schur complement of T in Z,

Algorithm 4: *The naive algorithm to compute a common base of two matroids* $\mathbf{M}_1 = (S, \mathcal{B}_1)$ *and* $\mathbf{M}_2 = (S, \mathcal{B}_2)$*. An inefficient scheme is used to compute and update* Z^{-1}*.*

```
MatroidIntersection($\mathbf{M}_1$,$\mathbf{M}_2$)
  Construct $Z$ and assign random values to the indeterminates $t_1, \ldots, t_n$
  Compute $Z^{-1}$ and let $N$ be its south-east submatrix
  Set $J := \emptyset$
  FindAllowedElements($S$), where $S$ is the ground set of $\mathbf{M}_1$ and $\mathbf{M}_2$

FindAllowedElements($S = \{a, \ldots, b\}$)
  Invariant 1: $J$ is an extensible intersection
  Invariant 2: $N_{a:n,\, a:n}$ is the south-west submatrix of $Z(J)^{-1}$
  If $|S| \geq 2$ then
    Partition $S$ into two equal-sized parts $S_1$ and $S_2$
    FindAllowedElements($S_1$)
    FindAllowedElements($S_2$)
  Else
    This is a base case: $S$ consists of a single element $i$
    If $N_{i,i} \neq t_i^{-1}$ (i.e., element $i$ is allowed) then
      Set $J := J + i$
      Set $U_{i+1:n,\, i} := N_{i+1:n,\, i}$
      Set $V_{i,\, i+1:n} := N_{i,\, i+1:n}$
      Set $C_{i,i} := \left(t_i^{-1} - N_{i,i}\right)^{-1}$
      Set $N_{i+1:n,\, i+1:n} := N_{i+:n,\, i+1:n} + C_{i,i}\, U_{i+1:n,\, i}\, V_{i,\, i+1:n}$
```

i.e., $Y = -Q_1 \cdot T^{-1} \cdot Q_2$. One may verify that

$$Z^{-1} = \begin{pmatrix} Y^{-1} & -Y^{-1} \cdot Q_1 \cdot T^{-1} \\ -T^{-1} \cdot Q_2 \cdot Y^{-1} & T^{-1} + T^{-1} \cdot Q_2 \cdot Y^{-1} \cdot Q_1 \cdot T^{-1} \end{pmatrix}. \tag{4.2}$$

4.5 Matroid intersection algorithm

In this section we describe our matroid intersection algorithm, which achieves running time $O(nr^{\omega-1})$. The algorithm uses Z and Z^{-1} to implement the key test of Algorithm 3. One difficulty is that Z^{-1} potentially has $\Omega(n^2)$ non-zero entries, so it cannot be explicitly computed — our desired running time is linear in n. To begin, we describe Algorithm 4, which ignores the computational cost of computing and updating Z^{-1}. Later, we describe a more intricate algorithm which computes and updates Z^{-1} while storing it implicitly as suggested by Eq. (4.2).

Naive algorithm. We assume that Z is non-singular, i.e., $\lambda(\emptyset) = r$. We also assume for convenience that both n and r are powers of two. As in Algorithm 1, the first step is to randomly substitute values for the indeterminates from the field $\mathbb{F}$, or a sufficiently large extension. By standard arguments, Z remains non-singular with high probability. Next, we compute Z^{-1} and let N be its south-west submatrix. As shown in Eq. (4.2), we have

$$N = T^{-1} - T^{-1} Q_2 \left(Q_1 T^{-1} Q_2\right)^{-1} Q_1 T^{-1}. \tag{4.3}$$

The algorithm uses a trivial divide-and-conquer approach to examine every element exactly once. When an element i is examined, Invariants 1 & 2 and Lemma 4.3 allow the algorithm to decide if i is allowed. Since only allowed elements are added to J, this ensures that Invariant 1 is maintained throughout the algorithm.

It remains to ensure that Invariant 2 is restored after adding an allowed element. That is, given $Z(J)^{-1}$, we wish to construct the matrix $Z(J+i)^{-1}$. By definition, $Z(J+i)$ is identical to $Z(J)$ except that t_i has been set to 0. This can be expressed as the rank-1 update

$$Z(J+i) \;=\; Z(J) \;-\; t_i e_i e_i^{\mathsf{T}}.$$

Here, e_i is the i^{th} elementary vector, i.e., e_i is 1 in the i^{th} component and zero elsewhere. Using Fact 4, $Z(J+i)^{-1}$ may be computed as follows:

$$Z(J)^{-1} \;-\; \big(-t_i^{-1} + (Z(J)^{-1})_{i,i}\big)^{-1}\,(Z(J)^{-1})_{*,i}\,(Z(J)^{-1})_{i,*}. \tag{4.4}$$

Algorithm 4 stores the parameters of each update in the auxiliary matrices U, C and V, as was done in Algorithm 2. Note that the algorithm does not need to update N in any row or column j with $j \leq i$, since these entries of N will never again be examined by the algorithm. Hence, U is strictly lower-triangular and V is strictly upper triangular.

At the termination of the algorithm, J is an extensible intersection, but there are no more allowed elements, so J must be a common base.

Efficient updates. As in Section 3, the key to obtaining an efficient algorithm is to perform the updates in batches. The recursive structure of the algorithm dictates when updates should be performed. As before, we will say that a submatrix is *clean* if its entries are identical to those that the naive scheme would have computed at this point of the algorithm. Pseudocode for the efficient algorithm is shown in Algorithm 5.

As with the naive algorithm, the base of the recursion must decide if an element i is allowed. Invariant 2 ensures that the entry $N_{i,i}$ is clean, so the correct decision is made, as in Algorithm 4. When element i is added to the intersection, the matrices N, U and V are not fully updated. Instead, only $O(1)$ work is performed to update C, and further update work is postponed to the recursive ancestors.

Consider now a subproblem $S = \{a, \ldots, b\}$ at level i of the recursion tree. The number of elements in this subproblem is $n2^{-i}$. Let its children have entries S_1 and S_2. After subproblem S_1 completes, we must perform some work to restore the invariants. The scenario is illustrated in Figure 2.

The key issue is to ensure that invariant 3 will hold in subproblem S_2 by updating $U[S_2, S_1]$ and $V[S_1, S_2]$. Let A be the set of elements that were added to J before starting subproblem S. Let B be the new elements that were added to J during subproblem S_1. We would like to update $U[S_2, S_1]$ by simply copying $N[S_2, B]$ into $U[S_2, B]$. Unfortunately, $N[S_2, B]$ is dirty so it must also be updated. First, let us consider whether the updates corresponding to elements A have been applied to $N[S_2, B]$.

Case 1: $|S| \leq r$. In this case, invariant 2 ensures that those updates have been applied.

Case 2: $|S| > r$. In this case, no updates have been applied to $N[S_2, B]$. However, invariant 3 implies that $U[S, A]$ and $V[A, S]$ are clean. Therefore the updates corresponding to A may be applied as follows

$$N[S_2, B] \;:=\; N[S_2, B] \;+\; U[S_2, A] \cdot C[A, A] \cdot V[A, B].$$

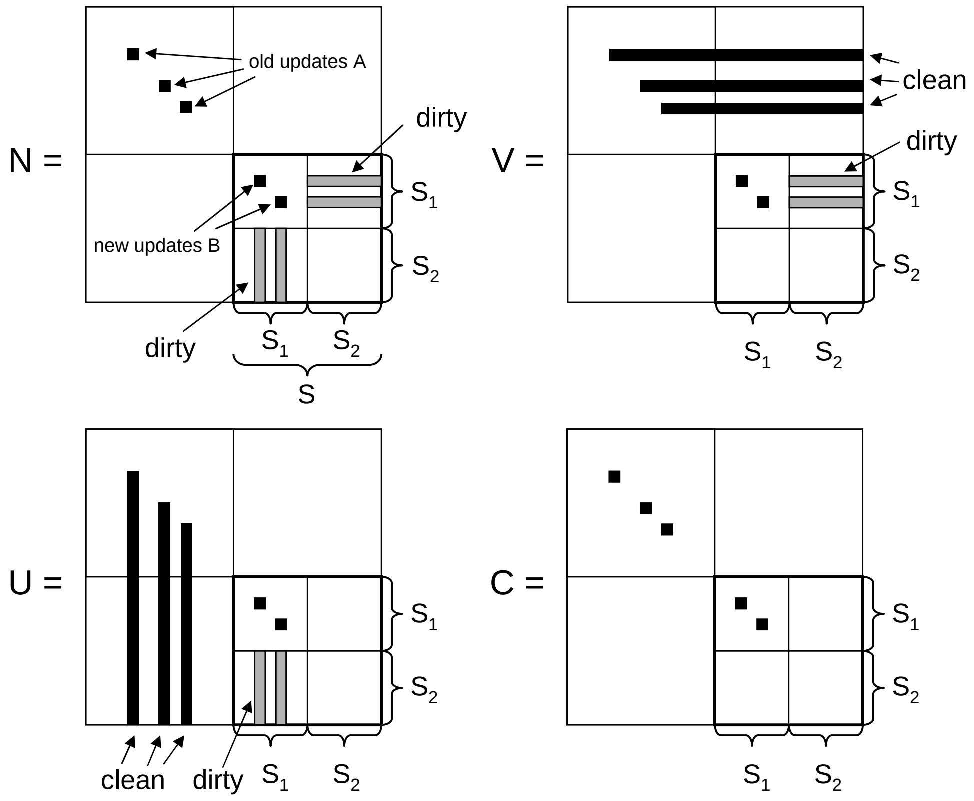


Figure 2: An illustration of the updates in Algorithm 5.

Algorithm 5: *The algorithm to compute a common base of two matroids* $\mathbf{M}_1 = (S, \mathcal{B}_1)$ *and* $\mathbf{M}_2 = (S, \mathcal{B}_2)$*. A more intricate scheme is used to compute and update* Z^{-1}*.*

```
MatroidIntersection($\mathbf{M}_1$,$\mathbf{M}_2$)
  Construct $Z$ and assign random values to the indeterminates $t_1, \ldots, t_n$
  Compute $N$ as in Eq. (4.3)
  Set $J := \emptyset$
  FindAllowedElements($S$), where $S$ is the ground set of $\mathbf{M}_1$ and $\mathbf{M}_2$

FindAllowedElements($S = \{a, \ldots, b\}$)
  *Invariant 1:* $J$ is an extensible intersection
  *Invariant 2:* If $|S| \leq r$ then $N[S, S]$ is clean.
    Otherwise, the $r \times r$ blocks on the diagonal of $N[S, S]$ are clean.
  *Invariant 3:* $U[S, \{1, \ldots, a-1\}]$ and $V[\{1, \ldots, a-1\}, S]$ are clean.
  If $|S| \geq 2$ then
    Partition $S$ into two equal-sized parts $S_1$ and $S_2$
    FindAllowedElements($S_1$)
    Perform updates, as described below
    FindAllowedElements($S_2$)
  Else
    This is a base case: $S$ consists of a single element $i$
    If $N_{i,i} \neq t_i^{-1}$ (i.e., element $i$ is allowed) then
      Set $J := J + i$
      Set $C_{i,i} := -1/(-t_i^{-1} + N_{i,i})$
  *Invariant 4:* $U[S, S]$ is clean and $V[S, S]$ is clean.
```

Recall that $|S_2| < n2^{-i}$ and note that $|A|$ and $|B|$ are both at most r, the rank of the given matroids. Hence, the time required for this update is at most $O(n2^{-i}r^{\omega-1})$.

At this stage, $N[S_2, B]$ is now clean with respect to the time at which subproblem S started. However, further updates are still required: updates for entries in B must be applied to other columns within $N[S_2, S_2]$. If $|S| \leq r$ then invariant 2 requires that each update be applied to all columns to the right within $N[S_2, S_2]$. If $|S| > r$ then actually it suffices to update the columns in $N[S_2, B]$. The two cases are similar so we consider only the latter. The update amounts to setting $N[S_2, B] = X \circledast Y$ where $X = N[S_2, B]$ and $Y = C[B, B]\,V[B, B]$ (cf. Lemma 3.2). (Note that $V[B, B]$ is clean by invariant 4 and strictly upper triangular by construction.) Thus $N[S_2, B]$ is now clean. The time required for this computation is $O(|S|^{\omega})$ if $|S| \leq r$ and $O(n2^{-i}r^{\omega-1})$ otherwise.

Copying $N[S_2, B]$ into $U[S_2, B]$ now makes the latter submatrix clean. A symmetric argument explains how to make $V[B, S_2]$ clean. Thus invariants 1 and 3 will be satisfied at the start of subproblem S_2. To ensure that invariant 2 also holds in subproblem S_2, we set

$$N[S_2, S_2] \;=\; N[S_2, S_2] \;+\; U[S_2, B]\,C[B, B]\,V[B, S_2].$$

This computation requires time at most $O(|S|^{\omega})$. However, if $|S_2| > r$ then it suffices to update the $r \times r$ blocks on the diagonal of $N[S_2, S_2]$, which requires time only $O(n2^{-i}r^{\omega-1})$ by the obvious approach.

Analysis. To analyze the time required by this algorithm, recall that there are 2^i subproblems at level i. For levels $i < \log(n/r)$, the total time required is $\sum_{i=1}^{\log(n/r)} 2^i \cdot O(n2^{-i}r^{\omega-1}) = O(nr^{\omega-1})$, ignoring a $\log n$ factor. For levels $i \geq \log(n/r)$ (i.e., subproblems on at most r elements), the time is

$$\begin{aligned}\sum_{i=\log(n/r)}^{\log n} 2^i \cdot O((n2^{-i})^{\omega}) &= \sum_{i=\log n - \log r}^{\log n} O(n^{\omega}2^{-(\omega-1)i}) \\ &= \sum_{i=0}^{\log r} O(n2^{(\omega-1)i}) \\ &= O(nr^{\omega-1}).\end{aligned}$$

Thus the total time required is $O(nr^{\omega-1})$.

Computing N. The preceding description of the algorithm assumes that N was fully computed at the beginning. However, it is clear that the only parts of N that are needed are those in the $r \times r$ diagonal blocks and those parts that are involved in updates. It is straightforward to extend the algorithm so that the necessary parts of N are computed on demand.

At the beginning of the algorithm, we compute only the $r \times r$ diagonal blocks of N. As shown by Eq. (4.3), this amounts to computing

- $Q_1T^{-1}Q_2$. This requires $O(nr^{\omega-1})$ time.
- $\big(Q_1T^{-1}Q_2\big)^{-1}$. This requires $O(r^{\omega})$ time.
- $\tilde{Q}_1 := \big(Q_1T^{-1}Q_2\big)^{-1}Q_1T^{-1}$. This requires $O(nr^{\omega-1})$ time.
- $\tilde{Q}_2 := T^{-1}Q_2$. This requires $O(nr)$ time.
- The $r \times r$ diagonal blocks of $\tilde{Q}_2\tilde{Q}_1$. This requires $O(nr^{\omega-1})$ time.

The additional work is negligible, so the total amount of initial work is only $O(nr^{\omega-1})$.

During the algorithm, the only time at which entries of N must be computed is during the updating process at any level i where $n2^{-i} > r$. In this case, the entries of $N[S_2, B]$ are needed to make $U[S_2, B]$ clean, and similarly for $V[B, S_2]$. Referring again to Eq. (4.3), we see that

$$N[S_2, B] \;=\; \underbrace{T^{-1}[S_2, B]}_{=\,0} - \tilde{Q}_2[S_2, *] \cdot \tilde{Q}_1[*, B].$$

The time required for this computation is only $O(n2^{-i}r^{\omega-1})$, since $\tilde{Q}_2$ has r columns, $\tilde{Q}_1$ has r rows, $|B| \leq r$, and $|S_2| < n2^{-i}$. Thus computing entries of N on demand does not asymptotically increase the work required for the update process. Thus the preceding analysis applies without change.

4.6 Maximum Cardinality Intersection

The algorithm presented is the previous sections constructs a common base of two matroids, if one exists. If the matroids do not have a common base, one would typically like to find a maximum cardinality intersection. The algorithm can be easily adapted for this purpose, without affecting the running time.

As in Section 3.1, the key idea is to restrict the focus to a full-rank submatrix of the form $Z[A_r, A_c]$ such that A_r and A_c both contain S, i.e., $Z[A_r, A_c]$ contains the entire submatrix T. Such a submatrix may be found in $O(nr^{\omega-1})$ time by the following scheme. First compute $Y = -Q_1 \cdot T^{-1} \cdot Q_2$, which requires $O(nr^{\omega-1})$ time as discussed above. Since Y is the Schur complement of T, Fact 2 shows that the submatrices of Y of rank k correspond to submatrices of Z with the desired form and rank $k+n$. A maximum rank submatrix of Y can be found in $O(r^{\omega})$ time by standard techniques. Redefining $Z := Z[A_r, A_c]$ and $r := |A_r| - n$, the assumption $\lambda(\emptyset) = r$ becomes satisfied.

The algorithm can be made Las Vegas instead of Monte Carlo by constructing an optimum dual solution. Simply construct the usual auxiliary graph used by Lawler's algorithm [8, 36], then find the vertices reachable from the sources; these vertices form an optimal dual solution whp. It is possible to construct the auxiliary graph in time $O(nr^{\omega-1})$, although we omit the details.

5 Generalizations

A nice feature of our previous algorithms is that they extend easily to several generalizations of matching and matroid intersection. We consider two such generalizations here.

5.1 Path-Matchings

An instance of the a basic path-matching is a tuple $G = (R_1, R_2, S, E, \mathbf{M}_1, \mathbf{M}_2)$ where $(R_1 \cup R_2 \cup S, E)$ is a graph and each $\mathbf{M}_i$ is a matroid $(R_i, \mathcal{I}_i, r_i)$. The vertex sets R_1, R_2 and S are disjoint and furthermore R_1 and R_2 are stable sets (no edge has both endpoints in either R_1 or R_2). Assume that $|R_1| = |R_2|$, and that $\mathbf{M}_1$ and $\mathbf{M}_2$ have the same rank r. A *path-matching* is a collection of node-disjoint paths with one endpoint in R_1 and the other in R_2, together with a matching on the S-vertices not contained in any of the paths. If $M \subseteq E$ is a path-matching, let $\partial_i M \subseteq R_i$ denote the set of vertices in R_i that are covered by M, and let $\partial_S M \subseteq S$ denote the covered S-vertices. A *perfect path-matching* is a path-matching M such that $\partial_i M = R_i$ and $\partial_S M = S$. A *basic path-matching* (bpm) is a path-matching such that $\partial_i M \in \mathcal{B}_{\mathbf{M}_i}$ and that $\partial_S M = S$.

Contracted Instances. A set $M \subseteq E$ is called an *extensible set* for G if there exists a bpm $M' \supseteq M$. Let M be an extensible set and note that $\partial_i M \in \mathcal{I}_{\mathbf{M}_i}$. We will now define a basic path-matching problem $G(M)$, which we call the *contraction* of G by M.

Let P_i be the set of paths in M with one endpoint in R_i and the other in S. Let P_{12} be the set of paths with one endpoint in R_2 and the other in R_2. Informally, we delete the paths in P_{12}, we contract (in the graph) each path in P_i to a single vertex, and we contract (in the matroid $\mathbf{M}_i$) the elements $\partial_i M$. Formally, the set $C_i \subseteq S$ consists of the endpoints in S of the paths in P_i. Define $\overline{\partial_i M} = R_i \setminus \partial_i M$, and note that $\partial_1 M$ and $\partial_2 M$ are not necessarily equicardinal. Define $R'_i := \overline{\partial_i M} \cup C_i$ and $S' := S \setminus \partial_S M$. Define $E' \subseteq E$ to be the set of edges with both endpoints in S' or with one endpoint

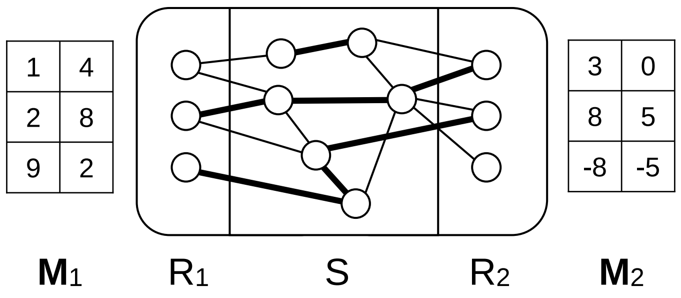


Figure 3: An illustration of a basic path-matching instance. The rows of the matrix on the left represent a matroid on R_1, and the rows of the matrix on the right represent a matroid on R_2. The edges in bold constitute a basic path-matching.

in R'_i and the other in $S' \cup R'_j$ where $i \neq j$. The matroid $\mathbf{M}'_i$ is $(\mathbf{M}_i/\partial_i M) \oplus \mathbf{F}(C_i)$, where $\mathbf{F}(C_i)$ denotes the free matroid on C_i and $\oplus$ denotes direct sum. The contraction of G by M is $G(M) := (R'_1, R'_2, S', E', \mathbf{M}'_1, \mathbf{M}'_2)$. Clearly $G(\emptyset) = G$.

Claim 5.1. *If $M' \supseteq M$ is a bpm for G then $M'' := M' \setminus M$ is a bpm for $G(M)$. Conversely, if M'' is a bpm of $G(M)$ then $M \cup M''$ is a bpm of G.*

Proof. $\Rightarrow$: As above, let P_{12} be the paths from R_1 to R_2 in M. The R_1-R_2 paths in M', excluding the paths in P_{12}, clearly form a path-matching when restricted to the vertex set of $G(M)$. These paths intersect R'_i at $C_i \cup (\partial_i M' \setminus \partial_i M)$. This is clearly a base of $\mathbf{M}'_i$ since $\partial_i M'$ is a base for $\mathbf{M}_i$.

$\Leftarrow$: The key point is that every base of $\mathbf{M}_i$ must contain C_i, so any basic path-matching in $G(M)$ must cover C_i. ■

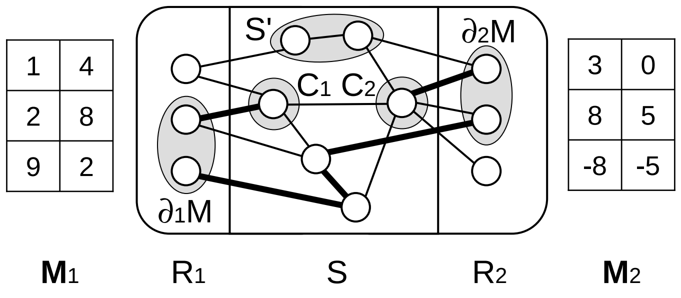
Formulation using Linear Algebra. We now define a matrix which captures the basic path-matching problem by generalizing the Tutte matrix and the matrix presented in Eq. (4.1) for matroid intersection.

First, there is a matrix of indeterminates T which is similar to the Tutte matrix and describes the graph underlying G. The rows of T are indexed by $R_1 \cup S$ and the columns are indexed by $R_2 \cup S$. The entries are defined as $T_{i,j} = \pm t_{\{i,j\}}$, where the signs are chosen such that $T[S,S]$ is skew-symmetric.

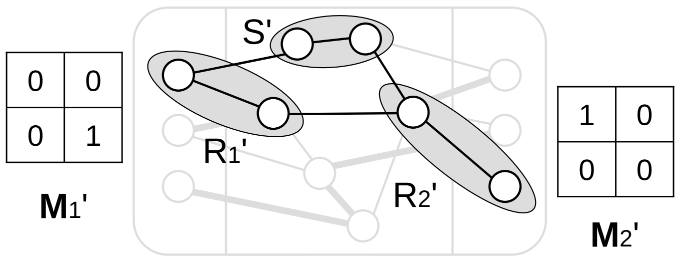
Lemma 5.2 (Geelen [26]). *T is non-singular iff G has a perfect path-matching.*

Let Q_1 and Q_2 be matrices as in Section 4. Let I denote an identity matrix of the appropriate size. Define $Z(M)$ to be the following matrix.

$$Z(M) = \begin{pmatrix} & Q_1^{\partial_1 M} & Q_1^{\overline{\partial_1 M}} & & \\ Q_2^{\partial_2 M} & & & & \\ Q_2^{\overline{\partial_2 M}} & & & I & \\ & & I & T[\overline{\partial_1 M}, \overline{\partial_2 M}] & T[\overline{\partial_1 M}, C_2 \cup S'] \\ & & & T[C_1 \cup S', \overline{\partial_2 M}] & T[C_1 \cup S', C_2 \cup S'] \end{pmatrix}$$



(a)



(b)

Figure 4: *(a)* The bold edges form an extensible set M since they are a subset of the basic path-matching shown in Figure 3. *(b)* The contracted instance.

Theorem 5.3. *Let M be a set of edges in G such that $\partial_1 M \in \mathcal{I}_{\mathbf{M}_1}$ and $\partial_2 M \in \mathcal{I}_{\mathbf{M}_2}$. Then $G(M)$ has a bpm iff $Z(M)$ is non-singular.*

By Claim 5.1, an equivalent statement of Theorem 5.3 is as follows.

Corollary 5.4. *M is an extensible set iff $Z(M)$ is non-singular.*

Allowed Edges. Suppose that M is an extensible set. We say that an edge $e = \{i, j\}$ in $G(M)$ is *allowed* (relative to M) if $M + e$ is also extensible. This section explains how to test efficiently whether e is allowed by deriving an analogue of Lemma 4.3. It will be convenient to identify the vertices $R'_1 \cup S'$ (vertices $R'_2 \cup S'$) with rows (columns) of the submatrix of T in $Z(M)$, and let $N = Z(M)^{-1}$. There are two cases to consider.

Update A: In this case, either i or j is in $R'_1 \cup R'_2$, say $i \in R'_1$ and $j \in R'_2 \cup S'$. Observe that $Z(M+e) = Z(M)_{\mathrm{del}(i,j)}$, modulo a permutation of the rows and columns. Therefore, e is allowed iff $N_{j,i} \neq 0$, since

$$\det Z(M+e) \;=\; \pm \det Z(M)_{\mathrm{del}(i,j)} \;=\; \pm \det Z(M) \cdot N_{j,i}.$$

Furthermore, the matrix $Z(M+e)^{-1}$ may be computed from N by a rank-1 update. From Fact 4, it follows that

$$Z(M+e)^{-1} \;=\; \Big(N \;-\; N_{*,i} \cdot (N_{j,i})^{-1} \cdot N_{j,*}\Big)_{\mathrm{del}(j,i)}. \tag{5.1}$$

Update B: In this case, $\{i, j\} \subseteq S'$. Note that $Z(M+e) = Z(M)_{\mathrm{del}(\{i,j\},\{i,j\})}$. By Fact 1,

$$\det Z(M)_{\mathrm{del}(\{i,j\},\{i,j\})} \;=\; \pm \det Z(M) \cdot \det N[\{i,j\}, \{i,j\}].$$

Therefore, e is allowed iff[2] $\det N[\{i,j\}, \{i,j\}] \neq 0$. It follows from Fact 3 that

$$Z(M+e)^{-1} \;=\; \Big(N \;-\; N_{*,\{i,j\}} \cdot (N_{\{i,j\},\{i,j\}})^{-1} \cdot N_{\{i,j\},*}\Big)_{\mathrm{del}(\{i,j\},\{i,j\})}. \tag{5.2}$$

Algorithm. Given the preceding discussion, the algorithm of Section 3 generalizes straightforwardly to solve the basic path-matching problem. One simply computes the matrix $N := Z(\emptyset)^{-1}$ in $O(n^\omega)$ time, then searches for allowed edges using the recursive scheme of Algorithm 1. To decide whether an edge is allowed, it suffices to examine the entries of N, as described above. Once an allowed edge is found, it is added to the extensible set, and updates are stored in the auxiliary matrices U, V and C as before. For the sake of brevity, we leave the details to the reader.

[2] Although the present discussion is similar to Section 3, it does not in general hold that $N[\{i,j\}, \{i,j\}]$ is skew-symmetric.

5.2 Bipartite Matroid Matchings

The bipartite matroid matching problem is simply the special case of the basic path-matching problem in which $S = \emptyset$. Furthermore, matroid intersection is the special case of bipartite matroid matching where all vertices have degree 1. The preceding discussion immediately yields an algorithm for solving the bipartite matroid matching problem in $O(n^\omega)$ time. However, the algorithm can be simplified since the matrix $Z(M)$ contains no skew-symmetric parts. We discuss this simplification below.

First, note that $Z := Z(\emptyset)$ has the following form.

$$Z = \begin{pmatrix} & Q_1 & \\ Q_2 & & I \\ & I & T \end{pmatrix}$$

Clearly Z^{-1} can be computed in $O(n^\omega)$ time. However, the algorithm only makes use of the south-east submatrix of Z^{-1} (the submatrix whose rows and columns correspond to the submatrix T in Z). This submatrix actually has a simple closed form, as the following lemma shows.

Lemma 5.5. *The south-east submatrix of Z^{-1} is $-Q_2\big(Q_1TQ_2\big)^{-1}Q_1$.*

Now consider how the algorithm described in Section 5.1 operates on this matrix. Since each indeterminate appears only once in T (i.e., T is not skew-symmetric), the problem is more similar to bipartite matching than non-bipartite matching. Thus the recursive structure can be modified as follows. Partition R_1 into $R_{1,1} \cup R_{1,2}$, partition R_2 into $R_{2,1} \cup R_{2,2}$. Next recurse on the four subproblems

$$R_{1,1} \cup R_{2,1}, \qquad R_{1,1} \cup R_{2,2}, \qquad R_{1,2} \cup R_{2,1}, \qquad \text{and} \qquad R_{2,2} \cup R_{2,2}.$$

Each update performed by the algorithm is of type Update A. Thus the resulting algorithm is essentially identical to the bipartite matching algorithm of Mucha and Sankowski [41, 40].

Thus we have the following surprising result. When the Mucha-Sankowski algorithm executes on T and T^{-1}, it computes a maximum bipartite matching. However, when it executes on T and $Q_2(Q_1TQ_2)^{-1}Q_1$ it computes a bipartite matroid matching.

Acknowledgements

The author thanks Michel Goemans, Satoru Iwata, and David Karger for helpful discussions on this topic. Additionally, the author thanks the anonymous referees for suggesting numerous improvements to the text of the paper. This work was supported by a Natural Sciences and Engineering Research Council of Canada PGS Scholarship, by NSF contract CCF-0515221 and by ONR grant N00014-05-1-0148.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] M. Aigner and T. A. Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, July 1971.

[3] A. C. Aitken. *Determinants and Matrices*. Interscience Publishers, New York, ninth edition, 1956.

[4] A. I. Barvinok. New algorithms for linear k-matroid intersection and matroid k-parity problems. *Mathematical Programming*, 69:449–470, 1995.

[5] P. M. Camerini, G. Galbiati, and F. Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992.

[6] P. Chalasani and R. Motwani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 26(6):2133–2149, 1999.

[7] J. Cheriyan. Randomized $\tilde{O}(M(|V|))$ algorithms for problems in matching theory. *SIAM Journal on Computing*, 26(6):1635–1669, 1997.

[8] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley, 1997.

[9] D. Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13(1):42–49, 1997.

[10] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[11] W. H. Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM Journal on Computing*, 15(4):948–957, Nov. 1986.

[12] W. H. Cunningham and J. F. Geelen. Vertex-disjoint directed paths and even circuits. Manuscript.

[13] W. H. Cunningham and J. F. Geelen. The optimal path-matching problem. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 78–85, 1996.

[14] W. H. Cunningham and J. F. Geelen. The optimal path-matching problem. *Combinatorica*, 17(3):315–337, 1997.

[15] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69B:125–130, 1965.

[16] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[17] J. Edmonds. Matroid partition. In G. B. Dantzig and A. F. Veinott Jr., editors, *Mathematics of the Decision Sciences Part 1*, volume 11 of *Lectures in Applied Mathematics*, pages 335–345. American Mathematical Society, 1968.

[18] J. Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and Their Applications*, pages 69–87. Gordon and Breach, 1970. Republished in M. Jünger, G. Reinelt, G. Rinaldi, editors, *Combinatorial Optimization – Eureka, You Shrink!*, Lecture Notes in Computer Science 2570, pages 11–26. Springer-Verlag, 2003.

[19] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971.

[20] J. Edmonds. Matroid intersection. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Discrete Optimization I*, volume 4 of *Annals of Discrete Mathematics*, pages 39–49. North-Holland, 1979.

[21] S. Even and O. Kariv. An $O(n^{2.5})$ algorithm for maximum matching in general graphs. In *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 100–112, 1975.

[22] S. Fujishige. *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics*. Elsevier, second edition, 2005.

[23] H. N. Gabow and Y. Xu. Efficient algorithms for independent assignments on graphic and linear matroids. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 106–111, 1989.

[24] H. N. Gabow and Y. Xu. Efficient theoretic and practical algorithms for linear matroid intersection problems. *Journal of Computer and System Sciences*, 53(1):129–147, 1996.

[25] F. R. Gantmakher. *The Theory of Matrices*, volume 1. Chelsea, New York, 1960. Translation by K.A. Kirsch.

[26] J. F. Geelen. *Matroids, Matchings, and Unimodular Matrices*. PhD thesis, University of Waterloo, Canada, 1995.

[27] J. F. Geelen. Matching theory. Lecture notes from the Euler Institute for Discrete Mathematics and its Applications, 2001.

[28] C. D. Godsil. *Algebraic Combinatorics*. Chapman & Hall, 1993.

[29] M. X. Goemans. Bounded degree minimum spanning trees. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

[30] A. V. Goldberg and A. V. Karzanov. Maximum skew-symmetric flows and matchings. *Mathematical Programming*, 100(3):537–568, July 2004.

[31] N. J. A. Harvey, D. R. Karger, and K. Murota. Deterministic network coding by matrix completion. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 05)*, pages 489–498, 2005.

[32] R. Hassin and A. Levin. An efficient polynomial time approximation scheme for the constrained minimum spanning tree problem using matroid intersection. *SIAM Journal on Computing*, 33(2):261–268, 2004.

[33] H. J. Karloff. A Las Vegas RNC algorithm for maximum matching. *Combinatorica*, 6(4):387–391, 1986.

[34] D. E. Knuth. The asymptotic number of geometrices. *Journal of Combinatorial Theory, Series A*, 16:398–400, 1974.

[35] E. L. Lawler. Matroid intersection algorithms. *Mathematical Programming*, 9:31–56, 1975.

[36] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover, 2001.

[37] L. Lovász. On determinants, matchings and random algorithms. In L. Budach, editor, *Fundamentals of Computation Theory, FCT '79*, pages 565–574. Akademie-Verlag, Berlin, 1979.

[38] L. Lovász and M. D. Plummer. *Matching Theory*. Akadémiai Kiadó – North Holland, Budapest, 1986.

[39] S. Micali and V. V. Vazirani. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 17–27, 1980.

[40] M. Mucha. *Finding Maximum Matchings via Gaussian Elimination*. PhD thesis, Warsaw University, 2005.

[41] M. Mucha and P. Sankowski. Maximum matchings via Gaussian elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 248–255, 2004.

[42] K. Murota. *Matrices and Matroids for Systems Analysis*. Springer-Verlag, 2000.

[43] K. Murota. *Discrete Convex Analysis*. SIAM, 2003.

[44] H. Narayanan, H. Saran, and V. V. Vazirani. Randomized parallel algorithms for matroid union and intersection, with applications to arboresences and edge-disjoint spanning trees. *SIAM Journal on Computing*, 23(2):387–397, 1994.

[45] J. G. Oxley. *Matroid Theory*. Oxford University Press, 1992.

[46] M. O. Rabin and V. V. Vazirani. Maximum matchings in general graphs through randomization. *Journal of Algorithms*, 10(4):557–567, 1989.

[47] P. Sankowski. Processor efficient parallel matching. In *Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 165–170, 2005.

[48] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag, 2003.

[49] A. Schrijver. On the history of combinatorial optimization (till 1960). In K. Aardal, G. L. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 1–68. North Holland, 2005.

[50] G. Strang. *Linear Algebra and its Applications*. Thomson Learning, 1988.

[51] N. Tomizawa and M. Iri. An algorithm for determining the rank of a triple matrix product AXB with application to the problem of discerning the existence of the unique solution in a network. *Electronics and Communications in Japan (Scripta Electronica Japonica II)*, 57(11):50–57, Nov. 1974.

[52] W. T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 22:107–111, 1947.

[53] V. V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph matching algorithm. In *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference (IPCO)*, pages 509–530, 1990.

[54] D. J. A. Welsh. *Matroid Theory*, volume 8 of *London Mathematical Society Monographs*. Academic Press, 1976.

[55] H. Whitney. On the abstract properties of linear dependence. *American Journal of Mathematics*, 57:509–533, 1935.

A Additional Proofs

A.1 Facts from Linear Algebra

This section proves the basic facts that are given in Section 2.

Proof (of Fact 1). The following proof is folklore; see also Gantmakher [25, §1.4]. Let M be of the form $M = \left(\begin{smallmatrix} W & X \\ Y & Z \end{smallmatrix}\right)$ and let $M^{-1} = \left(\begin{smallmatrix} \hat{W} & \hat{X} \\ \hat{Y} & \hat{Z} \end{smallmatrix}\right)$. Note that

$$\begin{pmatrix} W & X \\ Y & Z \end{pmatrix} \cdot \begin{pmatrix} \hat{W} & 0 \\ \hat{Y} & I \end{pmatrix} = \begin{pmatrix} I & X \\ 0 & Z \end{pmatrix}.$$

Taking the determinant of both sides shows that $\det M \cdot \det \hat{W} = \det Z$. This proves the result when $M[I,J]$ is the south-east submatrix Z. The general result follows via row/column permutations. ■

Proof (of Fact 2). See Murota [42]. Note that:

$$\begin{pmatrix} W - XZ^{-1}Y & 0 \\ Z^{-1}Y & I \end{pmatrix} = \begin{pmatrix} I & -X \\ 0 & I \end{pmatrix} \cdot \begin{pmatrix} I & 0 \\ 0 & Z^{-1} \end{pmatrix} \cdot \begin{pmatrix} W & X \\ Y & Z \end{pmatrix}. \tag{A.1}$$

Taking the determinant of both sides proves the fact. ■

Proof (of Fact 3). The condition for non-singularity of W follows from Fact 1. To prove the equation for W^{-1}, we reverse the roles of M and M^{-1}, i.e., we consider $\hat{W}^{-1}$ instead. Take inverses in Eq. (A.1):

$$\begin{aligned} \begin{pmatrix} (W - XZ^{-1}Y)^{-1} & 0 \\ Z^{-1}Y(W - XZ^{-1}Y)^{-1} & I \end{pmatrix} &= \begin{pmatrix} \hat{W} & \hat{X} \\ \hat{Y} & \hat{Z} \end{pmatrix} \cdot \begin{pmatrix} I & 0 \\ 0 & Z \end{pmatrix} \cdot \begin{pmatrix} I & X \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} \hat{W} & \hat{W}X + \hat{X}Z \\ \hat{Y} & \hat{Y}X + \hat{Z}Z \end{pmatrix}. \end{aligned}$$

The equality of the north-west submatrices shows that $\hat{W}^{-1} = W - XZ^{-1}Y$, as desired. ■

Proof (of Fact 4). First we prove the condition for existence of $\tilde{M}^{-1}$. Consider the matrix $A = \left(\begin{smallmatrix} -c^{-1} & v^{\mathsf{T}} \\ u & M \end{smallmatrix}\right)$. Use Fact 2 first on the south-east submatrix M, and then on the north-west submatrix $-c^{-1}$, obtaining:

$$\begin{aligned} &\det M \cdot \det\big(-c^{-1} - v^{\mathsf{T}}M^{-1}u\big) \\ &\quad = \det A = (-c^{-1}) \cdot \det\big(M - u(-c)v^{\mathsf{T}}\big). \end{aligned}$$

This shows that $\tilde{M}$ is non-singular iff $\alpha \neq 0$. To verify the equation for $\tilde{M}^{-1}$, note that

$$\big(M + cuv^{\mathsf{T}}\big) \cdot \big(M^{-1} - \alpha^{-1} M^{-1} u v^{\mathsf{T}} M^{-1}\big) \;=\; I. \qquad \blacksquare$$

Proof (of Fact 5). Suppose that M^{-1} exists. Then

$$\begin{aligned}(M^{-1})_{i,j} &= \big(\,(M^{-1})^{\mathsf{T}}\,\big)_{j,i} = \big(\,(M^{\mathsf{T}})^{-1}\,\big)_{j,i} \\ &= \big(\,(-M)^{-1}\,\big)_{j,i} = -(M^{-1})_{j,i}.\end{aligned} \qquad \blacksquare$$

A.2 Proofs from Section 3

Proof (of Lemma 3.2). First, note that $X^{(j)}_{*,i} = X^{(k)}_{*,i}$ if $i \leq j \leq k$ since Y is strictly upper triangular. Define $A = \big(\begin{smallmatrix} I-Y & 0 \\ X & I \end{smallmatrix}\big)$, and consider performing Gaussian elimination on A. Let $S^{(i)}$ denote the south-west submatrix of A just before the ith elimination. An easy inductive argument shows that $S^{(i)}_{*,\,i:n} = X^{(i)}_{*,\,i:n}$. The (lower half of the) column vector involved in the ith elimination is therefore $S^{(i)}_{*,\,i} = X^{(n)}_{*,\,i}$. Now consider the LU-decomposition of A:

$$\begin{pmatrix} I-Y & 0 \\ X & I \end{pmatrix} = \begin{pmatrix} I & 0 \\ B & I \end{pmatrix} \cdot \begin{pmatrix} I-Y & 0 \\ 0 & I \end{pmatrix}.$$

It is well-known that $B_{*,i}$ is precisely the (lower half of the) column involved in the ith elimination (see, e.g., Strang [50]). Thus $B = X^{(n)} = X \otimes Y$. The lemma follows by observing that $X = B \cdot (I - Y)$. $\blacksquare$

A.3 Proof of Theorem 4.1

The following result is useful for analyzing matrices of indeterminates.

Lemma A.1 (Murota [42, p139]). *Let Q and X be matrices with row-set and column-set $\mathcal{S}$. Suppose that the entries of Q are numbers in some field, and the entries of X are distinct indeterminates. Then*

$$\operatorname{rank}(Q + X) = \max_{A_r \subseteq \mathcal{S},\, A_c \subseteq \mathcal{S}} \big(\operatorname{rank} Q[A_r, A_c] + \operatorname{rank} X[\mathcal{S} \setminus A_r,\, \mathcal{S} \setminus A_c]\big). \quad \text{(A.2)}$$

We now wish to show that $Z(J)$ is non-singular iff J is an extensible intersection.

Proof (of Theorem 4.1). The approach is to apply Lemma A.1 to the matrix Z. Letting $Z = Q + X$, the matrix Q contains the two matrices Q_1 and Q_2 representing the matroids and zeros elsewhere. The matrix X contains the submatrix T and zeros elsewhere. The set $\mathcal{S}$, which indexes the rows and columns of Z, can be regarded as the set $[r] \cup S$, where S is the ground set of the matroids.

Our proof successively adds constraints to the sets A_r and A_c in Eq. (A.2) without changing the maximum value. First, we add the constraint $[r] \cup J \subseteq A_r$ because those rows cannot contribute to $\operatorname{rank} X[\mathcal{S} \setminus A_r, \mathcal{S} \setminus A_c]$. A similar argument holds for A_c.

Next, if $i \in A_r \setminus A_c$ then column i cannot contribute to $\operatorname{rank} X[\mathcal{S} \setminus A_r, \mathcal{S} \setminus A_c]$, since T (and hence X) are diagonal. The same argument applies to $A_c \setminus A_r$, so we may assume without loss of generality that $A_r = A_c$. For notational simplicity, we drop the subscripts and simply write $A = [r] \cup A'$, where $A' \subseteq S$.

Consider the $\operatorname{rank} Q[A, A]$ term of Eq. (A.2). Observe that Q_1 and Q_2 occupy disjoint rows and columns of Z, so $\operatorname{rank} Q[A, A] = r_1(A') + r_2(A')$. The $\operatorname{rank} X[\mathcal{S} \setminus A, \mathcal{S} \setminus A]$ term equals $|\mathcal{S} \setminus A| = |S \setminus A'| = n - |A'|$. Thus we have

$$\operatorname{rank} Z(J) \;=\; \max_{A' \subseteq S} \big(r_1(A') + r_2(A') + n - |A'| \big).$$

Recall that $J \subseteq A'$. Thus may write $A' = J \cup A''$, where $A'' \cap J = \emptyset$. Using the definition of $r_{\mathbf{M}_1/J}$, we obtain

$$\operatorname{rank} Z(J) \;= \\ \max_{A'' \subseteq S \setminus J} \big(\, r_{\mathbf{M}_1/J}(A'') + r_{\mathbf{M}_2/J}(A'') - |A''| \,\big) + n + r_1(J) + r_2(J) - |J|. \qquad \text{(A.3)}$$

Let A be a maximum intersection of $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$, so that $\lambda(J) = |A| = r_{\mathbf{M}_i/J}(A)$ for each i. Then $\operatorname{rank} Z(J) \geq \lambda(J) + n + r_1(J) + r_2(J) - |J|$. To show the reverse inequality, let A be a maximizer of Eq. (A.3) that additionally minimizes $2|A| - r_{\mathbf{M}_1/J}(A) - r_{\mathbf{M}_2/J}(A)$. Suppose that this latter quantity is not zero. Then for some i we have $r_{\mathbf{M}_i/J}(A) < |A|$, so there exists $a \in A$ with $r_{\mathbf{M}_i/J}(A - a) = r_{\mathbf{M}_i/J}(A)$. It follows that the set $A - a$ is also a maximizer of Eq. (A.3), contradicting our choice of A. Hence $r_{\mathbf{M}_i/J}(A) = |A|$ for both i. Thus A is an intersection of $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$ satisfying $\operatorname{rank} Z(J) = |A| + n + r_1(J) + r_2(J) - |J|$. Since $|A| \leq \lambda(J)$, the desired inequality follows. ■

A.4 Proof of Theorem 4.2

Claim A.2. *The function λ is non-increasing, i.e., $\lambda(I) \geq \lambda(J)$ for all sets $I \subseteq J$.*

Proof. Let $U \subseteq W$. Let $B_{i,U}$ be a base for U in $\mathbf{M}_i$, and extend it to a base $B_{i,W}$ for W in $\mathbf{M}_i$. Now suppose that $I \in \mathcal{I}_{\mathbf{M}_1/W} \cap \mathcal{I}_{\mathbf{M}_2/W}$. Then $I \cup B_{i,W} \in \mathcal{I}_{\mathbf{M}_i}\ \forall i \implies I \cup B_{i,U} \in \mathcal{I}_{\mathbf{M}_i}\ \forall i \implies I \in \mathcal{I}_{\mathbf{M}_1/U} \cap \mathcal{I}_{\mathbf{M}_2/U}$. Thus $\lambda(W) \leq \lambda(U)$. ■

Claim A.3. *Let J be an intersection. Then J is extensible iff $\lambda(\emptyset) = \lambda(J) + |J|$.*

Proof. Suppose that J is an extensible intersection. This means that there exists an intersection I with $J \subseteq I$ and $|I| = \lambda(\emptyset)$. Then $I \setminus J$ is an intersection of $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$, implying that $\lambda(J) \geq \lambda(\emptyset) - |J|$. Conversely, let I be an intersection of $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$ with $|I| = \lambda(J)$. Then $I \cup J$ is an intersection of $\mathbf{M}_1$ and $\mathbf{M}_2$, showing that $\lambda(\emptyset) \geq \lambda(J) + |J|$. This establishes the forward direction.

Now suppose that $\lambda(J) = \lambda(\emptyset) - |J|$. Then there exists an intersection I of $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$ with $|I| = \lambda(\emptyset) - |J|$. Then $I \cup J$ is an intersection of $\mathbf{M}_1$ and $\mathbf{M}_2$, of cardinality $\lambda(\emptyset)$. This shows that J is extensible. ■

Define the function $\psi(J) \;:=\; r_1(J) + r_2(J) - |J| + \lambda(J)$.

Claim A.4. *The function ψ is non-increasing.*

Proof. Suppose that $\psi(J+j) > \psi(J)$ for some set J and $j \notin J$. Clearly $r_i(J) \leq r_i(J+j) \leq r_i(J)+1$. By Claim A.2, $\lambda(J+j) \leq \lambda(J)$. Thus $\psi(J+j) > \psi(J)$ implies that, for both i, $r_i(J+j) = r_i(J)+1$. Thus, if $B_{i,J}$ is a base for J in $\mathbf{M}_i$, then $B_{i,J}+j$ is a base for $J+j$. Now suppose that $I \in \mathcal{I}_{\mathbf{M}_1/J+j} \cap \mathcal{I}_{\mathbf{M}_2/J+j}$. This implies that $I \cup (B_{i,J}+j) \in \mathcal{I}_{\mathbf{M}_i}$ $\forall i$. Rewriting, $(I+j) \cup B_{i,J} \in \mathcal{I}_{\mathbf{M}_i}$ $\forall i$. Thus $I+j \in \mathcal{I}_{\mathbf{M}_1/J} \cap \mathcal{I}_{\mathbf{M}_2/J}$, implying that $\lambda(J) \geq \lambda(J+j)+1$. This contradicts our assumption that $\psi(J+j) > \psi(J)$. ■

Claim A.5. *Assume that $\mathbf{M}_1$ and $\mathbf{M}_2$ have the same rank r. For any set $J \subseteq S$, we have $\lambda(J) \leq r - \max_{i\in\{1,2\}} r_i(J)$.*

Proof. Note that $\lambda(J)$ is at most the rank of $\mathbf{M}_i/J$, which is at most $r - r_i(J)$. ■

Proof (of Theorem 4.2). We assume that $\lambda(\emptyset) = r$. By Theorem 4.1, we wish to show that $\psi(J) = r$ iff J is an extensible intersection.

If J is an intersection then $r_1(J) + r_2(J) - |J| = |J|$. If J is also extensible then $\lambda(J) + |J| = \lambda(\emptyset) = r$, by Claim A.3. This establishes the reverse direction of the theorem.

Now let $\psi(J) = r$. Suppose we can show that J is an intersection. Then $|J| + \lambda(J) = \psi(J) = r = \lambda(\emptyset)$, so Claim A.3 shows that J is also extensible as required. It remains to show: $\psi(J) = r$ implies that J is an intersection.

Clearly $\psi(\emptyset) = \lambda(\emptyset) = r$. Furthermore, ψ is non-increasing by Claim A.4, so we can argue by induction on $|J|$. The induction hypothesis is that J is an extensible intersection; this holds trivially for $J = \emptyset$. Consider a set $J+j$ with $\psi(J+j) = r$. Since ψ is non-decreasing, we have $\psi(J+j) = \psi(J)$, or

$$\big(r_1(J+j) - r_1(J)\big) + \big(r_2(J+j) - r_2(J)\big) - 1 \;=\; \lambda(J) - \lambda(J+j). \tag{A.4}$$

We now consider several cases. If $r_i(J+j) - r_i(J) = 1$ for both i then the proof is complete: $J+j$ is an intersection since J is. Furthermore, our previous remarks show that $J+j$ is also extensible.

We now show that the other cases lead to contradictions. If $r_i(J+j) - r_i(J) = 0$ for both i then the left-hand side of Eq. (A.4) is negative, but the right-hand side is non-negative by Claim A.2. The remaining two cases are symmetric. Suppose without loss of generality that $r_1(J+j) - r_1(J) = 1$ and $r_2(J+j) - r_2(J) = 0$, i.e., the left-hand side of Eq. (A.4) has value 0. Since J is an extensible intersection, we have $\lambda(J) + |J| = r$ by Claim A.3 and $r_1(J+j) = |J+j|$. By Claim A.5, $\lambda(J+j) \leq r - |J+j|$. Combining these observations shows that the right-hand side of Eq. (A.4) is strictly positive, which is again a contradiction. ■

A.5 Proof of Theorem 5.3

The following determinant expansion is frequently useful.

Fact 6 (Generalized Laplace expansion)**.** *Let M be an $n \times n$ matrix. Fix a set of rows I such that $\emptyset \neq I \subset \{1, \ldots, n\}$. Then*

$$\det M \;=\; \sum_{J \subset \{1,\ldots,n\},\, |J|=|I|} \det M[I, J] \cdot \det M[\bar{I}, \bar{J}] \cdot (-1)^{\sum_{i \in I} i + \sum_{j \in J} j}. \qquad \text{(A.5)}$$

An analogous statement holds for a set of rows I by taking the transpose of A.

Proof. See Aitken [3] or Murota [42]. ■

For convenience let T' be the south-west submatrix of $Z(M)$, namely

$$T' \;=\; \begin{pmatrix} & I & \\ I & T[\overline{\partial_1 M}, \overline{\partial_2 M}] & T[\overline{\partial_1 M}, C_2 \cup S'] \\ & T[C_1 \cup S', \overline{\partial_2 M}] & T[C_1 \cup S', C_2 \cup S'] \end{pmatrix}.$$

Proof (of Theorem 5.3). We apply the generalized Laplace expansion (Fact 6) to $Z(M)$ with the set of rows $I = [r]$, obtaining

$$\det Z(M) \;=\; \sum_{A \subset [k],\, |A|=r} \pm \det Z(M)[I, A] \cdot \det Z(M)[\overline{I}, \overline{A}] \qquad \text{(A.6)}$$

Clearly A must be a subset of the columns of Q_1 and also $A \supseteq \partial_1 M$, otherwise either $Z(M)[I, A]$ or $Z(M)[\overline{I}, \overline{A}]$ will contain a zero column. So Eq. (A.6) may be rewritten

$$\det Z(M) \;=\; \sum_{\substack{A \subseteq \overline{\partial_1 M} \\ |A| = r - |\partial_1 M|}} \pm \det Q_1[*, A \cup \partial_1 M] \cdot \det Z(M)[\overline{I}, \overline{A \cup \partial_1 M}].$$

We now apply the Laplace expansion to $Z(M)[\overline{I}, \overline{A \cup \partial_1 M}]$ with the set of columns $I' = [r]$. A similar argument yields that $\det Z(M)$ equals

$$\sum_{\substack{A \subseteq \overline{\partial_1 M} \\ |A| = r - |\partial_1 M|}} \pm \det Q_1[*, A \cup \partial_1 M] \;\cdot\; \sum_{\substack{B \subseteq \overline{\partial_2 M} \\ |B| = r - |\partial_2 M|}} \pm \det Q_2[B \cup \partial_2 M, *] \cdot \det T'[\overline{B}, \overline{A}] \qquad \text{(A.7)}$$

Consider now the matrix $T'[\overline{B}, \overline{A}]$. We may use the remaining entries of the identity submatrices to clear their rows and columns, without affecting the determinant. The resulting matrix has the form

$$T'' = \begin{pmatrix} & I & & \\ I & & & \\ & & T[A, B] & T[A, C_2 \cup S'] \\ & & T[C_1 \cup S', B] & T[C_1 \cup S', C_2 \cup S'] \end{pmatrix}$$

where the rows/columns corresponding to set A have been deleted from the bottom-left identity submatrix, and the rows/columns corresponding to set B have been deleted from the other one. To simplify our notation, let $\tilde{T}(A,B)$ denote $T[A \cup C_1 \cup S',\ B \cup C_2 \cup S']$. Thus

$$T'' = \begin{pmatrix} & I & \\ I & & \\ & & \tilde{T}(A,B) \end{pmatrix},$$

implying that

$$\det T'[\overline{B}, \overline{A}] \;=\; \det T'' \;=\; \det \tilde{T}(A,B).$$

The crucial observation is that any monomial in $\det \tilde{T}(A,B)$ cannot appear in $\det \tilde{T}(A',B')$ for any sets with $A' \neq A$ or $B' \neq B$. Therefore there can be no cancelation among the terms of Eq. (A.7), so it follows that $Z(M)$ is non-singular iff there exist $A \subseteq \overline{\partial_1 M}$ and $B \subseteq \overline{\partial_2 M}$ such that $Q_1[*, A \cup \partial_1 M]$, $Q_2[B \cup \partial_2 M, *]$, and $\tilde{T}(A,B)$ are all non-singular. Equivalently, we must have $A \cup C_1 \in \mathcal{B}_{\mathbf{M}'_1}$, $B \cup C_2 \in \mathcal{B}_{\mathbf{M}'_2}$, and $\tilde{T}(A,B)$ non-singular. By Lemma 5.2, this is precisely the condition that $G(M)$ has a bpm. ■

A.6 Proof of Lemma 5.5

The following fact is useful for computing inverses of block matrices.

Fact 7. *Let a non-singular matrix M be given, where*

$$M \;=\; \begin{pmatrix} A & B \\ C & D \end{pmatrix}.$$

Suppose that D is non-singular. Let the Schur complement of D be $S = A - BD^{-1}C$. Then

$$M^{-1} \;=\; \begin{pmatrix} S^{-1} & -S^{-1}BD^{-1} \\ -D^{-1}CS^{-1} & D^{-1} + D^{-1}CS^{-1}BD^{-1} \end{pmatrix}.$$

Proof (of Lemma 5.5). We will apply Fact 7 with $M = Z$ and

$$A = \begin{pmatrix} 0 \end{pmatrix} \qquad B = \begin{pmatrix} Q_1 & 0 \end{pmatrix}$$
$$C = \begin{pmatrix} Q_2 \\ 0 \end{pmatrix} \qquad D = \begin{pmatrix} 0 & I \\ I & T \end{pmatrix}$$

Note that

$$D^{-1} \;=\; \begin{pmatrix} -T & I \\ I & 0 \end{pmatrix}.$$

It follows that

$$\begin{aligned} S &= -\,Q_1 T Q_2 \\ D^{-1}C &= \begin{pmatrix} -TQ_2 \\ Q_2 \end{pmatrix} \\ BD^{-1} &= \begin{pmatrix} -Q_1 T & Q_1 \end{pmatrix}. \end{aligned}$$

By Fact 7, the south-east submatrix of M^{-1} is

$$D^{-1} + D^{-1} C S^{-1} B D^{-1}.$$

However, we are only interested in *its* south-east submatrix, which is

$$Q_2 S^{-1} Q_1,$$

as required. ■