2013 IEEE 54th Annual Symposium on Foundations of Computer Science

Algebraic Algorithms for b-matching, Shortest Undirected Paths, and f-factors

Harold N. Gabow*, Piotr Sankowski†‡
*Department of Computer Science, University of Colorado at Boulder, USA. Email: hal@cs.colorado.edu
†Institute of Informatics, University of Warsaw , Poland. Email: sank@mimuw.edu.pl
‡Department of Computer and System Science, Sapienza University of Rome, Italy.

***Abstract*—Let $G = (V, E)$ be a graph with $f : V \to \mathbb{Z}_+$ a function assigning degree bounds to vertices. We present the first efficient algebraic algorithm to find an f-factor. The time is $O(f(V)^\omega)$. More generally for graphs with integral edge weights of maximum absolute value W we find a maximum weight f-factor in time $\widetilde{O}(Wf(V)^\omega)$. (The algorithms are correct with high probability and can be made Las Vegas.) We also present three specializations of these algorithms: For maximum weight perfect f-matching the algorithm is considerably simpler (and almost identical to its special case of ordinary weighted matching). For the single-source shortest-path problem in undirected graphs with conservative edge weights, we define a generalization of the shortest-path tree, and we compute it in $\widetilde{O}(Wn^\omega)$ time. For bipartite graphs, we improve the known complexity bounds for vertex-capacitated max-flow and min-cost max-flow on a subclass of graphs.**

***Keywords*- b-matching; shortest undirected paths; f-factors; min-cost max-flow; matrix multiplication**

I. Introduction

b-matching and f-factors are basic combinatorial notions that generalize non-bipartite matching, min-cost network flow, and others. This paper presents the first efficient algebraic algorithms for both weighted and unweighted b-matchings and f-factors. Our algorithms for this broad class of problems are the most efficient algorithms known for a subclass of instances (graphs of high density, low degree constraints and low edge weights). We also discuss single-source all-sinks shortest paths in conservative undirected graphs. (There is no known reduction to directed graphs.) We prove the existence of a simple shortest-path "tree" for this setting. We also give efficient algorithms – combinatoric for sparse graphs and algebraic for dense – to construct it.

We must first define b-matching and f-factors. The literature is inconsistent but in essence we follow the classification of Schrijver [1]. For an undirected multigraph $G = (V, E)$ with a function $f : V \to \mathbb{Z}_+$, an *f-factor* is a subset of edges wherein each vertex $v \in V$ has degree exactly $f(v)$. For an undirected graph $G = (V, E)$ with a function $b : V \to \mathbb{Z}_+$, a (perfect) *b-matching* is a function $x : E \to \mathbb{Z}_+$ such that each $v \in V$ has $\sum_{w:vw\in E} x(vw) = b(v)$. The fact that b-matchings have an unlimited number of copies of each edge makes them decidedly simpler. For instance b-matchings have essentially the same blossom structure (and linear programming dual variables) as ordinary matching [1, Ch.31]. Similarly our algorithm for weighted b-matching is almost identical to its specialization to ordinary matching ($b \equiv 1$). In contrast the blossoms and dual variables for f-factors are more involved [1, Ch.32] and our algorithm is more intricate. Thus our terminology reflects the difference in complexity of the two notions.[1]

Time	Author
$O(n^2B)$	Pulleyblank (1973) [7]
$O(n^2m \log B)$	Marsh (1979) [8]
$O(m^2 \log n \log B)$	Gabow (1983) [6]
$O(n^2m + n \log B(m + n \log n))$	Anstee (1987) [9]
$O(n^2 \log n(m + n \log n))$	Anstee (1987) [9]
$\widetilde{O}(W\phi^\omega)$[4]	this paper

Table 1: Time bounds for maximum b-matching. B denotes $\max_v b(v)$.

Complexity	Author
$O(\phi n^3)$	Urquhart (1965) [10]
$O(\phi(m + n \log n))$	Gabow (1983+1990) [6], [11]
$\widetilde{O}(W\phi^\omega)$	this paper

Table 2: Time bounds for maximum weight f-factors on simple graphs.

We begin with unweighted f-factors, i.e., we wish to find an f-factor or show none exists. Let $\phi = f(V)$ (or $b(V)$). We extend the Tutte matrix from matching to f-factors, i.e., we present a $\phi \times \phi$ matrix that is symbolically nonsingular iff the graph has an f-factor. Such a matrix can be derived by applying the Tutte matrix to an enlarged version of the given graph, or by specializing Lovász's matrix for matroid parity [2]. But neither approach is compact enough to achieve our time bounds.[2] Then we reuse the elimination framework for maximum cardinality matching, due to Mucha and Sankowski [3] and Harvey [4]. This allows us to find an f-factor in $O(\phi^\omega)$ randomized time.[3] For dense graphs and small degree-constraints we improve the best-known time bound of $O(\sqrt{\phi}m)$ [6], although the latter is deterministic.

We turn to the more difficult weighted version of the problem. Here every edge has a numeric weight; we assume

[1]Another version of b-matching considers $b(v)$ as an upper bound on the desired degree of v. This reduces to weighted b-matching by taking 2 copies of G joined by zero-weight edges. On the other hand in a *capacitated b-matching* we are given an upper bound $u(e)$ to each value $x(e)$. The simplicity of the uncapacitated case is lost, and we are back to f-factors.

[2]The Tutte matrix becomes too large, $m \times m$. Lovász's matrix is $\phi \times \phi$ but involves integers that are too large, n^n. Our matrix only uses ± 1.

[3]$O(n^\omega)$ is the time needed for a straight-line program to multiply two $n \times n$ matrices; the best-known bound on ω is < 2.3727 [5].

[4]The $\widetilde{O}$ notation ignores factors of $\log(n\phi W)$.

0272-5428/13 $26.00 © 2013 IEEE
DOI 10.1109/FOCS.2013.23

weights are integers of magnitude $\leq W$. We seek a *maximum f-factor*, i.e., an f-factor with the greatest possible total weight. Efficient algebraic algorithms have been given for maximum matching ($f \equiv 1$) in time $\widetilde{O}(Wn^\omega)$, first for bipartite graphs [12] and recently for general graphs [13].

The usual approach to generalized matching problems is by problem reduction. For instance in [1], Ch.31 proves the properties of the b-matching linear program and polytope by reducing to ordinary matching via vertex splitting; then Ch.32 reduces f-factors (capacitated b-matching) to b-matching. Efficient algorithms also use vertex splitting [6] or reduction to the bipartite case (plus by further processing) [9]. But reductions may obscure some structure. To avoid this we use a direct approach and get the following rewards. For b-matching, as mentioned, the similarity of blossoms to ordinary matching blossoms leads to an algorithm that is no more involved than ordinary matching. For undirected shortest paths we get a simple definition of a generalized shortest-path tree. (Again such a definition may have been overlooked due to reliance on reductions, see below.) For f-factors we get a detailed understanding of the more complicated versions of the structures that first emerge in b-matchings (2-edge-connected components giving the cyclic part of blossoms) and in shortest paths (bridges giving the incident edges of blossoms – these correspond to the ungeneralized shortest-path tree).

All three of our non-bipartite algorithms are implementations of the "shrinking procedure" given in [14] (a variant is the basis of the weighted matching algorithm of [13]). This procedure gives a direct way to find the optimum blossoms for a weighted f-factor – each blossom is (a subgraph of) a maximum weight "$2f$-unifactor" (a type of $2f$-factor) in the graph with (the cyclic part of) all heavier blossoms contracted (see [14]). Note that the classic weighted matching algorithm of Edmonds [15] finds the optimum blossoms, but only after forming and discarding various other blossoms. So it does not provide a direct definition of the optimum blossoms.

The first step of our algorithms use our generalized Tutte matrix to find the optimum duals of the vertices. Then we execute the shrinking procedure to get the blossoms, their duals, and a "weighted blossom tree" that gives the structure of the optimum f-factor. (This step is combinatoric. It is based on the detailed structure of $2f$-unifactors that we derive.) The last step finds the desired f-factor using a top-down traversal of the weighted blossom tree: At each node we find an f-factor of a corresponding graph, using our algorithm for unweighted f-factors. Our algorithms (like [13] for ordinary matching) can be viewed as a (combinatoric) reduction of the weighted f-factor problem into two subproblems: finding the optimum dual variables of the vertices, and finding an unweighted f-factor.

To facilitate understanding of the general f-factor algorithm we begin by presenting its specialization to two subcases. First b-matching. The blossoms, and hence the $2b$-unifactors, differ little from ordinary graph matching. As a result our development for weighted b-matching is essentially identical to the special case of ordinary matching, in terms of both the underlying combinatorics and the algorithmic details. When specialized to ordinary matching our algorithm provides a simple alternative to [13]. In fact an advantage is that our algorithm is Las Vegas – the dual variables allow us to check if the b-matching is truly optimum. (Our approach to weighted matching/b-matching differs from [13] – at the highest level, we work with critical graphs while [13] works with perfect graphs.)

Next we discuss shortest paths in undirected graphs with a conservative weight function – negative edges are allowed but not negative cycles. The obvious reduction to a directed graph (replace undirected edge uv by directed edges uv, vu) introduces negative cycles and so fails.

We consider the single-source all-sinks version of the problem. Again, this problem is often solved by reduction, first to the single-source single-sink version and then to perfect matching, using either T-joins [1, pp.485–486] or vertex-splitting [1, p.487]. A path can be viewed as a type of 2-factor. (For instance an ab-path is an f-factor if we enlarge G with a loop at every vertex $v \in V$ and set $f(v) = 2$ for $v \in V - \{a, b\}$, and $f(a) = f(b) = 1$.) This enables us to solve the all-sinks version directly. Examining the blossom structure enables us to define a generalized shortest-path tree that, like the standard shortest-path tree for directed graphs, specifies a shortest path to every vertex from a chosen source. It is a combination of the standard shortest-path tree and the blossom tree, plus a generalization of Bellman's inequalities. We give a complete derivation of the existence of this shortest-path structure, as well as an algebraic algorithm to construct it in time $\widetilde{O}(Wn^\omega)$. We also construct the structure with combinatoric algorithms, in time $O(n(m + n\log n))$ or $O(\sqrt{n\alpha(m,n)\log n}\ m\log(nW))$. These bounds are all within logarithmic factors of the best-known bounds for constructing the directed shortest-path tree [1, Ch. 8], [16], [17].

Although the shortest-path problem is classic, our definition of this structure appears to be new. Most notably, Sebö has characterized the structure of single-source shortest paths in undirected graphs, first for graphs with ± 1 edge weights [18] and then extending to general weights by reduction [19]. Equation (4.2) of [18] (for ± 1-weights, plus its version achieved by reduction for arbitrary weights) characterizes the shortest paths from a fixed source in terms of how they enter and leave "level sets" determined by the distance function. [18] also shows that the distances from the source can be computed using $O(n)$ perfect matching computations. Our structure differs from [18], [19]: it does not give a necessary and sufficient condition to be a shortest path, but it gives an exact specification of a set of shortest paths that are simply related to one another (as in the standard shortest-

path tree). One can give an alternative proof of the existence of our structure by starting from [18], [19].

The general algorithm for maximum f-factors is the most difficult part of our results. It involves a detailed study of the properties of blossoms. A simple example of how these blossoms differ from ordinary matching is that the hallmark of Edmonds' blossom algorithm – "blossoms shrink" – is not quite true. In other words for ordinary matching a blossom can be contracted and it becomes just an ordinary vertex. For f-factors we can contract the "cycle" part of the blossom, but its incident edges remain in the graph and must be treated differently from ordinary edges. Our discussion of shortest paths introduces this difficulty in the simplest case – here a blossom has exactly 1 incident edge (as opposed to an arbitrary number). Even ignoring this issue, another difficulty is that there are three types of edges that behave differently (see [14] or Section VII) and the type of an edge is unknown to the algorithm! Again the three types are seen to arise naturally in shortest paths. Our contribution is to develop the combinatoric properties of these edges and blossoms so the shrinking procedure can be executed efficiently, given only the information provided by the Tutte matrix.

While non-bipartite graphs present the greatest technical challenge, we also achieve some best-known time bounds for two bipartite problems, maximum network flow and min-cost network flow. Bipartite f-factors generalize network flow: max-flow (min-cost max-flow) is a special case of unweighted (weighted) bipartite f-factors, respectively, e.g., [20]. The question of an efficient algebraic max-flow algorithm has confronted the community for some time. The only advance is the algorithm of Cheung et. al. [21], which checks whether d units of flow can be sent across a unit-capacity network in $O(d^{\omega-1}m)$ time. We consider networks with integral vertex and edge capacities bounded by D. We find a max-flow in time $\widetilde{O}((Dn)^\omega)$ time and a min-cost max-flow in $\widetilde{O}(W(Dn)^\omega)$ time. The latter algorithm handles convex edge-cost functions (with integral breakpoints) as well. The max-flow problem has a rich history (see e.g. [1, Chs. 10, 12]) and our time bounds are the best-known for dense graphs with moderately high vertex capacities. Specifically, previous algorithms for vertex-capacitated max-flow in dense networks (i.e., $m = \Theta(n^2)$) use $O(n^3/\log n)$ time [22] or $O(n^{8/3}\log D)$ time [23]. Previous algorithms for dense graph min-cost max-flow use $O(n^3 \log D)$ time [24] or $O(n^3 \log n)$ time [25], whereas for minimum convex-cost max-flow one needs $O(Dn^3 \log D)$ time (by simple reduction to min-cost max-flow) or $O(n^3 \log D \log(nW))$ time [20].

In summary the novel aspects of this paper are: *(i)* new time bounds for the fundamental problems of b-matching, undirected single-source shortest paths, and f-factors; *(ii)* extension of the Tutte matrix to f-factors; *(iii)* definition of the shortest-path structure for undirected graphs, plus algebraic and combinatorial algorithms to construct it; *(iv)* an algebraic algorithm for b-matching that is no more involved than ordinary matching; *(v)* an algebraic algorithm for f-factors based on new combinatorial properties of blossoms; *(vi)* new time bounds for vertex-capacitated max-flow, min-cost max-flow and convex-cost max-flow on dense graphs.

Organization of the paper: This extended abstract introduces the basic ideas of our results. It presents the bipartite case in full detail, and indicates how this extends to the non-bipartite case. It briefly describes our results for shortest paths, which also generalize to the non-bipartite case. The non-bipartite case is the highlight of this paper and is technically much more demanding. However due to space limitations the detailed development of our non-bipartite and shortest-path algorithms is in the full version of this paper [26].

In detail, we start by giving basic definitions and our algebraic tools. Next, we present an $O(\phi^\omega)$ time algorithm for unweighted bipartite f-factors. This requires our generalized Tutte matrix (Section III) plus algorithmic details (Section IV). Section V gives the algorithm for weighted bipartite f-factors. Flows are discussed in Section VI. Section VII overviews the generalized shortest-path tree.

II. Preliminaries

Problem definitions: The symmetric difference of sets is denoted by $\oplus$, i.e., $A \oplus B = (A - B) \cup (B - A)$. We use a common convention to sum function values: If f is a real-valued function on elements and S is a set of such elements, $f(S)$ denotes $\sum\{f(v) : v \in S\}$.

Let $G = (V, E)$ be an undirected graph, with vertex set $V = \{1, \ldots, n\}$. We sometimes write $V(G)$ or $E(G)$ to denote vertices or edges of graph G. A *walk* is a sequence $A = v_0, e_1, v_1, \ldots, e_k, v_k$ for vertices v_i and edges $e_i = v_{i-1}v_i$. The notation v_0v_k*-walk* specifies the two endpoints. The *length* of A is k, and the parity of k makes A *even* or *odd*. A *trail* is an edge-simple walk. A *circuit* is a trail that starts and ends at the same vertex. The vertex-simple analogs are *path* and *cycle*.

In an undirected multigraph $G = (V, E)$ each edge $e \in E$ has a positive multiplicity $\mu(e)$. Each copy of a fixed $e \in E$ is a distinct edge, e.g., a trail may use up to $\mu(e)$ distinct copies of e.

For a set of vertices $S \subseteq V$ and a subgraph H of G, $\delta(S, H)$ ($\gamma(S, H)$) denotes the set of edges with exactly one (respectively two) endpoints in S (loops are in γ but not δ). $d(v, H)$ denotes the degree of vertex v in H. When referring to the given graph G we often omit the last argument.

An edge weight function w assigns a numeric weight to each edge. For complexity bounds we assume the range of w is $[-W..W]$, i.e., the set of integers of magnitude $\leq W$. The *weight of edge set* $F \subseteq E$ is $w(F)$. w is *conservative* if there are no negative weight cycles. For multigraphs we denote by $w(e, k)$ the weight of the kth copy of edge e.

Let $G = (V, E)$ be a multigraph. For a function $f : V \to \mathbb{Z}_+$, an *f-factor* is a subset of edges $F \subseteq E$ such that $d(v, F) = f(v)$ for every $v \in V$. Let $G = (V, E)$ be a graph, where E may contain loops vv but no parallel edges. For a function $b : V \to \mathbb{Z}_+$, a (perfect) *b-matching* is a function $x : E \to \mathbb{Z}_+$ such that $\sum_{w:vw\in E} x(vw) = b(v)$. A *maximum f-factor* is an f-factor F with maximum weight $w(F)$. Similarly a *maximum b-matching* is a perfect b-matching of maximum weight, i.e., $\sum_{e\in E} x(e)w(e)$.

To simplify the time bounds we assume matrix multiplication time is $\Omega(n^2 \log n)$. This allows us to include terms like $O(m)$ and $O(m \log n)$ within our overall bound $O(\phi^\omega)$. Observe that $m = O(n^2) = O(\phi^2)$ if there are no parallel edges, or if all copies of an edge have the same weight. In the most general case – arbitrary parallel edges – we can assume $m = O(n\phi) = O(\phi^2)$ after linear-time preprocessing. In proof, for each edge uv the preprocessing discards all but the $f(u)$ largest copies. This leaves $\leq \sum\{d(u)f(u) : u \in V\} \leq \sum\{nf(u) : u \in V\} \leq n\phi$ edges in G.

Algebraic tools: We encode graph problems in matrices, in such a way that determinant of a matrix is (symbolically) non-zero if and only if the problem has a solution. The Schwartz-Zippel Lemma [27], [28] provides us with an efficient non-zero test for such symbolic determinants. For our purposes the following simplified version suffices.

Lemma 2.1 (Schwartz-Zippel): For any prime p, if a nonzero multivariate polynomial of degree d over $\mathcal{Z}_p$ is evaluated at a random point, the probability of false zero is $\leq \frac{d}{p}$.

After finding the right encoding we compute the determinant using the following result in symbolic computation by Storjohann [29].

Theorem 2.2 (Storjohann '03): Let K be an arbitrary field, $A \in K[y]^{n\times n}$ a polynomial matrix of degree W, and $b \in K[y]^{n\times 1}$ a polynomial vector of the same degree. Then the rational system solution $A^{-1}b$, and the determinant $\det(A)$ can be computed in $\widetilde{O}(Wn^\omega)$ operations in K, w.h.p. [29, Algorithms 5 and 12].

III. Determinant formulations

Consider a simple bipartite graph G, with both vertex sets V_0, V_1 numbered from 1 to n. Let $\phi = \sum_i f(i)/2$. Define a $\phi \times \phi$ matrix $B(G)$, the *symbolic adjacency matrix of G, f*, as follows. A vertex $i \in V_0$ is associated with $f(i)$ rows indexed by pairs i, r, for $0 \leq r < f(i)$. Similarly $j \in V_1$ is associated with $f(j)$ columns indexed by j, c, for $0 \leq c < f(j)$. Define $B(G)$ using indeterminates x_r^{ij} and y_c^{ij} as

$$B(G)_{i,r,j,c} = \begin{cases} x_r^{ij} y_c^{ij} & ij \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Observe that each edge in the graph is represented by a rank-one submatrix given by the product of two vectors $x^{ij}(y^{ij})^T$. Before we prove the main theorems we make the following observation that each edge can be used only once.

Lemma 3.1: Let A be a symbolic $n \times n$ matrix, with R (C) a set of l rows (columns) of A. If $A[R, C]$ (the submatrix of A restricted to rows in R and columns in C) has rank bounded by r then each term in the expansion of $\det(A)$ contains at most r elements from $A[R, C]$.

Proof: Using Laplace expansion we expand $\det(A)$ into $l \times l$ minors that contain all l rows of R, i.e.,

$$\sum_{M\subseteq V_1, |M|=l} \operatorname{sgn}(M) \det(A[R, M]) \det(A[V_0 - R, V_1 - M]),$$

where $\operatorname{sgn}(M) = \prod_{c\in M}(-1)^c$. Consider each element of the above sum separately. If M contains $> r$ columns of C then $\det(A[R, M]) = 0$, so the elements contributing to $\det(A)$ have $\leq$ columns of C. Moreover, $A[V_0 - R, V_1 - M]$ has no rows of $A[R, C]$, so $\det(A[R, M]) \det(A[V_0 - R, V_1 - M])$ has $\leq r$ entries from $A[R, C]$. ∎

Note that $\det(B(G))$ is the sum of many different terms each containing exactly ϕ occurrences of variable pairs $x_r^{ij} y_c^{ij}$. Each $x_r^{ij} y_c^{ij}$ corresponds to an edge $ij \in E$. For a term σ let F_σ be the multiset of edges that correspond to the variable pairs in σ. Define $\mathcal{F}$ to be the function that maps each term σ to F_σ.

Theorem 3.2: Let G be a simple bipartite graph. The function $\mathcal{F}$ from terms in $\det(B(G))$ is a surjection onto the f-factors of G. Hence G has an f-factor iff $\det(B(G)) \neq 0$.

Proof: First we show that the image of $\mathcal{F}$ contains all f-factors of G. Suppose F is an f-factor in G. Order the edges of F that are incident to each vertex arbitrarily. If $ij \in F$ is the $r+1$st edge at i and the $c+1$st edge at j then it corresponds to entry $B(G)_{i,r,j,c}$. Thus F corresponds to a nonzero term σ in the expansion of $\det(B(G))$. Observe that entries $B(G)_{i,r,j,c}$ define σ in a unique way, so no other term has exactly the same indeterminates. Of course there can be many terms representing F.

Can $\det(B(G))$ contain terms that do not correspond to f-factors? The answer is no. Suppose $\det(B(G)) \neq 0$ and take any term σ in the expansion of $\det(B(G))$. σ corresponds to an f-factor unless more than one entry corresponds to the same edge of G. This is impossible because edges are represented by rank-one submatrices and Lemma 3.1 shows elements of such submatrices appear at most once. ∎

Now let G be a bipartite multigraph. Let $\mu(e)$ denote the multiplicity of any edge $e \in E$. Redefine $B(G)$ by

$$B(G)_{i,r,j,c} = \sum_{k=1}^{\mu(ij)} x_r^{ij,k} y_c^{ij,k}. \tag{2}$$

In other words, now edge ij of multiplicity $\mu(ij)$ is represented by a submatrix of rank $\mu(ij)$. Hence Lemma 3.1 shows edge ij can appear in a term of $\det(B(G))$ at most $\mu(ij)$ times. This leads to a generalization of Theorem 3.2:

Corollary 3.3: Let G be a bipartite multigraph. The function $\mathcal{F}$ from terms in $\det(B(G))$ is a surjection onto the f-factors in G. Hence, G has an f-factor iff $\det(B(G)) \neq 0$.

Finally let us discuss the complexity of using the $B(G)$ matrix. As in most algebraic algorithms we evaluate $B(G)$ using a random value for each indeterminate to get a matrix B. If G is a simple graph this can done in time $O(\phi^2)$ using (1). For multigraphs we need to use a different approach and construct B in time $O(\phi^\omega)$, as follows.

As observed above $B[I,J]$ is the product of an $f(i) \times \mu(ij)$ matrix X of indeterminates $x_r^{ij,k}$ and an $\mu(ij) \times f(j)$ matrix Y of indeterminates $y_c^{ij,k}$. Wlog assume $f(i) \leq f(j)$. Moreover, if $\mu(ij) < f(i)$ we pad X and Y with zeros and set $\mu(ij) = f(i)$. Break up the product XY into products of $f(i) \times f(i)$ matrices. Using fast matrix multiplication, computing XY requires $O(f(i)^\omega \frac{f(j)}{f(i)})$ time. The total time to compute B is bounded by a constant times

$$\sum_{i,j} f(i)^{\omega-1} f(j) \leq \sum_i f(i)^{\omega-1} (\sum_j f(j)) \leq \phi^{\omega-1}\phi = \phi^\omega.$$

IV. Finding f-factors

This section gives our algorithm to find an f-factor of a bipartite multigraph. We follow the development from [3]: We start with an $O(\phi^3)$-time algorithm. Then we show it can be implemented in $O(\phi^\omega)$ time using the Gaussian elimination algorithm of Bunch and Hopcroft [30].

An *allowed edge* is an edge belonging to some f-factor. For perfect matchings the notion of allowed edge is expressible using the inverse of $B(G)$: ij is allowed if and only if $B(G)^{-1}_{i,j}$ is non-zero. We prove a similar statement for bipartite f-factors. For a given f define $f_{i,j}$ to be

$$f_{i,j}(v) = \begin{cases} f(v) - 1 & \text{if } v = i \text{ or } v = j, \\ f(v) & \text{otherwise.} \end{cases}$$

Lemma 4.1: Let G be a bipartite multigraph having an f-factor. Edge $ij \in E$ is allowed iff G has an $f_{i,j}$-factor.

Proof: The "only if" direction is clear: If F is an f-factor containing ij, then $F - ij$ is an $f_{i,j}$-factor.

Conversely, suppose F does not contain the chosen edge ij. Take an $f_{i,j}$-factor F' that maximizes $|F' \cap F|$. $F' \oplus F$ contains an alternating ij-trail P that starts and ends with edges of F. In fact P is a path. (Any cycle C in P has even length and so is alternating. This makes $F' \oplus C$ an f_{ij}-factor containing more edges of F than F', impossible.) ij is not the first edge of P ($ij \notin F$). So $ij \notin P$, since P is vertex simple. Thus $(F \oplus P) + ij$ is an f-factor containing ij. ■

Lemma 4.2: Let G be a bipartite multigraph having an f-factor, and let $i \in V_0$ and $j \in V_1$. Then $(B(G)^{-1})_{j,0,i,0} \neq 0$ iff G has $f_{i,j}$-factor.

Proof: Observe that

$$(B(G)^{-1})_{j,0,i,0} = \frac{(-1)^{n(j,0)+n(i,0)} \det(B(G)^{i,0,j,0})}{\det(B(G))},$$

where $B(G)^{i,0,j,0}$ is the matrix $B(G)$ with $i,0$th row and $j,0$th column removed, and $n(i,k)$ is the actual index of the row or column given by the pair i,k. We have $\det(B(G)) \neq 0$ since G has an f-factor. Hence $(B(G)^{-1})_{j,0,i,0} \neq 0$ iff $\det(B(G)^{i,0,j,0}) \neq 0$. Furthermore $B(G)^{i,0,j,0}$ is the symbolic adjacency matrix for G when $f = f_{i,j}$. ■

Observe that by the symmetry of the matrix $B(G)$, when $(B(G)^{-1})_{j,0,i,0} \neq 0$ then as well $(B(G)^{-1})_{j,\kappa,i,\iota} \neq 0$ for all $0 \leq \iota < f(i)$ and $0 \leq \kappa < f(j)$. Combining the above two lemmas with this observation we obtain the following.

Corollary 4.3: Let G be a bipartite multigraph having an f-factor, then the edge $ij \in E$ is allowed if and only if $(B(G)^{-1})_{j,\kappa,i,\iota} \neq 0$ for $0 \leq \iota < f(i)$ and $0 \leq \kappa < f(j)$.

Being equipped with a tool for finding allowed edges we can now use the Gaussian elimination framework from [3]. Here, the following observation is essential.

Lemma 4.4 ([3]): Let A be a non-singular $\phi \times \phi$ matrix and let $1 \leq i,j \leq n$ be such that $(A^{-1})_{j,i} \neq 0$. Let A' be the matrix obtained from A^{-1} by eliminating row j and column i using Gaussian elimination. Then $A' = (A^{i,j})^{-1}$ (i.e., A' is the Schur complement of $(A^{-1})^{j,i}$).

The above lemma can be used to obtain the following algorithm that finds an f-factor.

Algorithm 1 $O(\phi^3)$ time algorithm for bipartite f-factors.

1: Let $B(G)$ be the $\phi \times \phi$ matrix representing G, f
2: Replace the variables in $B(G)$ by random values from $\mathcal{Z}_p$ for prime $p = \Theta(\phi^2)$ to obtain B
3: If B is singular return "no f-factor"
4: (with probability $\geq 1 - \frac{1}{\phi}$ matrix B is non-singular when $B(G)$ is non-singular) ▷ by Lemma 2.1
5: Compute $N := B^{-1}$ and set $F := \emptyset$
6: (each column i,ι of B^{-1} has an allowed edge, since $BB^{-1} = I$ gives j,κ with $B_{i,\iota,j,\kappa} B^{-1}_{j,\kappa,i,\iota} \neq 0$)
7: **for** $i = [1..n]$ **do**
8: **for** $\iota = [0..f(i)-1]$ **do**
9: Find j,κ such that $ij \in E - F$ and $N_{j,\kappa,i,\iota} \neq 0$
10: ▷ by Corollary 4.3 edge ij is allowed
11: Eliminate j,κ'th row and i,ι'th column from N
12: ▷ using Gaussian elimination
13: ($N = (B^{i,\iota,j,\kappa})^{-1}$, i.e., N encodes $f_{i,j}$-factors)
14: ▷ by Lemma 4.4 in the first iteration
15: Set $F := F + ij$
16: **end for**
17: **end for**
18: Return F

The comment of line 12 is adequate for $\iota = 0$. However $\iota > 0$ requires an additional observation. To see this first recall the logic of each iteration: Let f' be the residual degree requirement function, i.e., the current F, enlarged with an f'-factor of the current graph, gives an f-factor of G. In line 10, the f'-factor F' that contains ij is a subgraph of the graph corresponding to (the current) N and its corresponding matrix B. Now suppose the iteration for $\iota = 0$ adds edge ip to F. When the row and column for ip are deleted from B, the remaining rows for vertex i still contain entries corresponding to edge ip (recall the definition of $B(G)$). So when the iteration for $\iota = 1$ chooses its edge ij, the corresponding f'-factor F' may contain edge ip. But

F cannot be enlarged with F', since that introduces two copies of ip. The same restriction applies to iterations for $\iota > 1$, but now it concerns all previously chosen edges ip.

Actually there is no problem because we can guarantee an f'-factor avoiding all the previous ip's exists. The guarantee is given by the following corollary to Lemma 4.1. (Note when $r = 0$ the corollary is a special case of the lemma. Also, the converse of the corollary holds trivially.)

Corollary 4.5: Consider a set of edges $P = \{ip_1, \ldots, ip_r\}$, $r \geq 0$. Suppose $G - P$ has an f-factor. If G has an f_{ij}-factor for some edge $ij \notin P$ then $G - P$ has an f-factor containing ij.

Proof: The proof of Lemma 4.1 applies, assuming we start by taking F to be the assumed f-factor. ∎

Observe that Algorithm 1 is implementing Gaussian elimination on B^{-1}, the only difference being that pivot elements are chosen to correspond to edges of the graph. If there exists an f-factor, there is an allowed edge incident to each vertex. Hence, even with this additional requirement the elimination is able to find a non-zero element in each row of B^{-1}.

Bunch and Hopcroft [30] show how to speed up the running time of Gaussian elimination from $O(\phi^3)$ to $O(\phi^\omega)$, by using lazy updates: Divide the columns of the matrix into two almost equal parts. Let L denote the first $\lceil \phi/2 \rceil$ columns that are to be eliminated, and R the remaining $\lfloor \phi/2 \rfloor$ columns. Note that columns in R are not used until we eliminate all columns from L.[5] Hence all updates to columns in R resulting from elimination of columns in L can be done once using fast matrix multiplication in $O(|R|^\omega)$ time. By applying this scheme recursively we obtain an $O(\phi^\omega)$ time algorithm.

V. Weighted f-factors

This section discusses how to find a maximum f-factor in a weighted bipartite graph. For simplicity assume in this section that the weight function is non-negative, i.e., $w : E \to [0..W]$. (If not, redefine $w(ij) := w(ij) + W$, changing the weight of each f-factor by exactly $Wf(V)/2$.) Let us start by recalling the dual problem for maximum f-factors. In this problem each vertex v is assigned a real-valued weight $y(v)$. We say that the dual y *dominates* the edge $uv \in E$ when $y(u) + y(v) \geq w(e)$, or it *underrates* the edge $uv \in E$ when $y(u) + y(v) \leq w(uv)$. The dual objective is to minimize

$$y(V, E) = \sum_{v \in V} f(v)y(v) + \sum_{uv \in E \text{ is underrated}} w(uv) - y(u) - y(v).$$

The dual y minimizes $y(V, E)$ when there exists an f-factor F such that F contains only underrated edges while its complement contains only dominated edges. Observe that when we are given the minimum dual y, then the above f-factor F is a maximum weight f-factor. On the other hand, to construct such a maximum f-factor we need to include every strictly underrated edge and arbitrary *tight* edges, i.e., edges $uv \in E$ for which $y(u) + y(v) = w(uw)$. Hence, we can observe the following.

Lemma 5.1: Given an optimal dual function y, a maximum f-factor of a bipartite multigraph can be constructed in $O(\phi^\omega)$ time.

Proof: Let U be equal to the set of underrated edges with respect to y. Set $f'(v) = f(v) - d(v, U)$. Using Algorithm 1 find an f'-factor T over the set of tight edges with respect to y. The maximum f-factor is equal to the multiset sum of U and T, i.e., to $U \uplus T$. ∎

This lemma shows that given an algorithm for finding unweighted f-factors all we need to know is an optimal dual. Such an optimal dual can be obtained from the combinatorial interpretation as given in [14]. Let us define G^+ to be G with additional vertex $s \in V_1$ and new 0 weight edges su, for all $u \in V_0$. In G^+ we set $f(s) = 1$. Let f_v be the degree constraint function defined to be identical to f except for $f_v(v) = f(v) + (-1)^i$, where $v \in V_i$. Let F_v be a maximum f_v-factor in G^+. To show that F_v always exists take F to be any f-factor in G. Now, when $v \in V_0$ then $F + sv$ is an f_v-factor, whereas when $v \in V_1 - s$ then for any $uv \in F$, $F - uv + su$ is an f_v-factor

Theorem 5.2 ([14]): For a bipartite multigraph with an f-factor, optimal duals are given by $y(v) = (-1)^i w(F_v)$ for $v \in V_i$.

Hence to construct the optimal duals we need to know the weights $w(F_v)$ for all $v \in V_0 \cup V_1$. At first sight it might seem we did not gain anything, since instead of finding one F factor we now need to find all factors F_v. But we only need the weights of these factors. In fact we need $w(F_v)$ for just one side of the bipartite graph.

Lemma 5.3: For a bipartite multigraph with an f-factor, let $y(v)$ be an optimal dual for each $v \in V_1$. An optimal dual $y(u)$ for $u \in V_0$ is equal to the largest value y_u that makes at least $f(u)$ edges incident to u underrated, i.e., $|\{uv^{\in E} : y_u \leq w(uv) - y(v)\}| \geq f(u)$.

Proof: Observe that there are at least $f(u)$ underrated edges incident to u with respect to optimal dual y, as each maximum f-factor needs to contain only underrated edges. On the other hand, the complement contains at least $d(u) - f(u)$ dominated edges. This fixes the largest possible value for $y(u)$ as the value given in the lemma. ∎

Now consider a simple bipartite graph G and similar to (1) define $B(G)$ as

$$B(G)_{i,r,j,c} = \begin{cases} z^{w(ij)} x_r^{ij} y_c^{ij} & ij \in E, \\ 0 & \text{otherwise,} \end{cases}$$

where z is a new indeterminate. Theorem 3.2 shows that there is a mapping $\mathcal{F}$ from terms of $\det(B(G))$ onto f-factors in G. Consider a term σ in $\det(B(G))$. Observe that

[5] In their paper the elimination proceeds row by row, whereas it is nowadays more usual to present Gaussian elimination on columns.

its degree in z is equal to the weight of $\mathcal{F}(\sigma)$ because the powers of z get added in the multiplication. For a polynomial p, denote the degree of p in z by $\deg_z(p)$.

Corollary 5.4: For a bipartite graph G, $\deg_z(\det(B(G)))$ equals the weight of a maximum f-factor in G.

To compute f_v-factors in G^+ we use the following auxiliary graph. Let G_* be G^+ with an additional vertex $t \in V_0$ that is joined to every vertex of $V_1 - s$ by an edge of weight zero. Set $f(t) = 1$.

Lemma 5.5: $\forall_{v \in V_1} \deg_z(\mathrm{adj}(B(G_*))_{v,0,t,0}) = w(F_v)$.

Proof: Observe that

$$\mathrm{adj}(B(G_*))_{v,0,t,0} = (-1)^{n(t,0)+n(v,0)} \det(B(G_*)^{t,0,v,0}),$$

where $B(G_*)^{t,0,v,0}$ is the matrix $B(G_*)$ with row $t, 0$ and column $v, 0$ removed and $n(i, r)$ gives the order of rows and columns indexed by pairs i, r. By Theorem 3.2 we know that $\det(B(G_*))$ consists of terms corresponding to f-factors in G_*. Hence the above equality shows terms of $\mathrm{adj}(B(G_*))_{v,0,t,0}$ correspond to f-factors that use edge tv, but with this edge removed. These are exactly the f_v-factors in G^+, because forcing the f-factor to use edge tv effectively decreases $f(v)$ by 1. As we observed in Corollary 5.4 the degree of z equals the total weight of corresponding f-factor. The lemma follows. ∎

Recall that the adjoint of a nonsingular matrix A is $\det(A)A^{-1}$. The lemma shows we are interested in column $t, 0$ of the adjoint. So let $e_{t,0}$ be a unit vector, with 1 in row $t, 0$ and zeroes elsewhere. Then the desired weights are found in the vector $\mathrm{adj}(B(G_*))e_{t,0} = \det(B(G_*))B(G_*)^{-1}e_{t,0}$. This leads to the following algorithm to find optimal duals for weighted bipartite f-factors.

Algorithm 2 Finding optimal duals in a bipartite graph G, f.

1: Let $B(G_*)$ be $\phi \times \phi$ matrix representing G_*
2: Replace x and y variables in $B(G_*)$ by random values from $\mathcal{Z}_p$ for prime $p = \Theta(\phi^3)$ to obtain B
3: Compute vector $a := \mathrm{adj}(B)e_{t,0} = \det(B)B^{-1}e_{t,0}$.
4: ▷ requires $\widetilde{O}(W\phi^\omega)$ time using Theorem 2.2
5: **for** $v \in V_1$ **do**
6: $(\deg_z(a_v) = \deg_z(\mathrm{adj}(B(G_*))_{v,0,t,0}))$
7: ▷ with probability $\geq 1 - \frac{1}{\phi^2}$ by Lemma 2.1
8: $w(F_v) := \deg_z(a_v)$ ▷ equality by Lemma 5.5
9: $y(v) := w(F_v)$ ▷ $y(v)$ is optimal by Theorem 5.2
10: **end for** ▷ by union bound y is correct with pr. $\geq 1 - \frac{1}{\phi}$
11: **for** $u \in V_0$ **do**
12: $y(u) := \max\{y_u : |\{uv^{\in E} : y_u \leq w(uv) - y(v)\}| \geq f(u)\}$
13: **end for** ▷ $y(u)$ is optimal by Lemma 5.3

Combining the above algorithm with Lemma 5.1 we obtain an $\widetilde{O}(Wn^\omega)$ time algorithm for maximum f-factors in weighted bipartite graphs. We note that this development works for bipartite multi-graphs as well – see [26].

VI. Min-cost max-flow

We are given a directed network $N = (V, E)$, with source s and sink t, $s, t \in V$. For convenience let V^- denote the set of nonterminals, $V - \{s, t\}$. The edges and nonterminal vertices have integral capacities given by $c : V^- \cup E \to [1..D]$. Let $g : V \times V \to \mathbb{Z}$ be a flow function. Besides the standard *edge capacity* and *flow conservation* constraints we have *vertex capacity* constraints, i.e., for each vertex $v \neq s, t$ we require $\sum_{u \in V} g(u, v) \leq c(v)$.

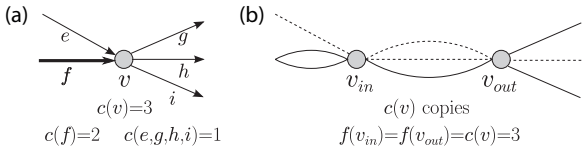

Figure 1. A vertex v of capacity $c(v) = 3$ in N is represented in G_N by two vertices v_{in} and v_{out} connected by 3 edges. The f-factor in (b) is marked with solid edges. Observe that the f-factor must choose the same number of edges going in and out of v.

We begin by constructing a bipartite graph G_N whose maximum f-factor has weight equal to the value of a maximum flow in N. Wlog assume that no edge enters s or leaves t. The construction proceeds as follows: for each $v \in V^-$ place vertices v_{in}, v_{out} in G_N; also place vertices s_{out}, t_{in} in G_N; for each $v \in V^-$ add $c(v)$ copies of edge $v_{in}v_{out}$ to G_N; for each $(u, v) \in E$ add $c(u, v)$ copies of edge $u_{out}v_{in}$ to G_N; for $v \in V^-$, set $f(v_{in}) = f(v_{out}) = c(v)$; set $f(s_{out}) = f(t_{in}) = c(V^-)$; set $w(e) = 1$ for each edge e leaving s_{out} and $w(e) = 0$ for every other edge of G_N; add $c(V^-)$ copies of edge $s_{out}t_{in}$ to G_N, all of weight 0. Note that in addition to the weight 0 edges $s_{out}t_{in}$, G_N may contain edges $s_{out}t_{in}$ of weight 1 corresponding to an edge $st \in E$.

Corollary 6.1 ([20]): Let N be the flow network. The weight of the maximum f-factor in G_N is equal to the maximum flow value in N.

Proof: The main idea of the reduction is shown in Figure 1. Observe that f-factors in G_N correspond to integral flows in N that fulfill the flow conservation constraints. Moreover the edge capacities are not exceeded since an edge cannot be used by an f-factor more times than its capacity. Similarly a vertex cannot be used more times than its capacity. The only edges with non-zero weights are edges incident to s_{out}, so the maximum f-factor maximizes the amount of flow leaving s. Finally all this flow must wind up at t, since any $v \neq s, t$ has an equal number of f-edges incident to v_{in} and v_{out}. ∎

Observe that G_N has $f(V) \leq 4c(V^-)$. So the algorithm of Section V uses $\widetilde{O}(c(V^-)^\omega) = \widetilde{O}((Dn)^\omega)$ time to find a maximum flow in a network.

Now assume that an edge (u, v) of N has a cost $a_{u,v} \in [-W..W]$, i.e., the cost of sending t units of flow on edge (u, v) is linear and equals $ta_{u,v}$. First find the maximum flow value f_{max} in N. Then modify the construction of G_N to $G_{N,a}$ in the following way: add vertices s_{in} and t_{out}; add

$c(V^-)$ copies of edges $s_{in}s_{out}$ and $t_{in}t_{out}$; set $f(s_{in}) = f(t_{out}) = c(V^-)$ and $f(s_{out}) = f(t_{in}) = c(V^-) + f_{max}$; for each copy of the edge $u_{out}v_{in}$ set $w(u_{out}v_{in}) = -a_{u,w}$.

Corollary 6.1 observed that f-factors in G_N correspond to feasible flows in N. The new vertices s_{in} and t_{out} allow an f-factor to model flow along cycles containing s and t. The values of f at the terminals make the corresponding flow from s to t have value f_{max}. Thus f-factors in $G_{N,a}$ correspond to maximum flows in N. Since costs of edges are negated between both networks, we have the following.

Corollary 6.2 ([20]): Let N be a flow network with linear edge costs. A maximum f-factor in $G_{N,a}$ has weight equal to the minimum cost of a maximum flow in N.

Now the algorithm of Section V gives an $\widetilde{O}(Wc(V^-)^\omega) = \widetilde{O}(W(Dn)^\omega)$ time algorithm to find a min-cost max-flow. Let us extend this reduction to convex cost functions. Assume the cost of sending t units of flow on an edge (u, v) is given by a convex function $a_{u,v}(t)$ such that marginal costs satisfy $m_{u,v}(t) = a_{u,v}(t) - a_{u,v}(t-1) \in [-W..W]$. As usual for this scenario we assume $a_{u,v}$ is linear between successive integers. This ensures that there exists an integral optimal solution. We encode such cost functions in the graph by assigning different costs to each copy of an edge, i.e., the kth copy of edge $u_{out}v_{in}$ has cost $w(u_{out}v_{in}, k) = -m_{u,v}(k)$ for $1 \le k \le c(u, v)$.

Lemma 6.3: Let N be a flow network with convex edge costs. A maximum f-factor in $G_{N,a}$ has weight equal to the minimum cost of a maximum flow in N.

Proof: The maximum f-factor in $G_{N,a}$, when using t copies of edges between u_{out} and v_{in}, will use the most expensive copies. The convexity of $a_{u,v}$ implies $m_{u,v}(t)$ is a non-decreasing function. Hence, we use the edges of costs $-m_{u,v}(1), \ldots, -m_{u,v}(t)$, which sum to $-a_{u,v}(t)$. ■

Clearly this extension does not change the running time of our algorithm. So we find a min-cost max-flow for convex edge costs in time $\widetilde{O}(W(Dn)^\omega)$.

VII. The shortest-path structure

This section presents the definition of our generalized shortest-path tree for conservative graphs. It sketches the existence proof and the algorithmic construction. All theorems and proofs are omitted.

Let (G, t, w) be a connected undirected graph with distinguished vertex t and conservative edge-weight function $w : E \to \mathbb{R}$. We wish to find a shortest path P_v from each vertex v to the fixed sink t. As usual $d(v)$ denotes the minimum weight of a vt-path, $d(v) = w(P_v)$. Bellman's inequalities for d needn't hold and a shortest-path tree needn't exist (e.g., the subgraph on $\{a, b, c\}$ in Fig.2(a)). But we show d is optimum for a related set of inequalities (specifically we generalize Bellman's inequalities to the system (3) below).

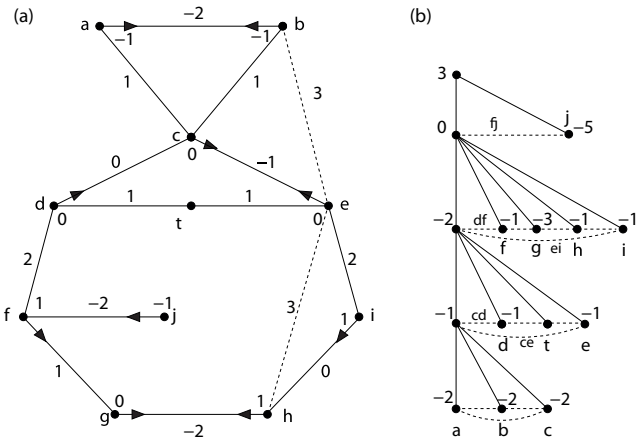

Figure 2. (a) Conservative undirected graph. Vertex labels are shortest-path distances; arrows show the first edge of shortest paths. Dashed edges are not in any shortest path. (b) Shortest-path structure. Node labels are z-values. $E(N)$ edges are the dashed edges joining the children of N.

We begin by defining the analog of the shortest-path tree. When there are no negative edges this analog is a variant of the standard shortest-path tree (node $\mathcal{V}$ below). Fig.2(a)–(b) illustrate the definition. In Fig.2(a) the arrow from each vertex v gives the first edge in v's shortest path. This edge is $e(v)$ in the definition below. More generally it is $e(N)$ for any node N that it leaves, e.g., $ce = e(c) = e(\{a, b, c\})$.

Definition 7.1: A generalized shortest-path tree (gsp-tree) *$\mathcal{T}$ is a tree whose leaves correspond to the vertices of G. For each node N of $\mathcal{T}$, $V(N)$ denotes the set of leaf descendants of N, $V(N) \subseteq V(G)$. Let $\mathcal{V}$ be the root of $\mathcal{T}$. For each node N, $V(N)$ contains a* sink *vertex denoted $t(N)$; for $N \neq \mathcal{V}$, $t(N)$ is the end of an edge $e(N) \in \delta(V(N))$. For $N = \mathcal{V}$ the sink is t, and we take $e(\mathcal{V}) = \emptyset$; for $N \neq \mathcal{V}$, $t(N)$ and $e(N)$ are determined by the parent of N as described below.*

Consider an interior node N of $\mathcal{T}$, with children N_i, $i = 1, \ldots, k$, $k \ge 2$. $V(N_1)$ contains $t(N)$ and $t(N_1) = t(N)$, $e(N_1) = e(N)$. N has an associated set of edges $E(N)$ with $\{e(N_i) : 1 < i \le k\} \subseteq E(N) \subseteq \gamma(N)$. Let $\overline{N}_i$ denote the contraction of $V(N_i)$ in G.

Case *$N \neq \mathcal{V}$: $E(N)$ forms a cycle on vertices $\overline{N}_i$, $i = 1, \ldots, k$.*

Case *$N = \mathcal{V}$: Either (i) $E(N)$ gives a cycle exactly as in the previous case, or (ii) $E(N)$ is a spanning tree on the nodes $\overline{N}_i$, rooted at N_1, with each $e(N_i)$ the edge from N_i to its parent.*

Note that $\{e(N_i) : 1 < i \le k\} = E(N)$ in case (ii) above but not in general, e.g., df in Fig.2(a) has no arrow.

For any vertex v, a top-down traversal of $\mathcal{T}$ gives a naturally defined vt-path $p(v)$ that starts with $e(v)$, that we now describe. As an example in Fig.2(a) $p(j) = j, f, g, h, i, e, c, d, t$; in Fig.2(b) this path is composed of pieces in the subgraphs of 3 nodes, j, f; f, g, h, i, e; and e, c, d, t. In general, if $p(v)$ traverses a node N of $\mathcal{T}$, it does

so on a path from some $x \in V(N)$ to $t(N)$. Specifically if x descends from the child N_i of N, the edges of $p(x) \cap E(N)$ are those of the unique $\overline{N}_i\overline{N}_1$-path P that begins with the edge $e(N_i)$. For any vertex $v \in V(G)$, to find the entire vt-path $p(v)$ start at the root $\mathcal{V}$ of $\mathcal{T}$ and apply this rule recursively, where vertex x is v at $\mathcal{V}$ and x is determined by the two edges incident to N for $N \neq \mathcal{V}$ (one of these edges is $e(N)$).

A *gsp-structure* consists of a gsp-tree plus two functions d, z. Each vertex v has a value $d(v)$. Each node N of $\mathcal{T}$ has a value $z(N)$ that is nonpositive for every $N \neq \mathcal{V}$. Enlarge $E(G)$ to the set $E_\ell(G)$ by adding a loop xx at every vertex except t, with $w(xx) = 0$. For any such x define $E(x)$ to be $\{xx\}$ and define $E(t) = \emptyset$. Say that a node N of $\mathcal{T}$ *covers* any edge with both ends in N (including a loop xx) as well as the edge $e(N)$ (if it exists). d and z are defined by requiring that every edge $xy \in E_\ell(G)$ satisfies

$$d(x) + d(y) + w(xy) \geq \sum\{z(N) : N \text{ covers } xy\}, \quad (3)$$

with equality holding for every edge of $\bigcup(E(N) \cup e(N) : N$ a node of $\mathcal{T})$. The proof that $p(v)$ is a minimum weight vt-path is based on adding the equalities of (3) for the edges of $p(v)$ and the loops of vertices not in $p(v)$, to get $d(v) + \sum\{2d(x) : x \in V - v, t\} + d(t) + w(P_v) = \sum\{|N - t|z(N) : N$ a node of $\mathcal{T}\}$; the analogous sum for any other vt-path is no smaller, since z is nonpositive.

To prove the gsp-structure exists define a *p-cycle* ("planted-cycle") to be the union of a cycle C and 2 copies of a path P from a vertex $c \in C$ to t, with $V(P) \cap V(C) = \{c\}$. (Possibly $c = t$.) For simplicity assume no two sets of edges have the same weight (if not, perturb the edge weights). We prove any graph contains an edge uv such that a minimum weight p-cycle is given by $P_u \cup P_v \cup uv$ if $uv \notin P_u \cup P_v$ or $P_u \cup P_v - uv$ if $uv \in P_u \cap P_v$. This leads to the following *shrinking procedure*; it shows the gsp-tree $\mathcal{T}$ exists by constructing it.

Initialize $\overline{G}$ (the current graph) to the graph $(V, E_\ell(G))$, and $\mathcal{T}$ to contain each vertex of G as a singleton subtree. Then repeat the following step until $\overline{G}$ is acyclic:

> Let a minimum weight p-cycle have cycle C and incident edge cc' on its Ct-path ($cc' = \emptyset$ if $t \in C$). Contract C to form the next graph $\overline{G}$. Set $e(C) = cc'$. Unless C is a loop, create a node in $\mathcal{T}$ with children corresponding to the vertices of C and edge set $E(C)$.

If $\overline{G}$ contains any edges when the loop halts, create a root node of $\mathcal{T}$ whose children and edge set correspond to the vertices and edges of $\overline{G}$ respectively.

Let $\overline{C}$ be the contracted version of C. By convention no loop is incident to $\overline{C}$ in the new graph. The weights of edges incident to $\overline{C}$ are modified to ensure that weights of shortest paths to t do not change. Specifically these weights are changed so (a) any shortest path starting at or passing through $\overline{C}$ uses edge $e(C)$; (b) the weight of edges used inside C is included in the new weight.

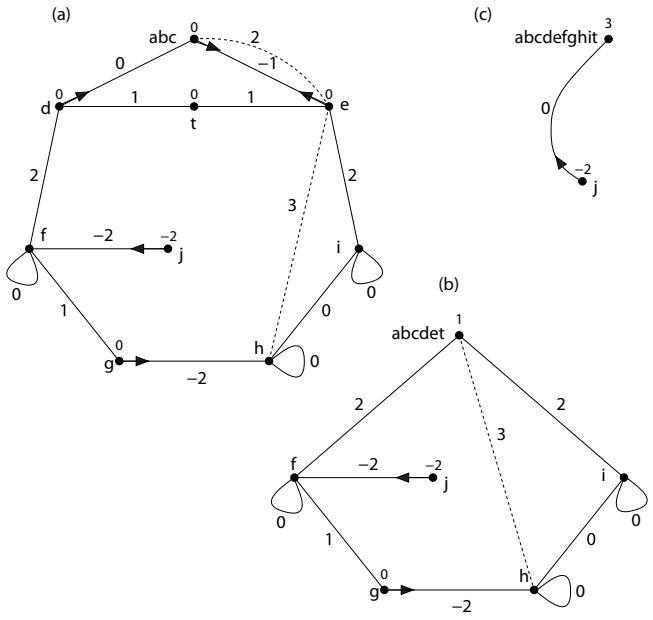

Figure 3. Execution of the shrinking procedure on graph of Fig. 2: the contracted graph after the last iteration for ζ^* equal to (a) 0; (b) 1; (c) 3.

We complete the gsp-structure by defining z. For each node N of $\mathcal{T}$ let ζ_N be the weight of its corresponding p-cycle. (For a leaf x use the p-cycle with cycle xx; for the root $\mathcal{V}$ of $\mathcal{T}$ use the last p-cycle to be shrunk.) Let p be the parent function in $\mathcal{T}$. For each node N define

$$z(N) = \begin{cases} \zeta_N & N = \mathcal{V} \\ \zeta_N - \zeta_{p(N)} & N \neq \mathcal{V}. \end{cases}$$

Fig.3 illustrates how this procedure constructs the gsp-structure of Fig.2(b). Let ζ^* denote the weight of the procedure's p-cycle. ζ^* never decreases from one iteration to the next. So we describe the p-cycles found at the various values of ζ^*.

> $\zeta^* = -2$: The first three iterations find p-cycles of weight $\zeta^* = -2$. Their cycles are the loops at a, b and j (e.g., the p-cycle for j consists of loop jj (multiplicity 1) plus the jt-path of weight -1 (multiplicity 2). The weight -2 of this p-cycle is recorded at j in Fig.3(a). The cycle jj has $e(jj) = jf$, which is indicated by the arrow on this edge. Fig.3 indicates all p-cycle weights and e-edges this way.
>
> $\zeta^* = 0$: The next five iterations find p-cycles of weight 0. Their cycles are the loops at d, e, g, t, and cycle (a, b, c). Fig.3(a) shows the graph after all these p-cycles have been shrunk. The dashed edge corresponding to be has decreased in weight from 3 to 2.

This accounts for the weight -1 path b, a, c in the contraction of a, b, c. The iteration for cycle (a, b, c) forms the interior node at depth 3 in Fig.2(b).

$\zeta^* = 1$: Contracted vertex $\{a, b, c\}$ plus vertices d, e, t form the next cycle to be shrunk, giving Fig.3(b). The interior node at depth 2 in Fig.2(b) is formed.

$\zeta^* = 2$: The loops at f, h, and i give p-cycles.

$\zeta^* = 3$: All vertices but j form a p-cycle. Fig.3(c) shows the graph after it is shrunk. The new weight 0 on the image of edge jf is its original weight -2 plus the weight 2 of the contracted edge fd. The interior node at depth 1 in Fig.2(b) is formed. Now the loop halts and the root of Fig.2(b) is formed.

The last p-cycle, as illustrated in Fig.3(b), can be viewed as $P_g + P_h - gh$ where $gh \in P_g \cap P_h$, as well as $P_d + P_f + df$ where $df \notin P_d \cup P_f$ (here "d" stands for the image of the original vertex d). This illustrates the above characterization of the minimum weight p-cycle. But in the original graph Fig.2(a), $P_g + P_h - gh = P_d + P_f + df$ is not a planted-cycle!

Our efficient algorithm for the gsp-structure has three subroutines. First we find the distance function d, using either a determinant-based algebraic algorithm or a combinatoric weighted-matching algorithm. Using d, a second routine finds z and a contraction of the gsp-tree $\mathcal{T}$. Specifically each node of z-value 0 is contracted into its parent. The simple cycles of $\mathcal{T}$ get lost in this contraction. The third routine recovers them, using an algorithm similar to maximum cardinality matching.

These ideas extend to maximum weight f-factors. The p-cycle generalizes to a "$2f$-unifactor". Its cycle generalizes to a blossom circuit C and its incident edge $e(N)$ generalizes to a set of edges I incident to C. Just as shortest paths P satisfy the condition $|(P \cap \delta(V(N))) \oplus e(N)| \leq 1$, maximum factors F satisfy $|(F \cap \delta(V(C))) \oplus I| \leq 1$, although the possibility $|F \cap \delta(V(C))| = |I| - 1$ is no longer trivial.

ACKNOWLEDGMENT

The second author was supported by the ERC StG project PAAl no. 259515 and Foundation for Polish Science.

REFERENCES

[1] A. Schrijver, *Combinatorial Optimization - Polyhedra and Efficiency*. Springer-Verlag, 2003.
[2] L. Lovász, "On determinants, matchings and random algorithms," in *FCT*, 1979, pp. 565–574.
[3] M. Mucha and P. Sankowski, "Maximum matchings via Gaussian elimination," in *Proc. of FOCS'04*, pp. 248–255.
[4] N. J. A. Harvey, "Algebraic algorithms for matching and matroid problems," *SIAM J. Comput.*, vol. 2, no. 39, pp. 679–702, 2009.
[5] V. V. Williams, "Multiplying matrices faster than Coppersmith-Winograd," in *Proc. STOC'12*, pp. 887–898.
[6] H. N. Gabow, "An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems," in *Proc. of STOC'83*, pp. 448–456.
[7] W. Pulleyblank, "Faces of matching polyhedra," Ph.D. dissertation, University of Waterloo, Ontario, Canada, 1973.
[8] A. B. Marsh, "Matching algorithms," Ph.D. dissertation, The John Hopkins Univeristy, Baltimore, 1979.
[9] R. Anstee, "A polynomial algorithm for b-matching: An alternative approach," *IPL*, vol. 24, pp. 153–157, 1987.
[10] R. Urquhart, "Degree-constrained subgraphs of linear graphs," Ph.D. dissertation, University of Michigan, 1967.
[11] H. N. Gabow, "Data structures for weighted matching and nearest common ancestors with linking," in *Proc. of SODA'90*, pp. 434–443.
[12] P. Sankowski, "Maximum weight bipartite matching in matrix multiplication time," *Theoretical Computer Science*, vol. 410, no. 44, pp. 4480–4488, 2009.
[13] M. Cygan, H. N. Gabow, and P. Sankowski, "Algorithmic applications of Baur-Strassen's theorem: shortest cycles, diameter and matchings," in *Proc. of FOCS'12*, pp. 531–540.
[14] H. N. Gabow, "A combinatoric interpretation of dual variables for weighted matching and f-factors," *Theoretical Computer Science*, vol. 454, pp. 136–163, 2012.
[15] J. Edmonds, "Maximum matching and a polyhedron with 0,1-vertices," *Journal of Research National Bureau of Standards-B*, vol. 69B, pp. 125–130, 1965.
[16] R. Yuster and U. Zwick, "Answering distance queries in directed graphs using fast matrix multiplication," in *Proc. of FOCS'05*, 2005, pp. 389–396.
[17] P. Sankowski, "Shortest paths in matrix multiplication time," in *Proc. of ESA'05*, pp. 770–778.
[18] A. Sebö, "Undirected distances and the postman-structure of graphs," *J. Combin. Theory Ser. B*, vol. 49, no. 1, pp. 10–39, 1990.
[19] ——, "Potentials in undirected graphs and planar multiflows," *SIAM J. Comput.*, vol. 26, no. 2, pp. 582–603, 1997.
[20] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for network problems," *SIAM Journal on Computing*, vol. 18, no. 5, pp. 1013–1036, 1989.
[21] H. Y. Cheung, L. C. Lau, and K. M. Leung, "Graph connectivities, network coding, and expander graphs," in *Proc. of FOCS'11*, pp. 190–199.
[22] J. Cheriyan, T. Hagerup, and K. Mehlhorn, "Can a maximum flow be computed in o(nm) time?" in *ICALP'90*, 1990, pp. 235–248.
[23] A. V. Goldberg and S. Rao, "Beyond the flow decomposition barrier," *J. ACM*, vol. 45, no. 5, pp. 783–797, Sep. 1998.
[24] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM*, vol. 19, no. 2, pp. 248–264, 1972.
[25] J. B. Orlin, "A faster strongly polynominal minimum cost flow algorithm," in *Prof. of STOC'88*, pp. 377–387.
[26] H. N. Gabow and P. Sankowski, "Algebraic algorithms for b-matching, shortest undirected paths, and f-factors," *CoRR*, vol. abs/1304.6740, 2013.
[27] R. Zippel, "Probabilistic algorithms for sparse polynomials," in *Proc. of EUROSAM'79*, 1979, pp. 216–226.
[28] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *J. ACM*, vol. 27, pp. 701–717, 1980.
[29] A. Storjohann, "High-order lifting and integrality certification," *J. Symbolic Comput.*, vol. 36, no. 3-4, pp. 613–648, 2003.
[30] J. Bunch and J. Hopcroft, "Triangular factorization and inversion by fast matrix multiplication," *Mathematics of Computation*, vol. 28, no. 125, pp. 231–236, 1974.