PARALLEL NESTED DISSECTION FOR PATH ALGEBRA COMPUTATIONS

Victor PAN *

Computer Science Department, SUNYA, Albany, NY 12222, USA

John REIF **

Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138, USA

[illegible]

This paper extends the authors' parallel nested dissection algorithm of [13] originally devised for solving sparse linear systems. We present a class of new applications of the nested dissection method, this time to path algebra computations (in both cases of single source and all pair paths), where the path algebra problem is defined by a symmetric matrix A whose associated graph G with n vertices is planar. We substantially improve the known algorithms for path algebra problems of that general class; this has further applications to maximum flow and minimum cut problems in an undirected planar network and to the feasibility testing of a multicommodity flow in a planar network.

graph computations * path algebras * parallel algorithms * network flow

1. Introduction

In this paper we substantially improve the known parallel algorithms for several problems of practical interest which can be reduced to path algebra computations. Gondran and Minoux [4, pp. 41–42, 75–81] list the applications of path algebras to the problems of: vehicle routing, investment and stock control, dynamic programming with discrete states and discrete time, network optimization, artificial intelligence and pattern recognition, labyrinths and mathematical games, encoding and decoding of information; compare also Lawler [8], Tarjan [17,18]. [4, pp. 84–102] thoroughly investigates general algorithms for such problems based on matrix operations in *dioids*, see next sections. We propose a substantial improvement of these general algorithms in the important case where the input matrix A is associated with an undirected planar graph or, more generally, with a graph from the class of graphs having small separator families; see Definition 2 of Section 4 and compare Lipton, Rose and Tarjan [10], Pan and Reif [13,15]. Our improvement relies on our extension of the nested dissection parallel algorithm of [13] to path algebra problems (originally the algorithm was applied in [13] to linear systems of equations to extend the sequential algorithm of [10] for the same problem; then, in [15], the algorithm was extended to the least-squares and linear programming computations). Our new extension is somewhat surprising because the divisions and subtractions of the original algorithm of [13] are not generally allowed in the dioid; furthermore for that reason we can extend (to dioids) neither the special recursive factorization of the input matrix A from [13] nor Choleski's factorization of A from [10], but we do extend the special recursive factorization of the inverse matrix A^{-1} of [13] to the similar factorization of the *quasi-inverse* A^*. The quasi-inverse A^* can be a dense matrix, so (unlike [4] and like [13] and [15]) we avoid explicitly computing A^* and exploit its recursive factorization when we need to solve the single source path problems.

As in [4], we define the algorithms over dioids

* Supported by NSF Grant DCR-8507573.

** Supported by Office of Naval Research Contract N00014-80-C-0647 and by NSF Grant DCR-850351.

(semirings); respectively, we estimate the computational cost in terms of dioid operations. We assume a customary machine model of parallel computation, where on every parallel step each processor performs at most one operation of the considered class; in our case this means at most one operation of the dioid. In the major specific applications to the classes of the problems of existence, optimization and counting (see the next section), an operation over a dioid is an addition, a multiplication or a comparison of two numbers; the numbers involved in the computation by our algorithms require about the same precision (number of binary digits) as the input values.

Table 1 shows our substantial improvement of the known algorithms in both sequential and parallel settings. (In case of parallel algorithms we assume that G is given with its $O(\sqrt{n})$-separator family.) Further applications lead, in particular, to computing a maxflow and a mincut in an undirected planar network using $O(\log^4 n)$ parallel steps, $n^{1.5}/\log n$ processors or alternatively $O(\log^3 n)$ steps, $n^2/\log n$ processors, versus the known bounds, $O(\log^2 n)$ and n^4, of Johnson and Venkatesan [6].

The estimates of that table hold in case of planar graphs and of all graphs having an $O(\sqrt{n})$-separator family (see definition in Section 4); that family is assumed given or readily computable. In some cases (*grid graphs*) the separator family is defined immediately. For a planar graph such a family can be computed in $O(n)$ sequential time using the algorithm of Lipton and Tarjan [9]. $O(n)$ is a minor contribution to the total sequential computational complexity of the problem, for which we have the bounds $O(n^2)$ (all pairs) and $O(n^{1.5})$ (single source); see Table 1. Thus our algorithms substantially improve the known sequential time estimates for the most general path algebra computations for planar graphs. They also decrease the bounds on the parallel complexity of such computations provided that the separator family is available. In many cases several computations must be performed for the same graph G, having variable edge weights; then one may precompute an $O(\sqrt{n})$-separator family of G, provided that there exists such a family. (In case of a planar graph that preconditioning can be reduced to the breadth first search (Miller [11]).) Whenever an $O(\sqrt{n})$-separator family of G has been precomputed, we may apply parallel algorithms to solve general path algebra problems using polylogarithmic time and having processor bounds less than the sequential time bound; see Table 1.

In particular all the bounds of the table can be applied to the shortest path computation in an undirected planar graph (network) G. This does not improve the known sequential time bounds for that specific problem, $O(n\sqrt{\log n})$ (single source) and $O(n^2)$ (all pairs) (Fredericson [3]). However, our parallel complexity bounds, $O(\log^3 n)$ parallel steps and $n^{1.5}/\log n$ processors (single source shortest paths) and $n^2/\log n$ processors (all pair shortest paths), substantially improve the known estimates (so far the polylog time parallel algorithms for both problems of all pair shortest paths and single source shortest paths in planar graphs require n^3 processors, as in the general paths computation; compare Table 1). Furthermore, our parallel algorithms for shortest paths in planar graphs, combined with the results of Klein and Reif [7] lead to a new parallel algorithm for the evaluation of a maximum flow and a minimum cut in G using $O(\log^4 n)$ steps and $n^{1.5}/\log n$ processors or alternatively $O(\log^3 n)$ steps and $n^2/\log n$ processors, to compare with the previous bounds of $O(\log^2 n)$ steps, n^4 processors (Johnson and Venkatesan [6]). Further applications can be obtained, in particular to feasibility testing of a multicommodity flow in a planar network; see [14].

In the next section we define dioids and state some path algebra problems; in Section 3 we estimate the computational cost of solving those problems in case of general graphs. (In both Sec-

Table 1

	Previous algorithms [4]			New algorithms		
	Sequential time	Parallel time	Processors	Sequential time	Parallel time	Processors
Single source	$O(n^2)$	$O(\log^2 n)$	n^3	$O(n^{1.5})$	$O(\log^3 n)$	$n^{1.5}/\log n$
All pair	$O(n^3)$	$O(\log^2 n)$	n^3	$O(n^2)$	$O(\log^3 n)$	$n^2/\log n$

tions 2 and 3 we follow [4].) In Section 4 we present our improvement of the known algorithms for those problems for the graphs having small separator families. In Section 4 we specify parallel algorithms for the shortest path problems in an undirected planar network with the applications to computing maxflow and mincut in Section 5.

2. Path algebra problems

We start with the special case of the shortest path problem in a graph G with n vertices defined by an $n \times n$ matrix $A = [a_{ij}]$ of non-negative arc lengths, where $a_{ij} = \infty$ if there is no arc between the vertices i and j in G. (A is a symmetric matrix if G is an undirected graph.) We seek the vector $x = [x(i)]$ of distances $x(i)$ (the lengths of the shortest paths) from vertex 1 to all vertices i in G. This is the *single source shortest path problem* (s.s.s.p.p). The distances satisfy the system, $x(1) = 0$, $x(i) = \min_j(x(j) + a_{ji})$, $i = 2, \ldots, n$, or equivalently,

$$x(1) = \min\Big(\min_j\,(x(j) + a_{j1}),\, 0\Big),$$
$$x(i) = \min\Big(\min_j\,(x(j) + a_{ji}),\, \infty\Big),$$
$$i = 2, \ldots, n.$$

We substitute $\oplus$ for min and $*$ for $+$ and rewrite that system as follows:

$$x(1) = \sum_j^{\oplus} x(j) * a_{j1} \oplus 0,$$
$$x(i) = \sum_j^{\oplus} x(j) * a_{ji} \oplus \infty, \qquad i = 2, \ldots, n,$$

or, in matrix notation, denoting $i^{(1)} = [0, \infty, \ldots, \infty]$,

$$x = x * A \oplus i^{(1)}. \tag{1}$$

Similarly, seeking the matrix $X = [x(i, j)]$ of distances between all pairs of vertices in G (this is *the all pair shortest path problem*, a.p.s.p.p.) and denoting $I = [\delta_{ij}]$, $\delta_{ii} = 0$, $\delta_{ij} = \infty$ if $i \neq j$, we arrive at the following matrix equation:

$$X = X * A \oplus I. \tag{2}$$

Restricting (2) to the hth row we arrive at the s.s.s.p.p. of computing the distances from the vertex h to all vertices in G (for $h = 1$ we arrive back at (1)), so an a.p.s.p.p. can be reduced to n s.s.s.p.p.'s.

The known algorithms of linear algebra can be extended to solve the systems (1) and (2); this turns most of them into known algorithms for the s.s.s.p.p. and/or the a.p.s.p.p. Here are two examples.

Algorithm 1. Set $x^{(0)} = i^{(1)}$; compute $x^{(k+1)} = x^{(k)} * A \oplus i^{(1)}$, $k = 0, 1, \ldots$ until $x^{(k+1)} = x^{(k)}$; then output the vector $x = x^{(k)}$ satisfying (1).

Algorithm 1 extends Jacobi's method of linear algebra and amounts to the algorithm of Bellman [1] for the s.s.s.p.p.

Algorithm 2. (a) Set $A^{[0]} = A$.

(b) For $k = 0, 1, \ldots, n-1$, compute $a_{ij}^{[k+1]} = a_{ij}^{[k]} \oplus a_{ik}^{[k]} * a_{kj}^{[k]}$, $i, j = 1, \ldots, n$.

(c) Output $X = A^{[n]} \oplus I$. (The matrix X satisfies (2).)

Algorithm 2 extends Jordan's algorithm of linear algebra and amounts to the algorithm of Floyd [2] for the a.p.s.p.p.; compare also Algorithm 4 (in Section 4).

Other problems listed above can be also reduced to the linear systems (1) or (2) or to similar computations. We need to recall the general concept already implicitly used in our reduction of the s.s.s.p.p. to (1) and of the a.p.s.p.p. to (2).

Definition 1. A *dioid* (sometimes called a *semiring*) is a set S with two operations, $\oplus$ and $*$, such that for any triplet of elements a, b, $c \in S$ and for two special elements e (unity) and ϵ (zero) of S, the following equations hold:

$$a \oplus b = b \oplus a \in S, \quad (a \oplus b) \oplus c = a \oplus (b \oplus c),$$
$$a \oplus \epsilon = a,$$
$$a * b \in S, \quad (a * b) * c = a * (b * c),$$
$$a * e = e * a = a, \quad a * \epsilon = \epsilon * a = \epsilon,$$
$$a * (b \oplus c) = (a * b) \oplus (a * c),$$
$$(b \oplus c) * a = (b * a) \oplus (c * a).$$

In the above reduction of the s.s.s.p.p. to (1) and of the a.p.s.p.p. to (2), we used the dioid where $S = \mathbb{R} \cup \infty$, $\mathbb{R}$ being the set of real numbers, $\oplus = \min$, $* = +$, $e = 0$, $\epsilon = \infty$. Generalizing

(1) and (2) to arbitrary dioids we define that

$$i^{(1)} = [e, \epsilon, \dots, \epsilon], \quad I = [\delta_{ij}], \quad \delta_{ii} = e,$$
$$\delta_{ij} = \epsilon \quad \text{if} \quad i \neq j. \tag{3}$$

Let us list some classes of path problems, which can be reduced to solving the systems (1) and (2) or to some similar matrix operations in appropriate dioids:

(i) existence (problems of connectivity),
(ii) enumeration (elementary paths, multicriteria problems, generation of regular languages),
(iii) optimization (paths of maximum capacity, paths with minimum number of arcs, shortest paths, longest paths, paths of maximum reliability, reliability of a network),
(iv) counting (counting of paths, Markov chains),
(v) optimization and post-optimization (problems of kth path, η-optimal paths).

Specifically, the class (i) includes the problems of

(a) the existence of paths having k (or at most k) arcs (for a given k) between vertices i and j in a given (di)graph G,
(b) computing the transitive closure of G,
(c) testing G for being strongly connected and for having circuits.

An appropriate dioid for problems of class (i) is the Boolean algebra, $S = \{0, 1\}$, $\oplus = \max$, $\circledast = \min$, $\epsilon = 0$, $e = 1$, and in the incidence matrix $A = [a_{ij}]$ of G, $a_{ij} = 1$ if and only if $\{i, j\}$ is an arc of G.

The subclass of shortest path problems in (iii) includes s.s.s.p.p., a.p.s.p.p. (also in the versions where the shortest paths are required to have k or at most k arcs) and testing a graph for having circuits of negative lengths.

Class (iv) includes counting the numbers of

(a) distinct paths having k (or at most k) arcs between i and j in G,
(b) all the distinct paths between i and j in G,
(c) all the circuits in G having a given number of arcs.

For that class we choose the dioid where S is the set of integers, $\oplus = +$, $\circledast = *$ (that is, $\oplus$ and $\circledast$ are the conventional addition and multiplication, respectively) $\epsilon = 0$, $e = 1$; $A = [a_{ij}]$, $a_{ij} = 1$ if and only if $\{i, j\}$ is an arc of G; see further comments in [4, pp. 91, 94–102].

3. Computational complexity of path problems for general graphs

The solution of most of the path problems listed in the previous section can be reduced to the evaluation (over the dioid) of the entries of the matrix $A^{(k)}$ (the all pair path problems) or of the vector $bA^{(k)}$ (the single source path problems) for some positive k, usually for $k = n - 1$. Here

$$A^{(q+1)} = A^{(q)} \oplus A^{q+1}, \qquad q = 0, 1, \dots,$$

$A^{(0)} = I$ (see (3)), A is an $n \times n$ input matrix, $b = i^{(h)}$ is a fixed coordinate vector of dimension n. Here and hereafter we assume that all computations, in particular computing matrix sums, products and powers, are performed over the dioid associated with a given path problem. For every incidence matrix A of the path problems listed in the previous section there exists the *quasi-inverse* matrix

$$A^* = \lim_{q \to \infty} A^{(q)}, \tag{4}$$

except for the shortest path and multicriteria problems where there exist circuits of negative lengths in G and for counting problems where there exists a circuit in G. In both latter cases the existence of such circuits is detected via computing A^k or $(I \oplus A)^k$ over the dioids. Hereafter we will consider the most important case where there exists A^*, the quasi-inverse of A, and moreover where

$$A^* = A^{(n-1)}, \quad A^{(q)} = A^{(q-1)} \quad \text{for} \quad q \geq n. \tag{5}$$

(Our estimates of this section for the cost of the evaluation of A^* and bA^* under (5) can be immediately extended to the case of the evaluation of A^q, $A^{(q)}$ and $bA^{(q)}$ for $q \neq n - 1$.) The eqs. (5) arise naturally where A^* is the incidence matrix of the transitive closure of the graph of A (this is the case for the existence and connectivity problems). (5) implies that

$$A^* = I \oplus A \oplus A^2 \oplus \cdots \oplus A^{n-1}$$
$$A^* = (I \oplus A)(I \oplus A^2)(I \oplus A^4)\dots(I \oplus A^{2^k}),$$
$$k = \lfloor \log_2 n \rfloor.$$

Thus $k + 1$ matrix additions and $2k$ matrix multiplications suffice, which means $(4nk - k + 1)n^2$ operations in the dioid. (The known fast matrix multiplication algorithms, see Pan [12], cannot be generally applied over dioids.) For many dioids the operation $\oplus$ is idempotent, that is, $a \oplus a = a$ for all $a \in S$. In that case

$$A^* = \sum_{r=0}^{n} A^r = \sum_{r=0}^{n} A^r r!/n!(n-r)! = (I \oplus A)^n$$
$$= (I \oplus A)^{2^k}$$

(where Σ denotes a sum in the dioid, $k = \lceil \log_2 n \rceil$), so A^* can be computed (via repeated squaring of $I \oplus A$) using only $k-1$ matrix multiplications and a single addition of the two matrices A and I, that is, using a total of $n^2(k-1)(2n-1)+n$ operations. It is easy to parallelize these two known algorithms, which yields rather efficient parallel scheme, with $O((k-1)\log n)$ steps, $\lceil 2n^3/\log n \rceil$ processors, for the evaluation of A^* where A is a dense matrix. If A is sparse, then the above ways are relatively less effective for the sparsity of A is not generally preserved during the computation.

Computing bA^* (the single source path problems) can be reduced to n successive postmultiplications of the vectors $b\Sigma_{r=0}^{k}A^r$ by the matrix A for $k = 0, 1, \ldots, n-1$ (and to $n-1$ vector additions in the case where the operation $\oplus$ is not idempotent in the dioid), that is, to a total of $(2D(A)-n)(n-1)$ operations in the dioid (or of $2D(A)(n-1)$ operations in the case of non-idempotent $\oplus$), provided that the operations in the dioid are not counted if at least one of the operands is zero, ϵ. Here $D(A)$ denotes the number of non-zero entries of A. This way we exploit sparsity of A to some extent; note, however, that the above estimates translate into $O(n \log n)$ parallel steps (substantially more than $O(\log^2 n)$ achievable even in the dense case) and $D(A)$ processors and that n times more operations and processors are needed to extend this to computing A^*.

4. Solving path problems for graphs with small separators

Definition 2. Compare [10] and [13]. $G = (V, E)$ has an *$s(n)$-separator family* if either $|V| \le n_0$ (for a constant n_0) or deleting some separator set S of vertices, such that $|S| \le s(|V|)$, partitions G into two disconnected subgraphs with the vertex sets V_1 and V_2, such that $|V_i| \le \alpha |V|$, $i = 1, 2$, α is a constant, $\alpha < 1$, and furthermore each of the two subgraphs of G defined by the vertex sets $S \cup V_i$, $i = 1, 2$, also has an $s(n)$-separator family.

The grid graphs on a d-dimensional hypercube have $(n^{1-1/d})$-separator families, which are readily available; in particular, square grids have $\sqrt{n}$-separator families. An undirected planar graph has a $\sqrt{8n}$-separator family, which can be computed in $O(n)$ time [9].

In this section we consider a path problem for an undirected graph $G = (V, E)$ given together with its $s(n)$-separator family, $s(n) = n^\sigma$, $\sigma < 1$. In that case we may further reduce the computational cost of computing A^* and bA^* using the nested dissection algorithms of [10] for sequential computation and of [13] for parallel computation.

The nested dissection algorithms of [10] and [13] require to invert some auxiliary matrices. The dioid elements and matrices in dioids may have no inverses, but we compute quasi-inverses of matrices over dioids applying the following generalized Jordan elimination algorithm, which requires only the operations $\oplus$ and $\ast$ [4, p. 110].

Algorithm 3 (evaluation of A^*). Set $A^{[0]} = A$, $B^{[0]} = I$ and recursively compute $A^{[k]} = M^{[k]}A^{[k-1]}$, $B^{[k]} = M^{[k]}B^{[k-1]}$ for $k = 1, 2, \ldots, n$. Here $M^{[k]}$ is obtained from the matrix I of (3) by replacing the (k, k) entry of I by $(a_{kk}^{[k-1]})^*$ and by replacing other entries of the kth column of I by the entries of the vector $[a_{ik}^{[k-1]} \ast (a_{kk}^{[k-1]})^*]$ where $i, k = 1, \ldots, n$. Output $B^{[n]}$.

Algorithm 3 extends the Jordan elimination scheme for linear systems, avoiding subtractions and replacing the reciprocals $1/(1 - a_{kk}^{[k-1]})$ by $(a_{kk}^{[k-1]})^*$.

Lemma 1. *Let $B^{[n]}$ be computed by Algorithm 3. Then $A^* = B^{[n]}$.*

Lemma 1 is proven in Pan and Reif [14] using in particular the next simple lemma (which can be immediately verified; see (4)).

Lemma 2. *A^* satisfies* (2).

$A^{[n]} = B^{[n]}A$ (obvious), therefore $A^* = A^{[n]} \oplus I$, so we may dispense with the evaluation of $B^{[k]}$ and simplify Algorithm 3 as follows.

Algorithm 4 (evaluation of A^*).

```
(a) Set $A^{[0]} = A$.
(b) For $k$ from 1 to $n$,
    $a_{kk}^{[k]} = (a_{kk}^{[k-1]})^*$,
    $a_{ij}^{[k]} = a_{ij}^{[k-1]} \oplus a_{ik}^{[k-1]} * a_{kk}^{[k]} * a_{kj}^{[k-1]}$
    for all $i$, $j$ except for $i = j = k$.
(c) Output $A^* = A^{[n]} \oplus I$.
```

Algorithm 4 turns into Algorithm 2 of the Introduction for the dioids where $a_{kk}^{(k)} = e$; this includes the dioid associated with the shortest path problems.

To compute $A^* = P^{\mathrm{T}} A_0^* P$ in parallel, we use *recursive factorization* for the matrix $A_0^* = (PAP^{\mathrm{T}})^*$, extending the recursive factorization from Section 4 of [13] for the inverse A_0^{-1}. Here P is the permutation matrix obtained at the ordering stage of the nested dissection algorithm of [13] applied to the input matrix A. Here and hereafter W^* and W^{T} denote the quasi-inverse and the transpose of a matrix W, respectively; O denotes the null matrix, filled with the zeros ϵ; I denotes the identity matrices (3) of appropriate sizes. Here is our recursive factorization, where $h = 0, 1, \ldots, d-1$.

$$A_h = \begin{bmatrix} X_h & Y_h^{\mathrm{T}} \\ Y_h & Z_h \end{bmatrix}, \quad A_{h+1} = Z_h \oplus Y_h X_h^* Y_h^{\mathrm{T}}, \tag{6}$$

$$A_h^* = \begin{bmatrix} I & X_h^* Y_h^{\mathrm{T}} \\ O & I \end{bmatrix} \begin{bmatrix} X_h^* & O \\ O & A_{h+1}^* \end{bmatrix} \begin{bmatrix} I & O \\ Y_h X_h^* & I \end{bmatrix}. \tag{7}$$

Let us verify that (7) indeed defines A_h^*. Expand the right side of (7) deleting h and replacing $h+1$ by 1 in the subscripts, to simplify the notation,

$$W = \begin{bmatrix} X^* \oplus X^* Y^{\mathrm{T}} A_1^* Y X^* & X^* Y^{\mathrm{T}} A_1^* \\ A_1^* Y X^* & A_1^* \end{bmatrix}. \tag{8}$$

Lemma 3 ([14], compare Remark 2 below). *Let $A = [\begin{smallmatrix} X & Y^{\mathrm{T}} \\ Y & Z \end{smallmatrix}]$ and W be defined by* (8). *Let there exist the quasi-inverses A_1^*, X^*. Then $WA \oplus I = W$.*

Lemma 3 implies that $W = A^*$ under (8) (provided that there exist quasi-inverses A_1^*, X^*). This substantiates the validity of the recursive factorization (6), (7). (Note that computing a^* for $a \in S$ may require more than one operation in a dioid unless (5) holds; on the other hand, we may compute a^* as $(e - a)^{-1}$ in the dioids that have inverse operations to $\oplus$ and $*$.)

Remark 1. The proofs of Lemmas 1 and 3 and consequently of the validity of Algorithms 3, 4 and of the recursive factorization (6), (7) do not require to assume (5). It is sufficient to use the definition (4) of quasi-inverse and to assume the existence of all the quasi-inverses included in Algorithms 3, 4 and in the recursive factorization (6), (7).

Remark 2. The matrix equation (7) can be reduced to Algorithm 3. Indeed rewrite (7) as follows:

$$A_h^* = \begin{bmatrix} I & X_h^* Y_h^{\mathrm{T}} A_{h+1}^* \\ O & A_{h+1}^* \end{bmatrix} \begin{bmatrix} X_h^* & O \\ Y_h X_h^* & I \end{bmatrix}.$$

Let $n = 2$ and let A_h replace A in Algorithm 3. Then Algorithm 3 computes just the latter factorization applied to the 2×2 block matrix A_h.

Remark 3. In [4] the fact that the solution $X = B^{[n]}$ of (2) equals A^* (see Lemma 1) is stated under the additional assumption that the preorder relation in the dioid ($a \le b$ if and only if there exists c such that $a \oplus c = b$) is the order relation ($a \le b$ and $a \ge b$ together imply that $a = b$). Under that assumption, [4] suggests (with the proof omitted) that $B^{[n]}$ is the minimum solution. This implies that $B^{[n]} = A^*$ for A^* is easily proven to be the minimum solution to (2); in Lemma 1 we prove that $B^{[n]} = A^*$ even where the order relation in the dioid is not assumed.

Remark 4. (6), (7) generalize the recursive factorization from [13] based on the following factorization of $A_0 = PAP^{\mathrm{T}}$ (which itself, however, does not seem to be extendable to the case of dioids):

$$A_h = \begin{bmatrix} X_h & Y_h^{\mathrm{T}} \\ Y_h & Z_h \end{bmatrix}, \quad Z_h = A_{h+1} + Y_h X_h^{-1} Y_h^{\mathrm{T}},$$

$$A_h = \begin{bmatrix} I & O \\ Y_h X_h^{-1} & I \end{bmatrix} \begin{bmatrix} X_h & O \\ O & A_{h+1} \end{bmatrix} \begin{bmatrix} I & X_h^{-1} Y_h^{\mathrm{T}} \\ O & I \end{bmatrix}.$$

Next we estimate the costs of computing bA^* and A^*. We proceed similarly to [13], noting that for some auxiliary $s(\alpha^h n) \times s(\alpha^h n)$ block-matrices B (where $\alpha < 1$, $h = k, k-1, \ldots, 0$, $k = O(\log n)$) we need to compute $B^* v$, v being a fixed vector. It is easy to extend the assumed property that $A^{(q+1)}$

$= A^{(q)}$ for $q \geq n-1$ to the equations $B^{(q+1)} = B^{(q)}$ for $q \geq s(\alpha^h n) - 1$. (Indeed, A and B are associated with the path problems of the same kind, having only different sizes, n and $s(\alpha^h n)$, respectively.) For the evaluation of B^* given B, we apply the cited earlier algorithms for the dense matrix case, using $s^2(4sk(s) - k(s) + 1)$ operations in the dioid or $O(k(s) \log s)$ steps, $\lceil 2s^3/\log s \rceil$ processors where $k(s) = \lceil \log_2 s \rceil - 1$, $s = s(\alpha^h n)$. Thus we arrive at the favorable complexity bounds of $O(\log n \log^2 s(n))$ parallel steps and $|E| + \lceil s^3(n)/\log s(n) \rceil$ processors for computing the recursive factorization (6), (7), and of $O(\log n \log s(n))$ parallel steps and $|E| + s^2(n)$ processors for computing bA^* for every b where the recursive factorization is available. Here $|E|$ denotes the number of edges of the graph associated with the matrix A, $|E| = O(n)$ for planar graphs. Therefore $O(\log n \log^2 s(n))$ steps suffice for both single source path problems (where b is fixed and only the row bA^*, but not the whole matrix A^*, must be computed; so $|E| + \lceil s^3(n)/\log s(n) \rceil$ processors suffice) and all pair path problem (where we compute A^* say by evaluating bA^* for all the n coordinate vectors b, so we use $n(|E| + \lceil s^2(n))/\log s(n) \rceil)$ processors). Multiplying the bounds for the numbers of steps and processors together, we obtain sequential time bounds, which can be slightly reduced further, to $|E| + s^3(n)$ for the single source paths and to $(|E| + s^2(n))n$ for the all pair paths, if we extend the sequential nested dissection algorithm of [10] (rather than the parallel algorithm of [13]) to path algebra computations. If $s(n) = O(\sqrt{n})$ and $|E| = O(n)$, as is the case for planar graphs, we arrive at the estimates shown in the table in our summary. In particular this includes the shortest path computations with some further applications (see the introduction).

5. Improvement of parallel evaluation of a minimum cut and a maximum flow in an undirected planar network

The best sequential algorithms for computing a minimum cut and a maximum flow in an undirected planar network $N = (G, c)$, ($G = (V, E)$ a graph, c a set of the edge capacities), run in $O(n \log n)$ time and exploit the reduction to the shortest path computations; see [3], Hassin and Johnson [5], Reif [16]. Specifically, [16] presented $O(n \log^2 n)$ time algorithm for computing a mincut, [5] extended that algorithm to computing a maximum flow and [3] improved the time bound to $O(n \log n)$. The previous best parallel polylog time algorithms [6] for those problems (via a rather straightforward parallelization of the sequential scheme) require an order of n^4 processors. Combining our results for the s.s.s.p.p. in planar graphs with the results of [7] we arrive at the bounds of $O(\log^4 n)$ steps and $n^{1.5}/\log n$ processors or alternatively $O(\log^3 n)$ steps and $n^2/\log n$ processors.

[6] reduces computing a mincut and the value $v_{\max}$ of a maxflow to the following stages (see [5,6,7,16] for further details):

(1) compute a plane embedding Π of N, for the estimated cost of $O(\log^2 n)$ steps, n^4 processors,

(2) find the plane dual network $D(N)$; step, processor bounds are $O(\log n)$, n^3,

(3) compute the μ-path, that is, the shortest path in the dual between the two faces F_s and F_t that adjoin the source s and the sink t of the primal network; step, processor bounds are $O(\log^2 n)$, n^3,

(4) compute the consistence clockwise orderings for the faces on the μ path; step, processor count is $O(\log n)$, n^2,

(5) compute the F-minimum cut-cycles in $D(N)$ for every dual vertex F on the μ-ath (that is, compute the cut-cycles of the minimum length in $D(N)$ passing through the vertex F); the latter stage can be reduced to solving the a.p.s.p.p. in the dual network $D(N)$; the cost is $O(\log^2 n)$ steps, n^3 processors,

(6) finally compute the minimum value of the F-minimum cut-cycles over all the dual vertices F on the μ-path; this gives $v_{\max}$ and a mincut; the cost is $O(\log n)$ steps, n processors.

The recent algorithm of [7] performs the computation at the substage (1) using $O(\log^3 n)$ steps, n processors; the computation also includes substages (2) and (4) performed using $O(\log n)$ steps, n processors. Applying our parallel algorithm for the s.s.s.p.p. at stage (3) and our parallel algorithm for the a.p.s.p.p. at stage (5), we perform the computations at those stages in $O(\log^3 n)$ steps using $n^{1.5}/\log n$ processors at stage (3) and $n^2/\log n$ at stage (5). Summarizing we need $O(\log^3 n)$ steps, $n^2/\log n$ processors for computing $v_{\max}$ and a mincut. Alternatively we may use $O(\log^4 n)$

steps and $O(n^{1.5}/\log n)$ processors at stage (5), which dominates the total complexity. To arrive at those bounds, we apply the algorithm of [16], which performs stage (5) by successively solving s.s.s.p.p.'s in the dual networks derived from $D(N)$. This is performed in at most $\lceil \log_2 n \rceil$ substages; on substage r up to 2^r s.s.s.p.p.'s are solved in the derived networks having the total number of edges at most $2|E|+2^r$, $r=0, 1, \ldots$ Here E is the edge set of the original planar network, $|E|=O(n)$, $2^r \le 2^{1+\log n}=2n$, so the total number of edges on each substance is $O(n)$. Therefore our algorithm for the s.s.s.p.p. enables us to perform each substage using $O(\log^3 n)$ steps, $n^{1.5}/\log n$ processors, so $O(\log^4 n)$ steps, $n^{1.5}/\log n$ processors suffice in all substages of stage (5) and consequently suffice for the entire computation of $v_{\max}$ and of a mincut.

When a mincut (passing through a vertex F on the μ-path) and the value $v_{\max}$ are known, we may immediately reduce computing a maxflow to an s.s.s.p.p. in the dual network, following [5]. (Specifically, this is the s.s.s.p.p. of computing the shortest distances between F and all other vertices in the dual network N.) Thus at that final stage we only need $O(\log^3 n)$ steps, $n^{1.5}/\log n$ processors.

Acknowledgement

The authors thank Sally Goodall for typing this paper.

References

[1] R. Bellman, "On a routing problem", *Quart. Appl. Math.* **16**, 87–90 (1958).

[2] R.N. Floyd, "Algorithm 97, shortest path", *Comm. ACM* **5**, 345 (1962).

[3] G.N. Fredericson, "Fast algorithms for shortest paths in planar graphs, with applications", CSD TR 486, Department of Computer Science, Purdue University, West lafayette, IN, 1984.

[4] M. Gondran and M. Minoux, *Graphs and Algorithms*, Wiley-Interscience, New York, 1984.

[5] R. Hassin and D.B. Johnson, "An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks", *SIAM J. on Computing*, forthcoming.

[6] D.B. Johnson and S.W. Venkatesan, "Parallel algorithms for minimum cuts and maximum flows in planar networks", *Proc. 23-rd Ann. IEEE Symp. FOCS*, 244–254 (1982).

[7] P. Klein and J. Reif, "An efficient parallel algorithm for planarity", Technical Report, Center for Research in Computer Technology, Aiken Computation Laboratory, Harvard University, Cambridge, MA and *Proc. 27-th Ann. IEEE Symp. FOCS*, Toronto, Oct. 1986.

[8] E.L. Lawler, *Combinatiorial Optimization: Networks and Matroids*, Holt, Rinehard and Winston, New York, 1976.

[9] R.J. Lipton and R.E. Tarjan, "A separator theorem for planar graphs", *SIAM J. Applied Math.* **36** (2), 177–189 (1979).

[10] R.J. Lipton, D. Rose and R.E. Tarjan, "Generalized nested dissection", *SIAM J. Numer. Analysis* **16** (2), 346–358 (1979).

[11] G. Miller, "Finding simple cycle separator for 2-connected planar graph", *J. Computer & System Science*, forthcoming.

[12] V. Pan, "How to multiply matrices faster", *Lecture Notes in Computer Science*, Vol. 179, Springer, Berlin, 1984.

[13] V. Pan and J. Reif, "Fast and efficient solution of linear systems", Technical Report TR-02-85, Center for Research in Computer Technology, Aiken Computation Laboratory, Harvard University, Cambridge, MA, 1985. Extended abstract in *Proc. 17-th Ann. ACM STOC*, Providence, RI, 143–152.

[14] V. Pan and J. Reif, "Extension of the parallel nested dissection algorithms to path algebra computations", Technical Report TR 85-9, Computer Science Department, SUNYA, Albany, NY, 1985.

[15] V. Pan and J. Reif, "Efficient sparse linear programming", Technical Report TR-11-85, Center for Research in Computer Technology, Aiken Computation Laboratory, Harvard University, Cambridge, MA, forthcoming in this Journal.

[16] J. Reif, "Minimum $s-t$ cut of a planar undirected network in $O(n \log^2(n))$ time, *SIAM J. Comput.* **12** (1), 71–81 (1983).

[17] R.E. Tarjan, "A unified approach to path problems", *J. ACM* **28** (3), 577–593 (1981).

[18] R.E. Tarjan, "Fast algorithms for solving path problems", *J. ACM* **28** (3), 594–614 (1981).