Mathematical Programming 33 (1985) 187-203
North-Holland

EFFICIENT DUAL SIMPLEX ALGORITHMS FOR THE ASSIGNMENT PROBLEM

D. GOLDFARB

Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027, USA

Received 31 May 1984
Revised manuscript received 2 January 1985

Efficient algorithms based upon Balinski's signature method are described for solving the $n \times n$ assignment problem. These algorithms are special variants of the dual simplex method and are shown to have computational bounds of $O(n^3)$. Variants for solving sparse assignment problems with m arcs that require $O(m)$ space and $O(mn + n^2 \log n)$ time in the worst case are also presented.

Key words: Assignment Problem, Dual Simplex Method, Weighted Matching, Optimization, Sparsity, Block Pivot

1. Introduction

The assignment problem is a fundamental problem in linear programming and network flow theory. Because of its importance, it has been extensively studied and numerous specialized algorithms have been developed to solve it. The most efficient of these algorithms have computational bounds of $O(n^3)$ for the $n \times n$ problem in the dense case. These include efficient versions of Kuhn's Hungarian method [16] (see [17, p. 205] for example), a hybrid algorithm due to Bertsekas [5] which combines the Hungarian method with a related method, a relaxation method due to Hung and Rom [15] which is very closely related to a method of Edmonds and Karp [9], and an efficient version of Balinski and Gomory's [3] primal algorithm due to Cunningham and Marsh [7]. All of the above $O(n^3)$ algorithms are 'dual' in nature except the last, and none are network simplex algorithms.

Recently three polynomially bounded network simplex algorithms for the assignment problem have been developed. Two of these, due to Roohy-Laleh [18] and Hung [14] are primal simplex algorithms based upon Cunningham's [6] 'strongly feasible' trees (see also Barr et al. [4], who call such trees 'alternating path bases'). Their bounds are respectively, $O(n^5)$ and $O(n^5 \ln \Delta)$, where Δ is the difference between the initial and final values of the objective. The third method, due to Balinski [2], is an $O(n^4)$ dual simplex algorithm based upon the 'signature' of a dual feasible basis. In this paper we show how to implement it and a variance in $O(n^3)$ operations.

This research was supported in part by the National Science Foundation under Grant No. MCS-8006064 and by the Army Research Office under Contracts No. DAAG 29-82-K0163 and DAAG 29-83-K0106

The $O(n^3)$ bound for our variants of Balinski's method is as good as the best bounds known for any other algorithm for the assignment problem. This is significant because no such statement can be made at the present time about the worst-case computational complexity of any variant of the simplex method for any other class of linear programming problems. Balinski's method is a dual simplex method since it proceeds from dual feasible basis (vertex) to adjacent dual feasible basis. We note that it does not, however, choose pivots in the usual way.

In the next section, we briefly describe Balinski's 'signature' method and introduce the terminology and notation needed subsequently. Our new algorithms are presented in Section 3. Detailed computational analyses are given in Section 4. In Section 5, we discuss how to modify our algorithms when some assignments are not admissible. In Section 6 we show how to implement these algorithms so that they require only $O(m)$ storage and at most $O(mn+n^2 \log n)$ computational time on problems with m admissible assignments. Finally, in Section 7, various matters relevant to the practical implementation of our algorithms and areas for future investigations are discussed.

2. The basic signature method

Consider a complete bipartite graph $G=(R, C, E)$ with $E=R\times C$ and $|R|=|C|=n$. By an assignment or matching, we mean a subset X of arcs $(i,j)\in E$ such that there is exactly one arc in X for each row node $i\in R$ and each column node $j\in C$. Given a weight a_{ij} for each arc $(i,j)\in E$, the $n\times n$ assignment problem is that of finding an assignment amongst the $n!$ possibilities X that minimizes $\sum_{(i,j)\in X} a_{ij}$.

It is well known that the $n\times n$ assignment problem can be formulated as the linear programming problem:

$$\text{minimize} \sum_{i,j} a_{ij}x_{ij} \tag{1a}$$

$$\text{subject to} \sum_{j} x_{ij}=1 \quad \text{for all } i=1,\ldots,n, \tag{1b}$$

$$\sum_{i} x_{ij}=1 \quad \text{for all } j=1,\ldots,n, \tag{1c}$$

$$x_{ij}\geq 0 \quad \text{for all } i,j=1,\ldots,n, \tag{1d}$$

where at optimality $x_{ij}=1$ if and only if $(i,j)\in X$. The corresponding dual linear program is:

$$\text{maximize} \quad \sum_{i} u_i+\sum_{j} v_j \tag{2a}$$

$$\text{subject to} \quad u_i+v_j\leq a_{ij} \quad \text{for all } i,j=1,\ldots,n. \tag{2b}$$

Consequently, the $n\times n$ assignment problem can be solved by applying some version of the simplex method to either of these linear programs. Moreover, as the assignment

problem is a special case of the minimum cost network flow problem, so-called network simplex algorithms can be used to obtain an optimal solution. (See Dantzig [8] for the first such algorithm.)

We now recall that a set of columns of the constraint matrix corresponding to (1b)-(1c) indexed by $T \subseteq E$ is a column basis for this matrix if and only if T is a spanning tree of G. Given T, a unique solution to the primal constraints (1b)-(1c) is obtained by setting $x_{ij} = 0$ for all $(i,j) \notin T$. If, in addition, we arbitrarily fix the value of one of the dual variables (e.g. $u_1 = 0$) then a dual solution is easily computed from the equations

$$u_i + v_j = a_{ij} \quad \text{for all } (i,j) \in T.$$

Of course, for an arbitrary T, x (u and v) will not, in general, be primal (dual) feasible; if x is (u and v are), then T will be called primal (dual) feasible. Henceforth, we shall use the notation T to denote a tree, and $P(v, w)$ to denote the unique path in T between two nodes v and w in T. It should not cause confusion that T is used to denote the set of arcs of a tree at certain times and the set of nodes at others. We shall also consider trees to be rooted; i.e. there is a designated node $r \in T$ called the root of T and the remaining nodes in T excluding r are partitioned into $k > 0$ disjoint subsets called subtrees of r, each of which is itself a tree.

In [2] Balinski defines the 'signature' of a tree T as the n-vector $d = (d_1, \ldots, d_n)$ where d_i is the degree in T of row node $i \in R$. Clearly the column, as opposed to row, signature could also be defined. An important property of the signature of T is that if it has only one component equal to 1 then the x corresponding to T is a feasible assignment. To see that this is indeed the case, observe that all other components must equal 2 since $\sum_i d_i = 2n - 1$ and $d_i \geq 1$ for all i (T is a spanning tree of $2n$ nodes and $2n - 1$ arcs). Let $r \in R$ be the row node in T with $d_r = 1$ and let

$$X = \{(i,j) \mid (i,j) = \text{arc on path } P(r,j) \text{ adjacent to } j, \text{ for all } j \in C\}.$$

Clearly each column node $j \in C$ is covered by X and since for all $i \in R$, $d_i = 2$ except for $i = r$ ($d_r = 1$) it is evident that each row node is covered by exactly one arc. See Fig. 1.

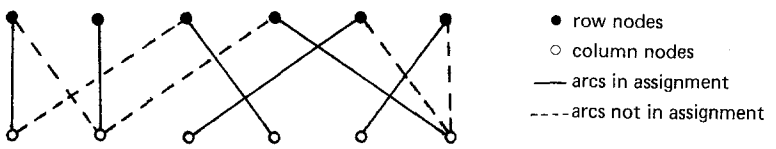


Fig. 1. An assignment corresponding to a basis tree with exactly one terminal row node.

If a basis tree T has exactly one 'terminal' row node (i.e. a row node with degree one) and is also dual feasible then the assignment X that corresponds to T is optimal. Hence, given a dual feasible basis tree T, an algorithm which reduces the number of terminal row nodes while maintaining dual feasibility can be used recursively to produce an optimal assignment. This is the basic step of a signature

method. As we shall now show, it can be achieved by a finite sequence of simplex pivots. These pivots are performed without explicit reference to the values of the primal variables; hence, the change in the primal (dual) objective function may be positive, negative or zero, in contrast with standard variants of the simplex method.

2. The basic signature step

Let $\hat{T}$ be a given dual feasible tree with more than one terminal row node. Choose one of these nodes, say t, as the 'target', and any row node s of degree $d_s \geq 3$ (there must be at least one such node) as the 'source'. Consider performing a simplex pivot which removes from $\hat{T}$ the arc (s, ℓ) on the path $P(s, t)$, and replaces it so as to maintain dual feasibility. Designate s as the root of $\hat{T}$, and let T^ℓ be the subtree of $\hat{T}$ with root ℓ and $T^s = \hat{T} \backslash (T^\ell \cup \{(s, \ell)\})$—i.e., removing (s, ℓ) from $\hat{T}$ divides it into two components, T^ℓ and T^s. Let $R^\ell(C^\ell)$ and $R^s(C^s)$ be the sets of row (column) nodes in T^ℓ and T^s, respectively. It is easily verified that, as the dual nonbasic slack $\pi_{s\ell} = a_{s\ell} - u_s - v_\ell$ is increased from zero to ε,

$$\pi_{ij} \begin{cases} \text{increases by } \varepsilon, & \text{for } i \in R^s, j \in C^\ell, \\ \text{decreases by } \varepsilon, & \text{for } i \in R^\ell, j \in C^s, \\ \text{remains the same} & \text{otherwise.} \end{cases}$$

Consequently, if a dual simplex pivot is performed in which arc (s, ℓ) is deleted from $\hat{T}$ and is replaced by arc (g, h), then $g \in \mathbb{R}^\ell$ and $h \in C^s$, and

$$\pi_{gh} = \min\{\pi_{ij} \mid i \in R^\ell, j \in C^s\} = \varepsilon \geq 0. \tag{3}$$

If g was a terminal node in $\hat{T}$, then our goal of reducing the number of terminal nodes is achieved, and the basic step is complete. If this is not the case, we take g as the source in place of s, and repeat the above procedure with $\hat{T}$ replaced by T^ℓ. Since $|R^\ell|$, the number of rows in T^ℓ, decreases by at least one (node g is removed) on each repetition of the above simplex pivot, only a finite number of pivots is required to complete the basic step. In fact, it is evident that, if $\hat{T}$ has k terminal row nodes, then the number of simplex pivots required is, at most, $n-k$.

In Balinski's signature method [2], an initial dual feasible basis tree is constructed with signature $(n, 1, 1, \ldots, 1)$. On each basic step, the degree of node 1 (the source) is reduced by one, and some terminal row node's degree is increased by one. Clearly, $n-2$ basic steps are needed to reach optimality, with the kth requiring at most k simplex pivots. Thus, as shown in [2], a total of at most of $\sum_1^{n-2} k = (n-1)(n-2)/2$ pivots is needed. The $O(n^4)$ computational bound given by Balinski arises from the fact that straightforward computation of (3) to determine the incoming arc on each simplex pivot is $O(n^2)$. In the next section, we show how each basic signature step can be accomplished using only $O(n^2)$ operations. Since Balinski's method requires only $n-2$ basic steps, this results in an $O(n^3)$ dual simplex algorithm.

Before presenting this algorithm, it should be stressed that the basic signature step can be carried out as long as the row signature contains more than one '1'—i.e., terminal row node. In Balinski's method, at the start of a basic step, all row nodes except the source have degree one or two. This ensures that the primal variable $x_{o\ell}$ that leaves the basis on a simplex pivot is less than or equal to zero. (See [2] for proof of this.) In the general case, the leaving primal basic variable may also be positive; hence, the dual objective function (2a) may increase, decrease, or remain unchanged on a dual simplex pivot step.

3. $O(n^3)$ signature algorithms

The key to the efficiency of the algorithms given below is the use of labels γ_i on all row nodes $i \in R^\ell$. These enable the algorithms to use information already computed on previous simplex pivots during the current basic signature step.

To be specific, the computation

$$\pi_{gh} = \min\{\pi_{ij} \mid i \in R^\ell, j \in C^o\} = \varepsilon \tag{3}$$

that is required to determine the arc (g, h) to enter the basis on a simplex pivot can be performed as

$$\varepsilon = \min_{i \in R^\ell} \{\gamma_i\} = \gamma_g \tag{4a}$$

where

$$\gamma_i = \min_{j \in C^o} \{\pi_{ij}\}, \quad i \in R^\ell. \tag{4b}$$

The column index which gives the minimum in (4b) for each $i \in R^\ell$ is also needed. It is easily verified that the dual variables before and after (indicated by an overbar) a simplex pivot are related as follows:

$$\bar{u}_i = u_i + \varepsilon, \quad i \in R^\ell, \qquad \bar{u}_i = u_i, \quad \text{otherwise.}$$

$$\bar{v}_j = v_j - \varepsilon, \quad j \in C^\ell, \qquad \bar{v}_j = v_j, \quad \text{otherwise.}$$

Furthermore, since $\bar{R}^\ell \subset R^\ell$ and $\bar{C}^o \supset C^o$, it follows that

$$\bar{\gamma}_i = \min\{\gamma_i - \varepsilon, \min_{j \in \bar{C}^o \setminus C^o} \{\pi_{ij}\}\} i \in \bar{R}^\ell.$$

Consequently, no matter how many simplex pivots are carried out during a basic signature step, each π_{ij} needs to be computed and compared only once. Since the rest of the work required by a simplex pivot is $O(n)$ and there are at most $n-2$ pivots per basic signature step, the step can be accomplished using only $O(n^2)$ operations.[1] An $O(n^3)$ signature algorithm which combines the above observations with Balinski's method [2] follows:

[1] The use of the row node labels γ_i to reduce the computational requirements of Balinski's method has independently been suggested by W. Cunningham [Balinski; private communication].

Signature Algorithm 1

Input: $n \times n$ matrix a_{ij}

Output: An optimal matching (assignment) for the bipartite graph $G(R, C, E)$; $E = R \times C$, $|R| = |C| = n$, $|E| = n^2$.

Step (0) *Initialization*: *Compute initial dual feasible solution* (u, v), *and basis tree* T. *Set* $T \leftarrow \emptyset$ *and* $Q \leftarrow R \backslash \{1\}$.

(i) Set $u_1 \leftarrow 0$ and $p \leftarrow 1$.

(ii) For every $j \in C$, set $v_j \leftarrow a_{1j}$ and $T \leftarrow T \cup \{(1, j)\}$

(iii) For every $i \in Q$, compute $\min_{j \in C} \{a_{ij} - v_j\} \equiv a_{ik} - v_k$ and set $u_i \leftarrow a_{ik} - v_k$ and $T \leftarrow T \cup \{(i, k)\}$.

Step (1) *Reduce the number of terminal row nodes,* Q, *by one.* If $|Q| = 1$ (or the assignment X is feasible)[2] STOP; otherwise, choose a target node $t \in Q$. Set $\sigma \leftarrow p$ and $\hat{T} \leftarrow T$.

Step (2) *Dual Simplex Step*

(a) *Choose edge to leave basis tree.* Choose σ as the root of $\hat{T}$. Let (σ, ℓ) be the edge on the unique path $P(\sigma, t)$ in $\hat{T}$ from σ to t that is adjacent to σ. Let T^ℓ be the subtree of $\hat{T}$ with root ℓ, and $T^\sigma = \hat{T} \backslash (T^\ell \cup \{(\sigma, \ell)\})$; i.e., removing arc (σ, ℓ) from $\hat{T}$ creates two components, T^ℓ and T^σ. Let $R^\ell(C^\ell)$ and $R^\sigma(C^\sigma)$ be the row (column) nodes in the current T^ℓ and T^σ, respectively. If $\sigma = p$, set $\gamma_i \leftarrow \infty$ for all $i \in R^\ell$.

(b) *Choose edge to enter basis tree.* For every $i \in R^\ell$ and $j \in C^\sigma$, $((i, j) \in E)$ compute $\pi_{ij} = a_{ij} - u_i - v_j$.[3] If $\pi_{ij} < \gamma_i$, set $\gamma_i \leftarrow \pi_{ij}$ and $\mathrm{col}(i) \leftarrow j$. Compute $\varepsilon = \min_{i \in R^\ell} \{\gamma_i\} \equiv \gamma_g$ and let $h \leftarrow \mathrm{col}(g)$.

(c) *Update Dual (and Primal) Variables and Basis Tree.*

For every $i \in \mathcal{R}^\ell$, set $u_i \leftarrow u_i + \varepsilon$ and $\gamma_i \leftarrow \gamma_i - \varepsilon$.

For every $j \in C^\ell$, set $v_j \leftarrow v_j - \varepsilon$.

Set $T \leftarrow (T \backslash \{(\sigma, \ell)\}) \cup \{(g, h)\}$.

(Update primal variables; i.e., assignment X.)

If $g \notin Q$, set $\sigma \leftarrow g$, $\hat{T} \leftarrow T^\ell$ and go to Step (2). Otherwise, set $Q \leftarrow Q \backslash \{g\}$ and go to Step (1).

We now give a signature algorithm whose worst-case computational bound is slightly better than the bound for the algorithm above. The improvement is obtained by solving a sequence of assignment problems, each larger than the preceding one by a pair of nodes, starting from a 1×1 problem.

Given an optimal solution to a $(k \times k)$ assignment problem and basis tree T having exactly one terminal row node, T is first extended to the $(k+1) \times (k+1)$ problem maintaining dual feasibility. Then, if necessary, one basic signature step is executed to reduce the number of terminal row nodes from two to one. Because this step requires at most $O(k+1)^2)$ rather than $O(n^2)$ operations, there is a reduction in the coefficient of n^3 in the worst-case computational bound. This algorithm follows.

[2] As Balinski [2] points out, it is not necessary to compute primal variables during the course of a signature method.

[3] Here C^σ is what was previously denoted by $\bar{C}^\sigma \backslash C^\sigma$.

Signature Algorithm 2

Input and Output: *Same as Algorithm 1*

Step (0) Compute an optimal solution to the (1×1) assignment problem.
Set $u_1 \leftarrow 0$, $v_1 \leftarrow a_{11}$, $T \leftarrow \{(1,1)\}$ and $Q \leftarrow \{1\}$.

Step (1) Do for $k = 1, \ldots, n-1$,
Call procedure: EXTEND(k)

Procedure: EXTEND(k)

Input: Optimal solution (u, v, x) to a $k \times k$ assignment problem, and basis tree T having exactly one terminal row node, i.e., $|Q| = 1$.

Output: Optimal solution to $(k+1)\times(k+1)$ assignment problem obtained by adding $(k+1)$st pair of nodes and their associated edges.

Step (0): *Extend $k \times k$ dual feasible solution and tree to $(k+1)\times(k+1)$ problem:*

(i) Compute $\min_{1\leqslant j\leqslant k}\{a_{k+1,j} - v_j\} \equiv a_{k+1,q} - v_q$
and set $u_{k+1} \leftarrow a_{k+1,q} - v_q$, $T \leftarrow T \cup \{(k+1, q)\}$,
and $Q \leftarrow Q \cup \{k+1\}$.

(ii) Compute $\min_{1\leqslant i\leqslant k+1}\{a_{i,k+1} - u_i\} \equiv a_{p,k+1} - u_p$
and set $v_{k+1} \leftarrow a_{p,k+1} - u_p$ and $T \leftarrow T \cup \{(p, k+1)\}$
If $p \in Q$, set $Q \leftarrow Q \backslash \{p\}$; otherwise execute Step (1) of Algorithm 1.

4. Detailed computational analysis

The upper bound of $\frac{1}{2}(n-1)(n-2)$ on the number of simplex pivots required by algorithm 1 is shown in [2] to be attained for the particular matrix $a_{ij} = (n-i)(j-1)$. In this section, we shall derive strict upper bounds on the computational requirements (i.e., arithmetic operations and comparisons) for algorithms 1 and 2 and exhibit matrices for which these bounds are achieved.

Let us consider a basic signature step starting from a basis tree with k terminal row nodes. The computational cost of a basic step comes for the most part from steps (2b) and (2c) on each simplex pivot. We shall first analyze the maximum amount of work required by step (2b) for choosing the edge to enter the basis. Clearly, this work is maximized when $|C^s|$ and $|R^\ell|$ are as large as possible on each pivot and the maximum number of pivots is performed. Since $|C^\ell| \geqslant 1$ and $|R^s| \geqslant 1$ ($s \in R^s$), and at least one node must be added to C^s and deleted from R^ℓ on each pivot it follows that the number of dual slacks π_{ij} (reduced costs) that must be computed during the at most $n-k$ pivots required by a basic step starting with k terminal nodes is

$$k(n-1) + \sum_{i=1}^{n-k-1} (n-1-i) = (n-1)^2 - \tfrac{1}{2}(n-k)(n-k-1).$$

Consequently, the total computational cost of computing the π_{ij} in step 2b for an

$n \times n$ assignment problem is at most

$$\sum_{k=2}^{n-1} [(n-1)^2 - \tfrac{1}{2}(n-k)(n-k-1)] = \tfrac{1}{6}(n-2)(n-1)(5n-3)$$

$$= \frac{5n^3 - 18n^2 + 19n - 6}{6}$$

where here we are considering 2 additions and 1 comparison as one unit of cost.

The work required by step (2c) of algorithm 1 is roughly proportional to the number of nodes in T^ℓ. This is because only the dual variables corresponding to nodes in T^ℓ are updated and the work required to update the list structures used to represent the basis tree T, is proportional to $|T^\ell|$. On the other hand there are only $|R^\ell|$ labels γ_i updated in step (2c).

As in the analysis above, the amount of work required is greatest if $|T^\ell|$ (i.e. $|R^\ell|$) is chosen as large as possible at the start of a basic step and is decreased as slowly as possible. In particular the worst case occurs when $|R^\ell| = n-1$ initially. In this case if the number of terminal nodes in T is k, then $|C^\ell| = n-k$ and $|T^\ell| = 2n-k-1$ since the non-terminal nodes in T^ℓ are all of degree 2. Since there are at most $n-k$ pivots and each reduces the number of nodes in T^ℓ by at least 2 an upper bound on the number of dual variable updates required by a basic step starting with k terminal nodes is

$$\sum_{i=0}^{n-k-1} (2n-k-1-2i) = n(n-k).$$

Consequently the total work required by step (2c) to solve an $n \times n$ assignment problem is approximately proportional to

$$\sum_{k=2}^{n-1} n(n-k) = \tfrac{1}{2}n(n-1)(n-2) = \tfrac{1}{2}(n^3 - 3n^2 + 2n)$$

in the worst case.

Let us now consider procedure EXTEND(k). In the worst case $k-1$ pivots are required by the signature step to reduce the number of terminal row nodes from two to one. As each pivot in this case results in $|C^a|$ increasing by one and $|R^\ell|$ decreasing by one starting from $|C^a| = 2$ and $|R^\ell| = k$, we obtain

$$2k + \sum_{i=2}^{k-1} i = \frac{k^2 + 3k - 2}{2}$$

as an upper bound on the number of π_{ij} that need to be computed and compared by EXTEND(k). Since procedure EXTEND(k) is called for $k = 1, \ldots, n-1$ we obtain

$$\sum_{k=1}^{n-1} \tfrac{1}{2}(k^2 + 3k - 2) = \tfrac{1}{6}n(n+5)(n-2) + 1 = \tfrac{1}{6}(n^3 + 2n^2 - 10n + 6)$$

as an upper bound on the total computations and comparisons of π_{ij} that are needed to solve an $n \times n$ assignment problem.

Further the number of dual variable updates required by EXTEND(k) is at most

$$\sum_{j=1}^{k-1} (2j+1) = k^2 - 1.$$

Therefore the total work required by step (2c) of the basic signature step in algorithm 2 is approximately proportional to

$$\sum_{k=1}^{n-1} (k^2-1) = \tfrac{1}{6}(2n+3)(n-1)(n-2) = \tfrac{1}{3}n^3 - \tfrac{1}{2}n^2 - \tfrac{5}{6}n + 1$$

in the worst case.

The only other major computational demands in our algorithms occur in the computation of ε in step (2b). One can verify that in algorithms 1 and 2, respectively, a total of $\frac{1}{6}(2n-3)(n-1)(n-2)$ and $\frac{1}{6}n(n-1)(n-2)$ comparisons are needed in the worst case. As we shall see in Section 6, these upper bounds on the cost of computing ε and those involved in step (2c) can be reduced by using sophisticated data structures and block pivots, respectively.

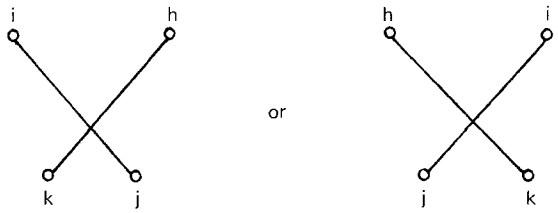
It is easy to show that the upper bounds for algorithm 1 are attained for the matrix $a_{ij} = (n-i)(j-1)$ (or equivalently $a_{ij} = -ij$) by expanding on the arguments given in [2]. Algorithm 2, however, solves this problem trivially using only $O(n^2)$ operations. On the other hand, both algorithms achieve their worst-case bounds for the matrix $a_{ij} = ij$ first used by R. Silver in [19] as a worst-case example for the Hungarian method. Dual feasible trees for this problem have the property that all nonadjacent edges 'cross'. If (i, j) and (h, k) are in T and T is dual feasible then

$$u_i + v_j = ij, \qquad u_h + v_k = hk, \qquad u_i + v_k \leqslant ik, \qquad u_h + v_j \leqslant hj.$$

Adding the inequalities and subtracting the equations yields

$$(h-i)(j-k) \geqslant 0$$

which implies that edges (i, j) and (h, k) cross if $i \neq h$ and $j \neq k$ i.e.



In contrast, dual feasible trees for the matrix $a_{ij} = (n-i)(j-1)$ have 'no crossings' as shown in [2].

5. Inadmissible arcs

The algorithms presented in Section 3 assume that all assignments are admissible. If this is not the case (i.e., G is not a complete bipartite graph) then in order for algorithms 1 and 2 to be used as given, inadmissible arcs must be replaced by artificial arcs with very large weights. In this section we describe how to modify algorithms 1 and 2 to take advantage of any sparseness in G directly. Let us assume that G is connected; otherwise, the assignment problem decomposes into a collection of smaller problems and our task becomes easier. Further, we assume that G has a complete matching (i.e., one with cardinality equal to n).

Our first appoach is based upon modifying algorithm 1. In the initialization step of that algoirthm an initial basis tree, T, is grown out of row node 1 with all column nodes at level one and all row nodes other than node 1 at level two. Here we are using level to denote the distance of a node in T from the root (i.e., row node 1). When there are inadmissible arcs (i.e., sparsity), it may be necessary to construct an initial basis tree that has more than two levels in addition to the root. Starting from a particular row node one simply grows T one level at a time where the nodes added to level $k+1$ of T are just those nodes, connected by an arc in E to nodes in level k of T, that have not yet been added to T. Specifically, the initialization step for the modified algorithm 1 becomes:

Initialization: *Compute initial dual feasible solution* (u, v) *and basis tree* T.

(i) Set $T \to \emptyset$, $\bar{C} \leftarrow C$, $\hat{R} \leftarrow \{1\}$, $\bar{R} \leftarrow R \backslash \hat{R}$, $Q \leftarrow R$, $P \leftarrow \emptyset$ and $u_1 \leftarrow 0$.

(ii) If $\bar{C} = \emptyset$, go to step (1);
else set $\hat{C} \leftarrow \{j \in \bar{C} \mid (i,j) \in E \text{ for some } i \in \hat{R}\}$, and
for every $j \in \hat{C}$, set $v_j \leftarrow \min_{i \in \hat{R}} \{a_{ij} - u_i\} = a_{kj} - u_k$ and $T \leftarrow T \cup \{(k,j)\}$.
If $k \in Q$, set $Q \leftarrow Q \backslash \{k\}$; else if $k \notin P$, set $P \leftarrow P \cup \{k\}$

(iii) If $\bar{R} = \emptyset$, go to step (1);
else set $\hat{R} = \{i \in \bar{R} \mid (i,j) \in E \text{ for some } j \in \hat{C}\}$, and
for every $i \in \hat{R}$, set $u_i \leftarrow \min_{j \in \hat{C}} \{a_{ij} - v_j\} = a_{ik} - v_k$ and $T \leftarrow T \cup \{(i,k)\}$.
Set $\bar{C} \leftarrow \bar{C} \backslash \hat{C}$ and $\bar{R} \leftarrow \bar{R} \backslash \hat{R}$ and go to (ii).

In the above procedure $\bar{R}$ and $\bar{C}$ are the sets of row and column nodes, respectively, that have not yet been added to T. In step (ii) the next level of column nodes $\hat{C} \subseteq \bar{C}$ to be added to T are determined. These are just those nodes in $\bar{C}$ that are connected by an arc to the last level of row nodes, $\hat{R}$, in T. Moreover, the choice of v_j in step (ii) ensures that $\pi_{ij} \geq 0$ for all $i \in \hat{R}$ and $j \in \hat{C}$ and $\pi_{ij} = 0$ for the arc added to T. The above statements about step (ii) apply to step (iii) if the roles of column and row, C and R, and v and u are interchanged. Consequently, the dual solution produced by this procedure is both basic and feasible. Note that when this initialization step is finished, P and Q are, respectively, the sets of candidate source nodes (row nodes with degree ≥ 3) and target nodes (terminal row nodes).

Step (1) of algorithm 1 remains the same except that prior to setting $s \leftarrow p$ we must insert the instruction 'choose a source node $p \in P$'.

Besides the initialization step the other major change to algorithm 1 that is required involves step (2b). Because of sparsity there may be no admissible arcs from which to choose an entering arc. (i.e., $\varepsilon = \infty$). If this happens we can introduce an artificial arc from the target node t to any column node h in $T \setminus T^{\ell}$, assign it a prohibitively large weight M, and set $g \leftarrow t$. By choosing $g = t$ we complete a basic signature step; consequently, the number of artificial arcs that may have to be introduced is less than the number of initial terminal nodes. Also, in contrast to replacing *all* inadmissible arcs by artificial arcs, there is no need to store a full $n \times n$ matrix of weights. See the discussion in the next section on data structures that require $O(m)$ space for the weights a_{ij} in problems with $m = |E|$ arcs.

In the above method we found it necessary to allow the introduction of artificial arcs when there were no admissible arcs from which to choose an arc to enter the basis tree. One might think that, when this happens, one need only choose an alternative source or target node, if any exist, and continue the algorithm. Unfortunately, it is not obvious how to do this and obtain an algorithm with a low order worst-case time bound.

We now show how algorithm 2 can be modified to give an efficient algorithm that does not require the introduction of artificial arcs. The only catch is that we must first find a complete matching in G. Since this preprocessing step can be done in $O(n^{1/2}m)$ time [12] it does not affect the worst case time bound for an efficient implementation (see the next section); on the other hand its cost is not negligible. After this step is completed we choose any matched pair of row and column nodes and compute the trivial optimal solution to this (1×1) assignment problem as in step (0) of algorithm 1. Before calling procedure EXTEND(k) in step (1) we must first find an arc (i, j) in G with $i \notin T$ and $j \in T$. (T is the basis tree for the current $k \times k$ assignment problem.) If such an arc is found, then i and its matched column node are treated as the $(k+1)$st pair of nodes and EXTEND(k) is executed with the change that minimizations in it are taken only over admissible arcs in E and the new row node is chosen as the target t.[4] If no such arc exists, then the full problem decomposes into two simpler problems: a $k \times k$ problem whose optimal solution we have already found an $(n-k) \times (n-k)$ problem which we can attempt to solve with algorithm 2.

6. Efficient algorithms for sparse assignment problems

If the algorithms described in the preceding section are to be efficient both in space and time, then care must be exercised in their implementation and in the

[4] Because there is an arc from row node i to its matched column node, the set of admissible arcs that can enter the basis during any simplex pivot in EXTEND(k) is never empty; in particular, throughout the signature step, $t = i$ remains in R^{ℓ} and its matched column node remains in C^{a}.

choice of data structures used. In this section we consider how to (i) store the problem data using only $O(m)$ locations while ensuring that the total time spent accessing this data is $O(mn)$, (ii) update the dual variables, row node labels and basis tree in $O(n)$ time per basic signature step even if such a step requires $O(n)$ simplex pivots, and (iii) choose the entering arc on a simplex pivot in less than $O(n)$ time.

(i) *Data Structures for Problem Data*: The standard way to store arc data for a sparse graph is with one adjacency list for each node. This approach requires $O(m)$ storage but is inefficient timewise because we may have to scan the entire list for node i to find the weight a_{ij} or determine that arc (i, j) is inadmissible. Clearly, we need a data structure with constant access time. The full $n \times n$ adjacency matrix of weights a_{ij} has constant access time for all n^2 possible arcs but requires $O(n^2)$ storage.

Recently, Fredman et al. [10] devised a clever method for storing n_i integers from the set $1, \ldots, n$ which uses $O(n_i)$ space and requires constant time for accesses. This scheme is based upon two levels of hashing; specifically a perfect hashing function is used at the second level to resolve collisions caused at the first level. Construction of this data structure can be done in expected time $O(n_i)$ and in $O(n_i n)$ time in the worst case. Consequently, if used for all adjacency lists (n_i is the cardinality of the list for node i), this structure can be constructed in at most $O(mn)$ time.

Unfortunately, constant access time is not enough to guarantee that no more than $O(m)$ amount of time is spent computing all dual nonbasic slacks π_{ij} required during a basic signature step. The problem with the above scheme, and the full adjacency matrix as well, is that we have no way of knowing whether a candidate arc is admissible or not other than trying to access it. Hence, since all arcs(i, j) with $i \in R^\ell$ and $j \in C^s$ are candidates in step (2b) of our algorithms, as many as $O(n^2)$, ($O(k^2)$ in versions of Algorithm 2) accesses may be required during one basic signature step even if there are only $O(n)$ admissible arcs.

Applying the sparse variant of algorithm 1 to a problem with $3n-2$ admissible arcs, with $a_{ii} = i$, $a_{1i} = a_{i1} = 1$, for all $i = 1, \ldots, n$ requires a total

$$\sum_{k=2}^{n-1} (n-1)k = \frac{(n^2-1)(n-2)}{2} = O(n^3)$$

accesses, even though only $n-2$ simplex pivots (one per basic signature step) are needed. The initialization step produces a basis tree T consisting of the arcs $(1, 1)$ and $(1, i)$ and $(i, 1)$ for all $i = 1, \ldots, n$. It is easily verified that on the kth iteration of the algorithm $R^\ell = \{2, 3, \ldots, n\}$ and $C^s = \{k+1, k+2, \ldots, n\}$ and arc $(1, k)$ in T is replaced by arc $(k+1, k+1)$. (See Fig. 2.)

For the sparse variant of algorithm 2 we can obtain similar results for an assignment problem containing $3n-2$ arcs with weights

$$a_{1,i} = 1, \ i = 1, \ldots, n, \quad a_{ii} = 2, \ i = 2, \ldots, n, \quad a_{i,i-2} = 1, \ i = 3, \ldots, n, \quad \text{and} \quad a_{21} = 1.$$

(See Fig. 3.) It is easy to show that on the call EXTEND(k), after arcs $(k+1, k-1)$

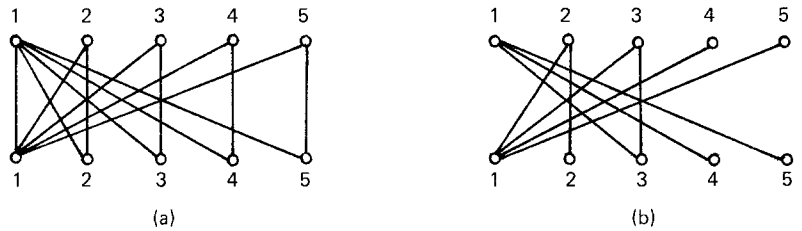


Fig. 2. (a) Graph for sparse 5×5 assignment problem with $a_{ij}=\min(i,j)$ for all admissible arcs. (b) Basis tree T after two iterations of the sparse variant of algorithm 1 on the problem in (a).

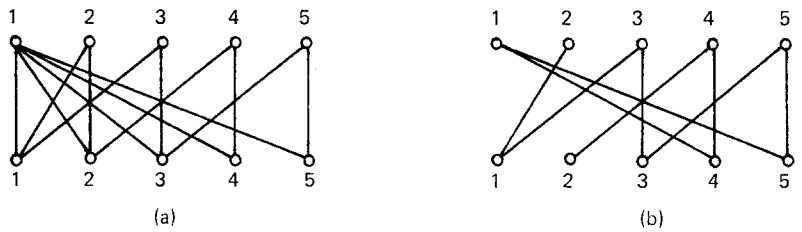


Fig. 3. (a) Graph for sparse 5×5 assignment problem with $a_{ii}=2$ for $i=2,\ldots,5$ and $a_{ij}=1$ for all other admissible arcs. (b) Basis tree T at the start of the kth call to procedure EXTEND(k) for the problem in (a).

and $(1, k+1)$ are added to T, a simplex pivot replaces arc $(1, k-1)$ by $(k+1, k+1)$ to complete a basic signature step. Moreover,

$$|R^{\ell}|\cdot|C^{s}|=\begin{cases}(p+1)^2 & \text{if } k \text{ is even and } k=2p,\\ (p+1)^2-1 & \text{if } k \text{ is odd and } k=2p+1.\end{cases}$$

Consequently, that step requires $O(k^2)$ accesses even though the $k\times k$ subproblem has only $O(k)$ arcs.

Clearly, we need a data structure that allows us to access in constant time all admissible arcs $(i, j)\in E$ with $i\in R^{\ell}$ and $j\in C^{s}$ in step (2b) and *only* those arcs. The following storage scheme meets these requirements. It is perhaps best viewed as a scheme for storing a sparse adjacency matrix A which allows rows to be deleted efficiently. The entries of A and their row and column indices are stored row-by-row in three m length arrays. Pointers to the start of each row are stored in a separate $(n+1)$ length array. (The $(n+1)$st element of this array has the value $m+1$ and is included as a convenient delimiter of the nth row.) Two additional m-length arrays of pointers are used to link together elements in the same column as a doubly-linked list. Finally, we need n pointers to the start of each of these column lists. These are appended to one of the other pointer arrays as shown in Fig. 4 below to make deletion of the first element in a column no different from any other element.

Because all elements in the same row are stored together and columns are stored as doubly-linked lists, all elements of rows that are removed from R^{ℓ} after a simplex pivot can also efficiently be removed from their column lists (i.e., in time proportional

$$A = \begin{bmatrix} 0 & a_{12} & a_{13} & 0 & 0 \\ a_{21} & 0 & 0 & a_{24} & 0 \\ a_{31} & 0 & a_{33} & 0 & a_{35} \\ 0 & a_{42} & 0 & 0 & 0 \\ a_{51} & a_{52} & 0 & a_{54} & 0 \end{bmatrix}$$

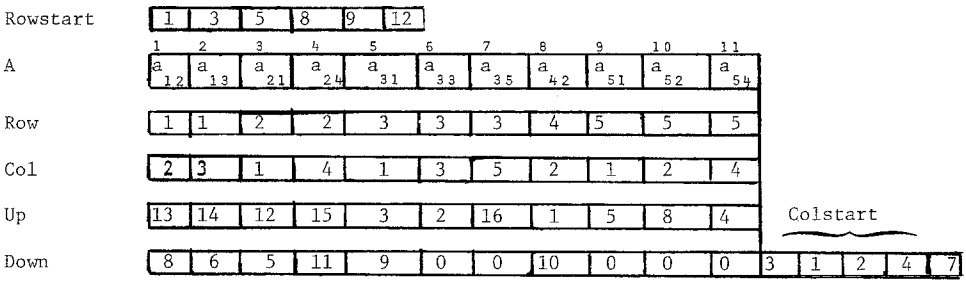


Fig. 4. A 5×5 sparse matrix and its representation by a file of packed rows with elements in the same column linked together as a doubly-linked list.

to the number of such elements.) It follows that in a basic signature step each nonzero element of A (i.e., each admissible arc) is accessed at most once, and no zero elements are accessed if the computation of π_{ij} in step (2b) is implemented by scanning those column lists currently indexed by the set C^a. The structure of each column initially contained in the pointer arrays UP and DOWN is at least partially destroyed during a signature step. However, because the structure of A is also maintained (rowwise) in the other arrays, it is easy to reconstruct these doubly-linked column lists in $O(m)$ time at the start of each basic signature step.

(ii) *Block Updates*: Since in the sparse case $O(n^2)$ simplex pivots may be required by the sparse variants of algorithms 1 and 2, and the computation of ε (i.e., the minimum γ_i) in step (2b) and the updating of the dual variables, the row node labels, and the basis tree in step (2c) all require $O(n)$ time per pivot, some further modification of these algorithms are necessary if we are to obtain a worst-case time bound that is better than $O(n^3)$.

We first observe that it is not necessary to update the dual variables and row node labels until the end of a basic signature step in order to compute ε and the entering arc in step (2b). Since the changes in u_i and v_j cancel one another in the computation of π_{ij}, this dual slack will be correct if it is computed using the value of the dual variables at the start of the basic signature step. The row node label γ_i with which it is compared should have had the ε computed in step (2b) subtracted from it on every iteration since that label was last changed. The following version of step (2b) accounts for this without changing the u_i, v_j and γ_i on each iteration.

Step (2b') For every $j \in \hat{C}^{\sigma}$,[5] set $\hat{v}_j \leftarrow v_j - \sigma$. and do: for every $i \in R^{\ell}$, compute $\hat{\pi}_{ij} = a_{ij} - u_i - \hat{v}_j$. If $\hat{\pi}_{ij} < \gamma_i$, set $\gamma_i \leftarrow \hat{\pi}_{ij}$ and $\text{col}(i) \leftarrow j$. Compute $\gamma_g = \min_{i \in R^{\ell}}\{\gamma_i\}$ and set $k \leftarrow \text{col}(g)$, $\varepsilon \leftarrow \gamma_g - \sigma$, and $\sigma \leftarrow \gamma_g$.

Of course it is necessary to set σ, the cumulative sum of the ε, to zero in step (1) at the start of a basic signature step.

Our second observation is that because $\bar{T}^{\ell} \subset T^{\ell}$ (i.e., each new T^{ℓ} is nested within the previous set T^{ℓ}), it is possible to postpone updating the basis tree until the completion of a full basic signature step. If we store the entering and leaving arcs and ε for each simplex pivot, we can do one 'block pivot' at the end of a basic signature step to update the basis tree and the dual variables in $O(n)$ time.

The above two observations apply, of course, to the standard assignment problem (i.e., the completely dense case). If implemented in this fashion it shows that Balinski's signature method behaves more like a dual simplex method which requires $O(n)$ pivots rather than one that requires $O(n^2)$ pivots.

(iii) *Choosing the Entering Arc*: We need to introduce one more data structure to obtain an improved worst-case bound for the running times of our algorithms that is better than $O(n^3)$. In particular, we need a way to compute ε in step (2b) (γ_g in step (2b')) that does not require $O(n)$ time. One way to do this is by keeping the row node labels γ_i as a heap (priority queue) [1]. Quite recently, a new data structure, the 'Fibonacci heap', was developed by Fredman and Tarjan [11] for implementing heaps that is ideally suited to our needs. Fredman and Tarjan showed how to use this structure to obtain an $O(mn + n^2 \log n)$ algorithm for the assignment problem that is quite different from ours. A Fibonacci heap with k items allows an arbitrary deletion, including the item with minimum value, in $O(\log k)$ amortized time and all other heap operations, such as changing the value of an item and finding the item with minimum value, in $O(1)$ amortized time. For information on heaps and amortized running times see [20] and [21], respectively.

The above amortized running times makes it possible to compute ε in step (2b) (and change the heap when γ_i is changed) with a total cost of $O(n^2 \log n)$ in the worst case, since the total number of simplex pivots (deletion of the item with minimum value) and deletions of row modes from R^{ℓ} (arbitrary deletions from the heap) is at most $O(n^2)$.

In [11] a Fibonacci heap is described as a 'collection of item-disjoint heap-ordered trees'. In addition to the value (or key) γ_i of each item it is shown in [11] how a k-item Fibonacci heap can be represented using an additional four k-dimensional arrays. These arrays enable each item to point to its parent and one of its children and all siblings to be linked together in a doubly-linked circular list. For further details of this representation and the algorithms for performing various operations on Fibonacci heaps we refer the reader to [11].

[5] We use the notation $\hat{C}^{\sigma}$ to indicate the subset of column nodes in $\hat{C}^{\sigma}$ that are connected by at least one admissible arc to the subset of row nodes in R^{ℓ}. Fortunately, the sparse matrix data structure introduced above allows us to identify $\hat{C}^{\sigma}$.

By combining all of the implementational techniques described above we obtain algoirthms for the sparse assignment problem that have a worst-case running time bound of $O(mn + n^2 \log n)$. This bound is the same as the one given by Fredman and Tarjan [11] and is the best that is currently known.

7. Practical matters and future work

Several of the implementational techniques described in the previous sections have practical as well as theoretical value. This is certainly the case for the row node labels γ_i. It is also true for the technique of 'block pivots' and the use of step (2b') in place of step (2b) which allows us to defer updating dual variables until the end of a basic signature step. From the computational analysis given in Section 4, we see that this reduces the total cost of step (2c) from $O(n^3)$ to $O(n^2)$ in the worst case.

On the other hand, the overhead involved in using Fibonacci heaps, probably makes their use in our algorithms impractical. Moreover, we conjecture that the number of pivots required by our algorithms for a sparse problem with m arcs can be bounded above by $O(m)$. This implies that the sparse assignment problem has time complexity $O(mn)$—an intriguing open question.

The sparse matrix data structure depicted in Fig. 4 would probably only be efficient in practice for problems that are fairly sparse. For problems that are only moderately sparse, it would probably be better to access arc weights using a simple hashing scheme, since the low overhead and storage requirements of such a method, would compensate for the time spent accessing nonexistant arcs.

We are in the process of programming our algorithms and intend to report the results of computational tests which will enable us to evaluate the practical efficiency of our algoirthms and the various implementational techniques already discussed. We also intend to study the usefulness of switching from row to column signatures after initialization when the latter has fewer terminal nodes.

From our perspective, the three most interesting open research problems raised by signature methods are:

(i) Can the signature method be generalized to give an efficient algorithm for the nonbipartite weighted matching problem, for which the current best bound is $O(\min\{mn \log n, n^3\})$ [12]?

(ii) Can the signature method be specialized to give an efficient algorithm for the maximum cardinality matching problem?

(iii) Can the signature method be generalized to give a 'genuinely' polynomial algorithm for the transportation problem?

Acknowledgement

I would like to thank Michel Balinski and Thomas McCormick for several stimulating discussions and Robert Tarjan for bringing references [10] and [11] to my attention.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The design and analysis of computer algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] M.L. Balinski, "Signature methods for the assignment problem", to appear in *Operations Research.*

[3] M.L. Balinski and R.E. Gomory, "A primal method for the assignment and transportation problems", *Management Science* 10 (1964) 578-593.

[4] R. Barr, F. Glover and D. Klingman, "The alternating path basis algorithm for assignment problems", *Mathematical Programming* 13 (1977) 1-13.

[5] D. Bertsekas, "A new algorithm for the assignment problem", *Mathematical Programming* 21 (1981) 152-171.

[6] W.H. Cunningham, "A network simplex method", *Mathematical Programming* 11 (1976) 105-116.

[7] W.H. Cunningham and A.B. Marsh, III, "A primal algorithm for optimum matching", *Mathematical Programming Study* 8 (1978) 50-72.

[8] G.B. Dantzig, "Application of the Simplex Method to a Transportation Problem", in: T.C. Koopmans, ed., *Activity analysis of production and allocation* (Wiley, New York, 1951) pp. 359-373.

[9] J. Edmonds and R.M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems", *Journal of the Association for Computing Machinery* 19 (1972) 248-264.

[10] M.L. Fredman, J. Komlós and E. Szemerédi, "Storing a sparse table with O(1) worst case access time", *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science* (1982), 165-169.

[11] M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms", preprint, January, 1984.

[12] Z. Galil, S. Micali and H. Gabow, "Maximal weighted matching on general graphs", *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science* (1982), 255-261.

[13] J.E. Hopcraft and R.M. Karp, "A $n^{5/2}$ Algorithm for maximum matchings in bipartite graphs", *SIAM Journal on Computing* 2 (1973) 225-231.

[14] M.S. Hung, "A polynomial simplex method for the assignment problem", *Operations Research* 31 (1983) 595-600.

[15] M.S. Hung and W.O. Rom, "Solving the assignment problem by relaxation", *Operations Research* 28 (1980) 969-982.

[16] H.W. Kuhn, "The Hungarian method for the assignment problem", *Naval Research Logistics Quarterly* 2 (1955) 83-97.

[17] E. Lawler, *Combinatorial optimization, networks and matroids* (Holt, Rinehart and Winston, New York, 1976).

[18] E. Roohy-Laleh, *Improvements to the theoretical efficiency of the network simplex method,* Ph.D. Thesis, Carleton University (Ottawa, 1981).

[19] D.D. Sleator and R.E. Tarjan, "Self-adjusting heaps", *SIAM Journal on Computing,* submitted.

[20] R.E. Tarjan, "Amortized computational complexity", *SIAM Journal on Algebraic and Discrete Methods,* to appear.