Algebraic Structures and Algorithms for Matching and Matroid Problems

Nicholas J. A. Harvey*
Massachusetts Institute of Technology
nickh@mit.edu

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. This algorithm is based on new algebraic results characterizing the size of a maximum intersection in contracted matroids. Furthermore, the running time of this algorithm is essentially optimal.

1. Introduction

The non-bipartite matching problem — finding the 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 [34]. For example, Edmonds' seminal work on matchings [9, 10] inspired the definition of the class P, and launched the field of polyhedral combinatorics. The matching theory book [24] 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 the largest common independent set in two given matroids — is another fundamental optimization problem, originating in the pioneering work of Edmonds [11, 12]. This work led to significant developments concerning integral polyhedra [33], submodular functions [13] and electrical networks [21]. Algorithmically, matroid intersection is a powerful tool that has been used in various areas such as approximation algorithms [2, 19], mixed matrix theory [28], and network coding [20].

*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.

1.1. Matching algorithms

The literature for non-bipartite matching algorithms is quite lengthy. The initial work of Edmonds [10] gives an algorithm with running time $O(n^2m)$, where n and m respectively are the number of vertices and edges. Several additional improvements culminated in the $O(\sqrt{n}m)$ algorithm of Micali and Vazirani in 1980 [25]. There was little subsequent progress until 2004, when an exciting development of Mucha and Sankowski [27] 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 $n \times n$ matrices [4]. A highly readable exposition of their algorithm is in Mucha's thesis [26].

Unfortunately, all of these algorithm mentioned above are quite complicated. Edmonds' algorithm requires much care in manipulating "blossoms", and the Micali-Vazirani algorithm was not formally proven correct for ten years [38]. 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 [26, §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 [26, §3.4].

In subsequent work, Sankowski [32] developed a parallel (RNC^5) algorithm for constructing perfect matchings that uses only $O(n^\omega)$ processors, yielding another sequential algorithm that uses only $O(n^\omega)$ time. However, this algorithm is also complicated: it depends on sophisticated parallel subroutines for evaluating a polynomial at a matrix and for computing the characteristic polynomial of a matrix.

1.2. Matroid intersection algorithms

Polynomial time algorithms for matroid intersection were developed in the 1970s by various authors [11, 12, 22]. The efficiency of these early algorithms was typically measured relative to an oracle for testing independence. For example, Edmonds' algorithm uses $O(nr^2)$ oracle queries, where r is the rank of the matroid and n is the size of the ground set. Cunningham [5] gave a more efficient algorithm, motivated by intersection of *linear* matroids (those that can be represented as a $r \times n$ matrix). This algorithm makes no oracle queries and uses only $O(nr^2 \log r)$ time. Gabow and Xu [14] obtained an improved bound of $O(nr^{1.62})$ through the use of fast matrix multiplication and quite technical arguments. However, their bound does not seem to be a natural one: for square matrices the 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. Matroid matching problems on general graphs require exponential time in the oracle model, although sophisticated polynomial-time algorithms do exist for linear matroids [24]. On the other hand, *bipartite* matroid matching problems are tractable: they are polynomial-time reducible to matroid intersection [11, Theorem 81] [13].

Another generalization of matroid intersections and non-bipartite matchings are *basic path-matchings*, introduced by Cunningham and Geelen [7, 8, 16]. Their work shows integrality of related polyhedra and shows that one can optimize over these polyhedra using the ellipsoid method [8]. Later work [6] used algebraic techniques together with a matroid intersection algorithm to compute a basic path-matching.

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[1]. 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 [27], and refutes the conjecture [26] 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 [25, 27] also admit such extensions.

Matroid intersection. We present a randomized 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. Restated, 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.

Our algorithm operates with a certain square matrix of size $n+r$, and its inverse. (See Section 4 for details.) Since the inverse has $\Omega(n^2)$ entries, it cannot be explicitly computed — our desired running time is linear in n. Thus a key aspect of our algorithm involves computing and updating the inverse of a matrix, even though it is sparse. These techniques may be useful for other similar problems.

Whereas most existing matroid algorithms use augmenting path techniques, ours uses an algebraic approach. Several previous matroid algorithms also use algebraic techniques [1, 23, 29]. This algebraic approach involves two assumptions. (1) We assume that the given matroids are linear. This is a standard assumption for algorithms that are not oracle-based, since linear matroids are the broadest class of matroids with efficient representations. (2) We make the mild technical assumption 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 [33]. Our results apply to any two matroids from the union of these classes.

Bipartite Matroid Matching. We show a surprising result: the Mucha-Sankowski bipartite graph matching algorithm [27], when run on an appropriate matrix, solves the bipartite matroid matching problem in $O(n^\omega)$ time. Our contribution is to derive the appropriate matrix and to prove some of its properties. Our result improves on the $O(n^2 r^{1.62}) = O(n^{3.62})$ bound obtained via earlier matroid intersection algorithms [14], and the usual reduction from

[1] If $\omega = 2$, the running time is actually $O(n^2 \log n)$. Henceforth, we ignore $\mathrm{polylog}(n)$ factors in expressions of the form $O(n^\omega)$.

bipartite matroid matching to matroid intersection [13].

Basic Path-Matching. We present a novel algebraic structure 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 [16]. We also define a new notion of contraction for basic path-matching problems, allowing us to extend our non-bipartite matching algorithm to a $O(n^\omega)$ algorithm for constructing basic path-matchings.

2. Notation and Basic Facts

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 $M_{\text{del}(i,j)}$ ($M_{\text{del}(I,J)}$). 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.

We will use the following basic facts from linear algebra, proofs of which are in Appendix A.

Fact 1. *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. *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:* $\det M = \det Z \cdot \det\left(W - XZ^{-1}Y\right)$. *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. *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 $\tilde{M}^{-1} = M^{-1} - \alpha^{-1}\left(M^{-1}u\right)\left(v^{\mathsf{T}} M^{-1}\right)$, which is itself a rank-1 update of M^{-1}.*

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

Another important fact about matrices concerns algorithmic efficiency. For matrices of size $n \times n$, the following operations can be performed in $O(n^\omega)$ time: multiplication, determinant computation, rank computation, and inversion (if the matrix is non-singular).

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 where $T_{i,j} = \pm t_{\{i,j\}}$ and the signs are chosen such that T is skew-symmetric. Tutte [37] showed that T is formally non-singular iff G has a perfect matching (see, e.g., Godsil [18]). However, this does not directly imply an efficient algorithm to test if G has a perfect matching: the determinant of T is a formal polynomial which may have exponential size, so computing it symbolically is inefficient. Fortunately, Lovász [23] 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.

A Self-Reducibility Algorithm. This 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-Vazirani [30] showed that, assuming G has a perfect matching, e is allowed iff $N_{i,j} \neq 0$. We observed[2] the following simple proof of their lemma. $G[S \setminus \{i, j\}]$ has a perfect matching iff $\det T_{\text{del}(\{i,j\},\{i,j\})}$ is non-zero (by Tutte's theorem). This determinant is

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

the first equality follows from Fact 1, and the second follows since Fact 5 shows that N is skew-symmetric. 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 $n/2$ matrix inversions and therefore uses $O(n^{\omega+1})$ time in total.

[2] We are grateful to Jim Geelen for pointing out that this argument is precisely equation (2) of Tutte [37].

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

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

As observed above, the matrix $N[\{i, j\}, \{i, j\}] = \hat{Z}$ is skew-symmetric, and therefore

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

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.

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 immediately: our algorithm is simply a variant of the Rabin-Vazirani algorithm that considers edges in an unusual order.

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

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

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 $\log_{\alpha/2}\binom{\alpha}{2} < 2 + \frac{1}{\log \alpha - 1}$, 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 been solved. (Note that the queries and updates to the bit vector do not depend on the input.) 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 update N in the base cases, as in Eq. (3.1), and does not update N after each recursive call. Each rank-1 update requires $O(n^2)$ 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

---

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 α 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

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 π_c, and append j and i to π_r

the parameters of the updates. Consider a single rank-1 update performed by the naive scheme (c.f. 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 $v^{\mathsf{T}} = N_{j,*}$. The algorithm will store the parameters of all updates in three additional $n \times n$ matrices U, V and C. Algorithm 2 illustrates this procedure. The algorithm also maintains two lists π_c and π_r which specify, for each k, which column of U and row of V store the parameters of the k^{th} rank-1 update.

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 $N_{k,i}$ is set to

$$N_{k,i} - \tfrac{1}{N_{j,i}} N_{k,i} N_{j,i} - \tfrac{1}{N_{i,j}} N_{k,j} N_{i,i},$$

which is zero since $N_{i,i} = 0$ by skew-symmetry.

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. Roughly speaking, 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:

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

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; this follows from our invariant.

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 3.1. *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 $1 \le i \le n$. 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; this follows from our earlier observation that $N_{*,i}$ is set to zero when column i participates in an update (i.e., when vertex i is matched). Therefore X and Y satisfy[3] the hypotheses of Lemma 3.1. 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

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

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.

Extensions. The algorithm that we have presented above is a Monte Carlo algorithm for finding a perfect matching. It can be extended to a Las Vegas algorithm for finding a maximum cardinality matching using existing techniques [3, 26, 30]. To find a maximum cardinality matching, simply find a full-rank principal submatrix of the Tutte matrix, then apply our perfect matching algorithm. To make the algorithm Las Vegas, one can efficiently construct an optimum dual solution, i.e., the Gallai-Edmonds decomposition, using Cheriyan's algorithm [3].

4. Matroid Intersection

4.1. Preliminaries

We assume that the reader is familiar with the basic definitions and properties of matroids; introductions can be found in standard references [33]. We write a matroid as a tuple $\mathbf{M} = (S, \mathcal{I}, \mathcal{B}, r)$ where S is the ground set, $\mathcal{I} \subseteq 2^S$ is the collection of independent sets, $\mathcal{B} \subseteq \mathcal{I}$ is the collection of bases, and r is the rank function. Together with S, any one of $\mathcal{I}$, $\mathcal{B}$, and r is sufficient to specify the matroid, so we do not necessarily mention all of them. To emphasize connection to a specific matroid, we sometimes use the notation $\mathcal{I}_{\mathbf{M}}$, $\mathcal{B}_{\mathbf{M}}$ and $r_{\mathbf{M}}$. The rank of the matroid $\mathbf{M}$ is defined to be $r(S)$. For $J \subseteq S$, $\mathbf{M}/J$ denotes the matroid obtained from $\mathbf{M}$ by contracting J. Recall that its rank function is $r_{\mathbf{M}/J}(A) := r_{\mathbf{M}}(J \cup A) - r_{\mathbf{M}}(J)$.

Let $\mathbf{M}_1 = (S, \mathcal{I}_1, r_1)$ and $\mathbf{M}_2 = (S, \mathcal{I}_2, r_2)$ be two matroids with rank r and let $n = |S|$. A set $J \subseteq S$ is called an *intersection* if $J \in \mathcal{I}_1 \cap \mathcal{I}_2$. A maximum intersection is one with maximum size. For convenience, we will assume that $\mathbf{M}_1$ and $\mathbf{M}_2$ have a common base. A subset of a maximum 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 size of a maximum intersection of $\mathbf{M}_1/J$ and $\mathbf{M}_2/J$.

Suppose that each $\mathbf{M}_i$ is a linear matroid representable over a common field $\mathbb{F}$. Let Q_1 be an $r \times t$ matrix whose columns represent $\mathbf{M}_1$ over $\mathbb{F}$ and let Q_2 be a $t \times r$ 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 extensible intersection J, we define the matrix

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

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 [17] and follows easily from the connection between matroid intersection and the Cauchy-Binet formula, as noted by Tomizawa and Iri [36]. Building on our Theorem 4.1, we obtain the following result which is crucial to our algorithm.

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.*

Interestingly, this theorem is false without the assumption that $\lambda(\emptyset) = r$. (There is an example with $|S| = 3$.) Theorem 4.2 can be proven using the following results and induction.

Fact 6. *An independent set J is extensible iff $\lambda(\emptyset) = \lambda(J) + |J|$.*

Fact 7. *For any set $J \subseteq S$, we have $\lambda(J) \leq r - \max_{i \in \{1,2\}} r_i(J)$.*

Fact 8. *The function λ is non-increasing, i.e., $\lambda(I) \geq \lambda(J)$ for all sets $I \subseteq J$.*

Lemma 4.3. *The function $\psi(J) := r_1(J) + r_2(J) - |J| + \lambda(J)$ is non-increasing.*

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 4.4. *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. ∎

The structure of Z will play a key role in our algorithm for matroid intersection below. For simplicity, let $Z = Z(\emptyset)$. Let Y denote the Schur complement of T in Z, i.e., $Y = -Q_1 \cdot T^{-1} \cdot Q_2$. One may verify that

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

4.2. Matroid intersection algorithm

In this section we describe our matroid intersection algorithm, which achieves running time $O(nr^{\omega-1})$. We start by constructing the matrix Z, which is non-singular by our assumption that $\lambda(\emptyset) = r$. As in Algorithm 1, the first step is to randomly substitute values for the indeterminates in Z from the field $\mathbb{F}$, or a sufficiently large extension. By standard arguments, Z remains non-singular whp.

Our algorithm maintains an extensible intersection, initially empty, and searches for allowed elements using Lemma 4.4. This seemingly requires computing the entire matrix Z^{-1}, which would require a prohibitive $\Omega(n^2)$ time. Initially we work under the assumption that Z^{-1} has been completely computed, and later we will show that this assumption can be removed. We also assume that both n and r are powers of two.

Suppose that an allowed element $i \in S$ has been found and we now wish to construct $Z(\{i\})^{-1}$. The matrix $Z(\{i\})$ is identical to Z except that t_i has been set to 0. This can be expressed as the rank-1 update $Z(\{i\}) = Z - t_i e_i e_i^\mathsf{T}$. Here e_i is the i^{th} elementary vector, that is, e_i is 1 in the i^{th} component and zero elsewhere. Using Fact 4, $Z(\{i\})^{-1}$ may be computed by the following equation:

$$Z^{-1} \;-\; (-t_i^{-1} + (Z^{-1})_{i,i})^{-1} \, (Z^{-1})_{*,i} \, (Z^{-1})_{i,*}. \quad (4.3)$$

This is a rank-1 update whose vector parameters are submatrices of Z^{-1}. Therefore it seems that updates can be applied in a similar manner to our non-bipartite matching algorithm. This is indeed possible, although achieving the running time $O(nr^{\omega-1})$ involves some intricacy. To harmonize the notation with Section 3, let N denote the south-east submatrix $(Z^{-1})_{S,S}$. As before, the update parameters will be stored in auxiliary matrices U, C and V.

Our matroid intersection algorithm is also recursive, but the recursion is much simpler than the non-bipartite matching algorithm. The root of the recursion considers the entire ground set S and the entire submatrix N. The ground set is split in two: let S_1 be the first $n/2$ elements and S_2 the remainder. We recurse first on the subproblem S_1, and then on S_2.

Although the recursion of this algorithm is simple, it needs to maintain somewhat complicated invariants in order to obtain the desired level of efficiency. As before, we will say that a submatrix is *clean* if all existing updates have been applied to it. The invariants are:

Invariant 1: When the recursion enters a subproblem with elements A, the submatrix $N[A, A]$ is clean if $|A| \le r$. If $|A| > r$ then only the $r \times r$ blocks on the diagonal of $N[A, A]$ are clean.

Invariant 2: For any subproblem, just before its second child subproblem begins processing, the U_{SW} and V_{NE} submatrices are clean.

A base case of the recursion tree considers a single element i, and checks if $N_{i,i} \neq t_i^{-1}$. (Invariant 1 ensures that this entry is clean.) If so, element i is added to the intersection, and we conceptually perform the update of Eq. (4.3). As in Section 3, we conceptually store the update parameters in U, C and V, but actually only perform $O(1)$ work:

$$U_{i,i} \,:=\, N_{i,i} \quad V_{i,i} \,:=\, N_{i,i} \quad C_{i,i} \,:=\, -1/(-t_i^{-1} + N_{i,i})$$
$$N_{i,i} \,:=\, N_{i,i} - (N_{i,i})^2/(-t_i^{-1} + N_{i,i})$$

The remainder of the entries are updated later.

Consider now a subproblem at level i of the recursion tree. The number of elements in this subproblem is $n2^{-i}$. Say that the entries of its children are S_1 and S_2. Once its first child subproblem has completed, we must restore invariant (2) by updating U_{SW} and V_{NE}. This step is more involved than in Section 3 because of our relaxed invariants. We say that an update is *old* if it was produced *before* entering subproblem S_1. The difficulty is that old updates might not have been applied to N_{SW}. The easy case is when $n2^{-i} \le r$. In this case, invariant (1) implies that the old updates have been applied. It remains to apply the pending updates that were generated within the first child subproblem. This is done using Lemma 3.1, as in Section 3. The time required is only $O((n2^{-i})^\omega)$.

The more difficult case is when $n2^{-i} > r$. In this case, there may be old updates which have not yet been applied to N_{SW}. In fact, no updates have been applied to N_{SW} whatsoever. However, the portions of U and V which contain the updates relevant to N_{SW} are clean; this follows from applying invariant (2) to the parent of the current subproblem. The only portion of N_{SW} that is of interest is the submatrix corresponding to columns that were added to the intersection. Let us denote this submatrix by $\tilde{N}$. Its size is at most $n2^{-i} \times r$, since any intersection has size at most r. The number of old updates which must be applied to $\tilde{N}$ is also at most r. Therefore the time required to apply the old updates is bounded by the time to multiply an $n2^{-i} \times r$ matrix by an $r \times r$ matrix, which is $O(n2^{-i} r^{\omega-1})$ time.

Next, we must make N_{SW} clean. That is, we must resolve the interdependencies of updates generated within the first child subproblem. Using Lemma 3.1, this requires $O(n2^{-i} r^{\omega-1})$ time. A similar argument applies to updating N_{NE} and V_{NE}. Finally, we must update N_{SE}. To restore the invariants, we only need update its $r \times r$ diagonal blocks. This requires time only $O(n2^{-i} r^{\omega-1})$ by the obvious approach.

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 $\log r$ factor. For levels $i \ge \log(n/r)$, the time is

$$\sum_{i=\log(n/r)}^{\log n} 2^i \cdot O((n2^{-i})^\omega) = \sum_{i=0}^{\log r} O(n2^{(\omega-1)i}) = O(nr^{\omega-1}),$$

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

Figure 1: The matrix $Z(M)$ used for the basic path-matching problem.

$$
Z(M) = \begin{pmatrix}
 & Q_1^{\partial_1 M} & Q_1^{\overline{\partial_1 M}} & & \\
Q_2^{\partial_2 M} & & & & \\
Q_2^{\overline{\partial_2 M}} & & & D_2 & \\
 & & D_1 & T[\overline{\partial_1 M}, \overline{\partial_2 M}] & T[\overline{\partial_1 M}, C_2 \cup S'] \\
 & & & T[C_1 \cup S', \overline{\partial_2 M}] & T[C_1 \cup S', C_2 \cup S']
\end{pmatrix} \tag{4.4}
$$

The preceding discussion assumes that N was initially fully computed. 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. (4.2), this amounts to computing the $r \times r$ diagonal blocks of $Q_2^{\mathsf{T}} Y^{-1} Q_1$ plus some negligible additional work, and hence $O(nr^{\omega-1})$ time suffices. Next, consider performing an update where $n2^{-i} > r$. Then we must compute the initial submatrix $\tilde{N}$ (before any updates are applied). Recall that $\tilde{N}$ has size at most $n2^{-i} \times r$. Then, referring again to Eq. (4.2), we see that $\tilde{N}$ can be computed in $O(n2^{-i} r^{\omega-1})$ time. Thus the preceding analysis applies without change.

Extensions. The algorithm presented above is a Monte Carlo algorithm for finding a common base. To find a maximum cardinality intersection instead, start by finding a full-rank submatrix of the matrix Z, then apply our common base algorithm. It is possible to find such a submatrix in time $O(nr^{\omega-1})$, although we omit the details. The algorithm can be made Las Vegas by constructing an optimum dual solution. Simply construct the auxiliary graph used by Lawler's algorithm [33], 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 $\partial_S M = S$.

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. Informally, 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 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 $G(M) := (R_1', R_2', S', E', \mathbf{M}_1', \mathbf{M}_2')$.

Fact 9. *If $M' \supseteq M$ is a bpm for G then $M' \setminus M$ is a bpm for $G(M)$. Conversely, if M' is a bpm of $G(M)$ then $M \cup M'$ is a bpm of G.*

We now define an algebraic structure which extending 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 5.1 (Geelen [16]). *T is non-singular iff G has a perfect path-matching.*

Let D_i be a diagonal matrix of size $|\overline{\partial_i M}|$ whose non-zero entries contain distinct indeterminates. Define $Z(M)$ to be the matrix in Eq. (4.4).

Theorem 5.2. *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. Equivalently, M is an extensible set iff $Z(M)$ is non-singular.*

An edge $e = \{i, j\}$ in $G(M)$ is called *allowed* (relative to M) if $M + e$ is also extensible. An extension of Lemma 4.4 characterizes whether edges are allowed. Using this, the algorithm of Section 3 can be extended to compute a basic path-matching in time $O(n^\omega)$. Details are postponed to the full version of the paper.

5.2. Bipartite Matroid Matchings

An instance of the bipartite matroid matching problem is a tuple $G = (R_1, R_2, E, \mathbf{M}_1, \mathbf{M}_2)$ where $E \subseteq R_1 \times R_2$ and $\mathbf{M}_i = (R_i, \mathcal{I}_i)$ is a matroid. An *independent matching* in G is a matching $M \subseteq E$ such that $\partial_1 M \in \mathcal{I}_1$ and $\partial_2 M \in \mathcal{I}_2$. The objective is to find a maximum cardinality independent matching. Bipartite matroid matching is a special case of basic path-matching where $S = \emptyset$. Also, matroid intersection is a special case of bipartite matroid matching where all vertices have degree 1. Theorem 5.2 extends to show that the maximum cardinality of an independent matching in $G(M)$ is $\operatorname{rank} Z(M) - 2n$, for any $M \subseteq E$.

Additionally, the algorithm for basic path-matchings immediately solves the bipartite matroid matching problem in $O(n^\omega)$ time. However, for bipartite matroid matching problems, the matrix $Z(M)$ contains no skew-symmetric parts. Therefore the algorithm can be simplified, and one obtains an algorithm that amounts to executing the Mucha-Sankowski bipartite matching algorithm on the lower-right submatrix of Z^{-1}. 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 $\tilde{Q} := Q_2(Q_1 T Q_2)^{-1} Q_1$ (which is the lower-right submatrix of Z^{-1}) it computes a maximum independent matching.

6. Open Questions

Graphical Matroid Intersection. One can obtain a polynomial-time algorithm for matroid intersection that works only with the matrix Y, as defined in Section 4. How efficient can such an algorithm be? Consider the specific case of graphical matroids — the standard representation is a matrix with two non-zero entries per column. In this framework, contracting an element requires only $O(1)$ changes to Y. Using Sankowski's dynamic matrix inverse operations [31], one obtains a $O(n^{2.575})$ algorithm for graphical matroid intersection, which is slightly worse than the $O(n^{2.5} \log n)$ algorithm of Gabow-Xu [14]. Does a $O(n^\omega)$ algorithm exist?

Skew-Symmetric Matrices. The recursion presented in Section 3 is a general technique for performing eliminations in skew-symmetric matrices in $O(n^\omega)$ time. Can it be used to obtain efficient algorithms for other operations on skew-symmetric matrices?

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.

References

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

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

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

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

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

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

[7] 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.

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

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

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

[11] 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.

[12] 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.

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

[14] 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.

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

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

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

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

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

[20] 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.

[21] M. Iri. Applications of matroid theory. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 158–201. 1983.

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

[23] 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.

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

[25] 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.

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

[27] 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.

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

[29] 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.

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

[31] P. Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 509–517, 2004.

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

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

[34] 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.

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

[36] 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.

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

[38] 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.

A. Additional Proofs

Proof (of Fact 1). This is Jacobi's theorem. The following proof is folklore; see also Gantmakher [15, §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 [28]. Note that:

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

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):

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

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 $A = \begin{pmatrix} -c^{-1} & v^\mathsf{T} \\ u & M \end{pmatrix}$. 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

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

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

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

Proof (of Lemma 3.1). First, note that $X^{(j)}_{*,i} = X^{(k)}_{*,i}$ if $i \leq j \leq k$ since Y is strictly upper triangular. Define $A = \left(\begin{smallmatrix} I-Y & 0 \\ X & I \end{smallmatrix}\right)$, and consider performing Gaussian elimination on A. Let $S^{(i)}$ denote the south-west submatrix of A just before the i^{th} 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 i^{th} 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 i^{th} elimination (see, e.g., Strang [35]). Thus $B = X^{(n)} = X \otimes Y$. The lemma follows by observing that $X = B \cdot (I - Y)$. ■