The Linear Assignment Problem

Mustafa Akgül*

Abstract

We present a broad survey of recent polynomial algorithms for the linear assignment problem. They all use essentially alternating trees and/or strongly feasible trees. Most of them employ Dijkstra's shortest path algorithm directly or indirectly. When properly implemented, each has the same complexity: $O(n^3)$ for dense graphs with simple data structures and $O(n^2 \log n + nm)$ for sparse graphs using Fibonacci Heaps.

Introduction

The assignment problem is one of the most-studied, well-solved and important problems in combinatorial optimization. It has numerous applications in various scheduling problems, vehicle routing etc. More importantly, it emerges as a subproblem in many NP hard problems. In particular it occurs as a relaxation of the travelling salesman problem. It has been generalized to bottleneck, quadratic and algebraic cases, see [24, 25] for references.

Solution procedures vary from primal-dual/successive shortest paths [19, 58, 59, 81, 43, 41, 26, 56] (see [35] for a survey), cost parametric [76], recursive [79], relaxation [39, 52], signature based [15, 16, 48, 9, 66, 67, 68] to primal methods [17, 31, 6] to name just a few.

Our aim is to give a rather informal survey of recent polynomial algorithms for the linear assignment problem. Our treatment is a bit biased toward our research in the field.

Most of these polynomial algorithms have the same time complexity: $O(n^3)$ for dense graphs using simple data structures, and $O(n^2 \log n + nm)$ for sparse graphs using Fibonacci heaps [44]. These are currently the best available bounds. Unless otherwise stated explicitly, each of the algorithms discussed has the above complexity.

*Bilkent University, Department of Industrial Engineering, Ankara, Turkey

NATO ASI Series, Vol. F 82
Combinatorial Optimization
Edited by M. Akgül et al.
© Springer-Verlag Berlin Heidelberg 1992

These algorithms share the following features:

i) They solve, in various ways, an increasing sequence of problems to optimality; the last of which being the original problem,

ii) Either they can be implemented using Dijkstra's algorithm as a subroutine after some transformation on the graph, or their behaviour can be better understood and implemented in the terminology of Dijkstra's algorithm,

iii) They work with alternating trees and/or strongly feasible trees.

In section 1, we set up the notation and terminology used in the rest of the paper. We tried to be uniform in our terminology and notation as far as possible. For this purpose, we first translate some of the algorithms to our terminology. Then we discuss algorithms according to our classification. We try to group the algorithms according to motivation and basic algorithmic primitives. Since most of the algorithms are 'near-equivalent' to each other, our classification may seem a bit arbitrary. Starting with section 2, we discuss the Hungarian algorithm, followed by successive shortest path, primal simplex, signature, dual simplex, signature guided, forest and other algorithms.

1 Preliminaries

We view the assignment problem (AP) as an instance of transshipment problem over a directed (bipartite) graph, $G = (U, V, E) = (N, E)$, where U is the set of source (row) nodes, V is the set of sink (column) nodes, $N = U \cup V$ and E is the set of edges. The edge $e = (i, j) \in E$ with *tail* $t(e) = i$ and *head* $h(e) = j$, is directed from its tail to its head, has weight (cost) $w_e = w_{ij}$ and flow x_e. Thus the AP can be formulated compactly as

$$\min \{wx \ : \ Ax = b, \ x \geq 0 \} \tag{1}$$

where $x \in \mathcal{R}^E$, $b \in \mathcal{R}^N$ with $b_u = -1, \ u \in U, \ b_v = +1, \ v \in V$, and A is the node-edge incidence matrix of G.

The dual of (1) is

$$\max \sum_{v \in N} y_v b_v$$

such that

$$y_j - y_i \ \leq \ c_e \ , \forall e = (i, j) \in E \tag{2}$$

Let us set up some notation: for $S,\ X\ ,\ Y \subset N$ and $X \cap Y\ =\ \emptyset$,

$$\left.\begin{array}{c} \gamma(S)\ =\ \{e \in\ E\ :\ t(e)\,,\ h(e)\ \in\ S\} \\ \delta(X,Y)\ =\ \{e\ \in\ E\ :\ t(e)\ \in\ X\,,\ h(e)\ \in\ Y\} \\ \delta^{+}(X) = \delta(X, N \setminus X),\ \delta^{-}(X) = \delta(N \setminus X, X) \\ G[S] = (S,\ \gamma(S)) \end{array}\right\} \tag{3}$$

For a subgraph H of G, we will represent edge set and node set of H by $E(H)$ and $N(H)$. But, very often we will simply write H for its edge set and node set. For $u \in U$, $N^{+}(u)$ will denote nodes or edges incident with u, and for $X \subset U$, we have $N^{+}(X) = \cup_{u \in X} N^{+}(u)$. $d_H(v)$, for $v \in N$ is the degree of node v in the (undirected) subgraph H. Degree 1 non-root nodes are called *leaf*.

The reduced cost of the edge $e = (i,j)$ with respect to y is $\bar{w}_e = \bar{w}_e(y) = w_{ij} - y_j + y_i$. Given a dual feasible y, let $E(y)$ be the *equality set* defined as

$$E(y) = \{\, e \in E\ :\ \bar{w}_e = 0\,\} \tag{4}$$

and *equality subgraph*

$$G(y) = (N,\ E(y)\,). \tag{5}$$

A set of edges $M \subset E$ is a *matching* if degree of every node in $G[M] = (N,\ M)$ is either zero or one. Degree one nodes are called *matched* and degree zero nodes are called *free*. An edge $e \in E \setminus M$ will also be called *free*. A matching M will be called *perfect* if every node is matched. We often assign flow values $x \in \mathcal{R}^E$ as $x_e = 1 \iff e \in M$ and $x_e = 0 \iff e \notin M$. For a dual feasible y, a matching M in $G(y)$ will automatically satisfy complementary slackness conditions

$$x_e > 0 \Longrightarrow \bar{w}_e = 0. \tag{6}$$

Such a $(y,\ M\,)$ pair is often called *compatible*. The importance of the compatible pair concept comes from the observation

Theorem 1 *Let y' be y restricted to $N(M)$. For a compatible pair (y, M) we have:*

i) (y', M) is compatible,

ii) y' and edges in M solve the assignment problem and its dual defined over $G[N(M)]$.

To store the current matching we use an array named **mate**. If $e = (i,\ j) \in M$, we will say $mate(i) = j$ and $mate(j) = i$. Node $i \in N$ will be free if and only if $mate(i) = 0$.

A path in $G(y)$ is *alternating* (with respect to matching M) if the edges are alternately free and matched edges. An *augmenting path* between $i,\ j \in N$ is an alternating path with the additional condition that i and j are free. An *augmentation* is just the switching of matched and free edges in an augmenting path, which increases the size of matched edges.

In the context of primal simplex and dual simplex; it is well-known that any basis of (1) corresponds to a tree T of G. Given any T, it is well-known that the flow values x_e, $e \in T$ are uniquely determined for each compatible ($\sum b_v = 0$) 'supply' vector b. Moreover, the complementary dual basic solution is also uniquely determined once one of the y's is fixed at an arbitrary level.

For every co-tree edge $e \in T^{\perp} = E - T$, $T \cup e$ contains a unique cycle $C(T,e)$, called the **fundamental cycle** determined by T and e. We orient $C(T,e)$ in the direction of e. This will give us a partition of $C(T,e)$ as

$$C(T,e) = C^{+}(T,e) \cup C^{-}(T,e)\ ,\quad e \in C^{+}(T,e) \tag{7}$$

where $C^{+}(T,e)$ contains all edges of $C(T,e)$ having the same orientation as e.

For $f \in T$, $T - f$ will have exactly 2 components, say X and $X^c = N - X$, with $t(f) \in X$. Unconventionally, we will call f the **cut-edge,** and X the **cut-set.**

The set of edges with one end in X and the other in X^c is called the **fundamental cocycle** of G determined by T and f, $D(T,f)$. This can now be partitioned into

$$D(T,f) = D^{+}(T,f) \cup D^{-}(T,f)\ ,\quad f \in D^{+}(T,f)\ , \tag{8}$$

where $D^{+}(T\ ,\ f)$ contains all the edges in the cocycle having the same orientation as f, i.e. $\{j \in E : t(j) \in X\ ,\ h(j) \in X^c\}$.

The dual variable change in primal simplex, dual simplex and primal-dual algorithms will be

$$y_v = \begin{cases} y_v & v \in X^c \\ y_v + \epsilon & v \in X \end{cases} \tag{9}$$

for some $X \subset N$; where ϵ is determined so that for some edge $e \in \delta^{+}(X) \cup \delta^{-}(X)$ we have $\bar{w}_e = 0$ with respect to new dual variables, i.e. ϵ is the amount of dual (in)feasibility of the edge e; $\epsilon = \pm \bar{w}_e$.

Thus, dual variable (potential) change defined by (9) will cause the following changes in reduced costs:

$$\bar{w}_j = \begin{cases} \bar{w}_j - \epsilon & j \in \delta^-(X) \\ \bar{w}_j + \epsilon & j \in \delta^+(X) \\ \bar{w}_j & \text{otherwise} \end{cases} \quad . \tag{10}$$

Cunningham [29] and Barr et al. [18], introduced the concept of strongly feasible tree. Given a specified node, say, r as a root, let $dist_T(x)$ be the distance of the node x from r in the (undirected) tree T, i.e., the number of edges in the unique path from r to x. We say $e \in T$ is directed toward r or a **reverse edge**, if $dist_T(t(e)) = dist_T(h(e)) + 1$, otherwise it is directed away from r or a **forward edge.** A feasible rooted tree is **strongly feasible (SFT)** if $\forall f \in T$, $x_f = 0$ implies f is a forward edge.

Let $\vec{T}$ be the tree obtained from T by changing all reverse edges to forward edges. $\forall v \in N$, $v \neq r$, there is a unique edge $e = (u, v) \in \vec{T}$ with $h(e) = v$. Through such an edge the **parent** of v is defined as $p(v) = u$. $\vec{T}$ is called a **branching** rooted at r. $\vec{T}$ is represented as a data structure using *parent*, *first* (child), *left* (sibling) and *right* (sibling) pointers. A node of degree 2 is completely characterized with parent and first pointers in a tree. Clearly, the root has no parent. For $v \neq r$, when $(p(v), v)$ is deleted from $\vec{T}$, the component containing v is called the subtree rooted at v and denoted by $\vec{T}(v)$. This subtree contains all nodes that can be reached from v by a directed path in $\vec{T}$. Clearly, $v \in \vec{T}(v)$ and $r \in \vec{T}(v) \iff v = r$. Thus in terms of original orientation of tree edges, an edge is forward if and only if its tail is the parent of its head.

Relevant properties of SFTs can be summarized as follows:

Lemma 1 *Let T be a spanning tree for the AP rooted at a sink node, r. Then the following are equivalent.*

(i) T is a SFT,

(ii) Every reverse tree edge has flow 1, and every forward tree edge has flow 0,

(iii) d(r)=1, d(v)=2, $\forall v \neq r, v \in V$ where d(.) denotes the degree of the specific node. □

Clearly (ii) implies that T is primal feasible and (iii) implies that the column signature of T, i.e., the degree sequence of the column (sink) nodes is $(2, 2, 2, \ldots, 2, 1)$. Moreover, if any T has column signature such as above and rooted at a node of degree 1, then such a T is SFT.

Clearly, a dual-feasible tree which is also SFT is an optimal tree. A signature-guided

method changes the tree by linking and cutting edges to obtain a tree having the desired signature, i.e., $(2, 2, 2, ..., 2, 1)$.

Alternating Tree

An *alternating tree* T is a tree rooted at a free source node r so that for each $v \in T$, the path from r to v in T is an alternating path. Moreover it has the following properties:

- $d(v) = 2, \forall\, v \in V \cap T$,
- r is the only free node,
- all leaf nodes are source nodes,
- when matched edges in T are reversed in orientation, the new tree is a branching. Equivalently, all matched edges are reverse and all free edges in T are forward.
- for some natural number k we have the equalities: $|N(T)| = 2k+1$, $|E(T) \cap M| = k$, $|N(T) \cap U| = k+1$, $|N(T) \cap V| = k$.

2 The Hungarian Algorithm

The primal-dual algorithm of Kuhn [58] and Munkres [59] starts with a compatible pair (y, M) ($M = \emptyset$, $y = 0$ is acceptable for $w \geq 0$), and maintains such a pair throughout the algorithm. It searches for an augmenting path by building an alternating tree rooted, say, at a free source node r. When the alternating tree reaches a free sink node, an augmenting path is found. Then the matching is enlarged by augmenting along this path and the process is repeated with a new alternating tree rooted at a free source node, if any. Let us call the work involved between two successive augmentations a **stage**. Thus the primal-dual algorithm needs at most n stages. The alternating tree and the current matching are subgraphs of the current equality subgraph $G(y)$.

When the alternating tree rooted at a free source node r is maximal, dual variables are changed so that at least one new edge (whose tail lies in the alternating tree and whose head is in the outside of the tree) is added to the equality subgraph to allow a larger alternating tree. The alternating tree is maximal means $\delta^+(T) \cap E(y) = \emptyset$. Letting

$$\epsilon = \min\{\bar{w}_e \,:\, e \in \delta^+(T)\,\} = \bar{w}_{\bar{e}}, \qquad \textbf{(findmin)}$$

if $\delta^+(T) \neq \emptyset$, then at least one edge $\tilde{e}$ is added to the alternating tree in the new equality subgraph. In order to achieve this, dual variables are updated according to:

$$y_v \leftarrow \begin{cases} y_v - \epsilon & \text{if } v \in T \\ y_v & \text{otherwise} \end{cases} . \qquad (\textbf{dual} - \textbf{update})$$

Note that as a result of the dual variable change yb is increased by ϵ. $\epsilon > 0$, simply because T was maximal; if there were $e \in \delta^+(T)$, with $\bar{w}_e = 0$, the algorithm should have used it. If however, $\delta^+(T) = \emptyset$, then there is no perfect matching saturating T by Hall's theorem: $N^+(U \cap T) = V \cap T$ and $|U \cap T| = |V \cap T| + 1$ by the definition of the alternating tree. From now on, we will assume, for ease in presentation, that the graph under study has a perfect matching.

It is worth pointing out that a theorem of alternatives comes into the picture at this point. The current alternating tree is maximal means the system

$$A'x = b, \quad x \geq 0$$

has no solution, where A' is the submatrix of A whose columns are indexed by edges in $E(y)$. Then the alternative system

$$\pi A' \leq 0, \quad \pi b = 1,$$

has the solution $\pi = -\chi_{N(T)}$, where $N(T)$ is the node set of T and χ is the *characteristic vector*. Thus the dual-update can be viewed as

$$y \leftarrow y + \epsilon\pi ,$$

and ϵ found by find-min is the largest value maintaining dual feasibility of y. Hall's theorem corresponds to case where π satisfies $\pi A \leq 0, \quad \pi b = 1$; implying that the system $Ax = b, \ x \geq 0$ has no solution, hence G has no perfect matching.

Thus the Hungarian algorithm is an instance of primal-dual algorithm of the general linear programming. Actually, the latter is a generalization of the former. The general primal-dual algorithm is finite whereas the Hungarian algorithm for the assignment problem is polynomial. The polynomiality of the Hungarian algorithm for the assignment problem comes from the continuation of the same alternating tree until an augmentation occurs or a proof that there is no perfect matching available. When we continue working with the same alternating tree, the tree grows at most n times, since we add at least 2 nodes at each 'grow_tree' step. If we change the algorithm so that after each dual-update we start afresh with a free source node as the alternating tree, the algorithm may take

exponential time, since we are only relying on the increase in the objective function. This is exactly what happens in Bertsekas' algorithms [19, 21]. Moreover, if the weights are irrational numbers, then the method may fail to give the optimal solution, for the sequence of objective function values may converge to a value lower than the optimal objective function value as shown by Araoz and Edmonds [14].

We now give the pseudo code for the tree version of the Hungarian algorithm for a stage. Nodes in T are labeled. L contains edges of the form $e = (u, v)$ in the equality subgraph $G(y)$ with $u \in T$ which are not processed yet, and it is automatically updated after each dual-update and grow_tree operation.

The Algorithm A1 /* stage */
Input: Compatible pair $(y,\ M)$, root $r \in U$
Initialization: $T \leftarrow r$, $L \leftarrow N^+(r)$, $done \leftarrow false$
while not done **do**
  if $L = \emptyset$ **then** findmin, dual-update
  else select $e \in L$, $e = (u, v)$, $L \leftarrow L \setminus e$
    if $v \notin T$ **then** /* v is unlabeled */
      if $mate(v) = 0$ **then** Augment, $done \leftarrow true$
      else $w = mate(v)$, $T \leftarrow T + (u, v) + (v, w)$ /* grow_tree */
      endif
    endif
endwhile

Now we would like to show that dual variable updates can be postponed until an augmentation occurs and the amount of dual variable change for each update can be calculated efficiently. Suppose we have k dual variable updates before an augmentation and let T^i be the tree just before $i + 1$'th dual update and y^i be the corresponding dual vector for $i = 0, \cdots, k-1$, and T^k and y^k be the tree and the dual vector when the augmentation detected. Furthermore, let $S_o = T^o$, and $S_i = T^i \setminus T^{i-1}$, $i \leq k$; and $\tilde{S}_i = S_i$, for $i = 0, \cdots, k-1$, $\tilde{S}_k = N \setminus T^{k-1}$. S_i, $1 \leq i \leq k$ is the part of the tree grown after i'th dual-update. Let ϵ_i be the amount of dual variable change and E_i be the set of edges e such that $\bar{w}(y^{i-1}) = \epsilon_i$. Let $\epsilon_o = 0$, $\tilde{\epsilon}_i = \sum_{j=o}^{i} \epsilon_j$, $\epsilon = \tilde{\epsilon}_k$. Clearly,

$$y_v^{i+1} = \begin{cases} y_v^i - \epsilon_i & v \in S_j,\ j \leq i \\ y_v^i & \text{otherwise} \end{cases}$$

and consequently,

$$y_v^k = y_v^o - \sum_{j=i+1}^{k} \epsilon_j, \quad \text{if } v \in S_i,\ i < k \ .$$

Let us define a new dual vector $\tilde{y}$ by

$$\tilde{y}_v = y_v^k + \epsilon, \qquad \text{for } v \in N \ .$$

Clearly, $y^k b = \tilde{y} b$, and the reduced costs with respect to y^k and $\tilde{y}$ are the same. Then

$$\tilde{y}_v = y_v^o + \sum_{j=o}^{i} \epsilon_j = y_v^o + \tilde{\epsilon}_i, \quad \text{for } v \in \tilde{S}_i \ .$$

So the main work is the calculation of $\tilde{\epsilon}_i$. Instead of updating y^i, we will update $\bar{y}^i = \bar{y}$. $\bar{y}$ will be, just before the i'th dual-update or find-min as

$$\bar{y}_v = \begin{cases} \tilde{y}_v & \text{for } v \in S_j,\ j < i \\ y_v^o & \text{otherwise} \end{cases} \ . \tag{11}$$

Let $e \in E_i$, $e = (u, v)$, $u \in S_\ell$, $v \in S_i$, $\ell < i$. We need to show that $\bar{w}_e(\bar{y}) = \tilde{\epsilon}_i \iff \bar{w}_e(y^i) = \epsilon_i$, to prove the validity of the new update. Notice that,

$$\bar{w}_e(y^i) = w_e - y_v^i + y_u^i = w_e - y_v^o + (y_u^o - \sum_{j=\ell}^{i-1} \epsilon_j) = \bar{w}_e(y^o) - \sum_{j=\ell}^{i-1} \epsilon_j \ .$$

On the other hand,

$$\bar{w}_e(\bar{y}) = w_e - \bar{y}_v + \bar{y}_u = w_e - y_v^o + y_u^o + \sum_{j=0}^{\ell-1} \epsilon_j = \bar{w}_e(y^o) + \sum_{j=0}^{\ell-1} \epsilon_j \ .$$

From these, it follows that

$$\bar{w}_e(y^i) = \epsilon_i \iff \bar{w}_e(\bar{y}) = \tilde{\epsilon}_i \ .$$

Thus one can update dual variables as in (11), and compute $\tilde{\epsilon}$'s accordingly. Then one does not need to carry the list L, but the list E_i after each findmin. But then the computation of $\tilde{\epsilon}_i$ and update of $\bar{y}$'s are essentially the same with Dijkstra's algorithm [38].

We now give a modification of the algorithm A1 which implements the above dual-update. Instead of the list $L \subset E(y)$, we carry the node list $Q = V \setminus T$. For $i \in Q$, π_i stores the temporary label $\min\{\bar{w}_{u,i}(\bar{y}) : u \in U \cap T\}$ and $nb(i)$ stores the tail of the edge defining π_i. The routine findmin returns ϵ, u, v, where $v = \text{argmin}\{\pi_i : i \in Q\}$ and $\epsilon = \pi_v$, $u = nb(v)$.

The Algorithm A2 /* stage */
Input: Compatible pair (y, M), (free) root $r \in U$

Initialization: $Q \leftarrow V$, $\pi_i \leftarrow \infty$, $nb(i) \leftarrow 0$, for $i \in N$, $\pi_r \leftarrow 0$, scan(r), $done \leftarrow false$
while not done **do**
  findmin
  if $mate(v) = 0$ **then** Augment, done $\leftarrow$ true
  else
    $Q \leftarrow Q \setminus v$, $w \leftarrow mate(v)$, $y_v \leftarrow y_v + \epsilon$, $y_w \leftarrow y_w + \epsilon$,
    $T \leftarrow T + (u,v) + (w,v)$, scan$(w)$
    } /* grow_tree */
endwhile
  $y_j \leftarrow y_j + \epsilon$ **for** $j \in v \cup N \setminus T$ /* **Extend_Dual** */

scan(i) /* $i \in U$ */
  for $(i,\ j) \in E$ and $j \in Q$ **do**
    $temp = w_{ij} - y_j + y_i$
    if $temp < \pi_j$ **then**
      $\pi_j = temp$, $nb(j) = i$
    endif
  endfor
endscan

3 Successive Shortest Path Algorithms

There is a strong relationship between assignment problem (AP) and shortest path problem (SP). It is known since Ford-Fulkerson [43] that one can solve AP by solving a minimum cost flow problem over an extended graph. Just add a supersource s and connect to every source node via artificial arcs (s,i), $i \in U$ with zero cost and unit capacity, and add a supersink t and arcs $(j,\ t)$, $j \in V$ with zero cost and unit capacity. Original edges retain their costs and get capacity 1 or ∞. Given any graph G and $0-1$ flow vector x, *the residual graph* $RG(x)$ is obtained from G by reversing the direction of edges with flow 1, and multiplying their cost by -1. General primal-dual algorithm for the min-cost problem when specialized to assignment problem (with $w \geq 0$) reduces to:

Algorithm A3

0) Pass from G to extended graph G' with (super)source s and (super)sink t

$$\hat{G} \longleftarrow G', \qquad y = 0$$

1) **for** $i = 1, \cdots, n$ **do**
   Solve SP over $\hat{G}$ (using reduced costs $\bar{w}_{ij} = w_{ij} - y_j + y_i$ (†))
   Send one unit of flow from s to t in $\hat{G}$ with dual variable vector π.

$$\hat{G} \longleftarrow R\hat{G}, \qquad y \longleftarrow y + \pi$$

   endfor

2) Edges in $\hat{G}$, except artificial edges, in the reverse direction give the optimum matching and y is an optimal dual vector.

Since in passing to residual graph the sign of edge costs change, (without using (†)), one can not use Dijkstra's algorithm and hence complexity of SP becomes $O(n^3)$ resulting $O(n^4)$ complexity for dense AP's. Edmonds-Karp[41] and Tomizawa[81] independently observed that one can work with reduced costs. Since edges subject to reversing are on the shortest path tree in the current graph, their reduced costs are zero, whence remain zero. Thus, edge costs in SP calculations remain nonnegative and hence one can use Dijkstra's algorithm resulting $O(n^3)$ or $O(n^2 \log n + nm)$ algorithm depending on density of the graph and data structures used.

The classical Kuhn's algorithm grows only one alternating tree rooted at a source node. To realize Kuhn's algorithm by the above algorithm one does not need to add a supersource; but choose a free source node as the root for SP.

The Shortest Path Problem in the above algorithm is the single source problem. In other words, one needs to reach or label every node in the original graph. Since Dijkstra's algorithm is a special case of general primal-dual algorithm[65], one does not need to form full shortest path tree. In other words, one can stop SP algorithm after at least one free sink node is reached. Then one can extend dual-variable vector to unlabeled nodes by assigning the last label to all of these nodes.

One can start with any compatible pair (y, M) instead of $M = \emptyset$, $y = 0$. For random problems the classical row minimum/column minimum yields an initial matching saturating %75 of nodes on the average[35]. Nawijn and Dorhout [60] studied the size of maximum matchings in $G(y)$, where y is obtained by the classical row reduction followed by column reduction and G is the complete bipartite graph. Under non-degeneracy and uniform distribution of cost coefficients assumptions, they showed that, asymptotically, the expected size of a maximum matching in $G(y)$ is equal to %80 n.

There are several successive shortest path algorithms e.g., [35, 26, 42, 47, 56, 27], differing mainly in the way they solve the shortest path problems. For some recent

improvement in data structures to solve the shortest path problem see [44, 4] and for earlier related works see, e.g., [33, 34, 36, 37, 40, 45, 49, 50, 53, 61, 62, 70, 80].

There is an alternative to residual graph. Given M one can *shrink* (contract) the edges in M; i.e. $e = (u, v)$ can be replaced by a pseudo node, say, $\bar{e}$. Let $\bar{G}$ be the graph resulting from shrinking, and $\bar{T}$ be the corresponding shortest path tree. Replacing each pseudo node $\bar{e} \in \bar{T}$ with the edge e appropriately, we obtain an alternating tree. Successive shortest path algorithms are performing this shrinking and unshrinking operations implicitly.

Relaxation Methods

We now discuss 1969 algorithm of Dinic-Kronrod [39] and 1980 algorithm of Hung-Rom [52]. Even though Dinic-Kronrod algorithm is published a decade ago, it did not get the attention it deserves. We believe this is partly due to the facts that: i) the paper does not use LP terminology, ii) it contains significant typographical errors, and iii) the translation is not very good. However, when properly implemented it should be faster than Hung-Rom algorithm.

Both algorithms work with semi-assignments and utilize star graphs. A semi-assignment is a many-to-one mapping from U to V (or from V to U). A star is a complete bipartite graph $K_{1,k}$ for some k; that is a source node is connected to k sink nodes or vice versa for $K_{k,1}$. A semi-assignment $\tilde{M}$ decomposes into a matching M and a collection of stars. In a star, there can be one matched edge. In both algorithms, the selection of matching edge within a star is postponed until the star reduces to a single edge. This 'equal employement' behaviour saves a little work.

Let us now give an equivalent description of Dinic-Kronrod algorithm in our terminology. Let us apply the classical column minimum, find a dual-feasible y, semi-assignment $\tilde{M}$ in $G(y)$, with matching part $M \subset \tilde{M}$. Let $U_- \subset U$ be set of free nodes, and U_+ be set of nodes of degree ≥ 2, and $U_o = U \setminus (U_- \cup U_+)$. Nodes in U_o are matched by the edges in M. Let $\tilde{V}_+$ be the neighbour set of U_+ with respect to $\tilde{M}$. A *stage* of Dinic-Kronrod algorithm starts with the selection of $r \in U_-$, and obtains a new semi-assignment for which $r \notin U_-$. Thus the number stages is bounded by the initial $|U_-|$. The algorithm for a stage amounts to finding a shortest path from r to $\tilde{V}_+$ on the graph $G[V \cup U_o + r]$ with edges in M reversed in orientation. Let P be such a path and $t \in \tilde{V}_+$ be the end of P and $u \in U_+$ be the node assigned to t by $\tilde{M}$. Then t is removed from star of u and $\tilde{M}$ is shifted along the path P.

Hung-Rom algorithm starts with row minimum and obtains a dual-feasible y, a semi-assignment $\tilde{M}$. Let us define V_-, V_o, V_+ as the set of free nodes, matched sink nodes, and nodes with degree ≥ 2 with respect to $\tilde{M}$. In a *stage*: i) they choose an $r \in V_+$ as a root, ii) form a shortest path tree spanning $\bar{N} \equiv N \backslash V_-$ with the edges in $\tilde{M}$ reversed in orientation, iii) choose a $t \in V_-$, extend T and y to t by an edge e via $e = \text{argmin}\{\bar{j} : j \in \delta(\bar{N}, t)\}$, and iv) change the semi-assignment along the path in T from r to t. When demand vector b is relaxed as $b_v = d_{\tilde{M}}(v)$, for $v \in V_+$, T is SFT for the resulting transshipment problem over $\bar{N}$.

In Dinic-Kronrod algorithm only one star is involved; whereas in Hung-Rom algorithm one star is chosen as a root and all others are forced to be on the shortest path tree T. Thus in Hung-Rom algorithm the shortest path problem is solved on a larger set of nodes. Moreover 'relaxation' in Dinic-Kronrod is combinatorial whereas in Hung-Rom it is algebraic.

Engquist [42] presented an algorithm which is essentially the same as the Dinic-Kronrod algorithm. It is described in LP terminology, and involves shrinking and/or reorienting semi-matched edges. He reports that his code is about six times faster than Hung-Rom code.

4 Primal Simplex Methods

Dantzig specialized the simplex method to networks early in 1951. The network simplex method in general is very efficient for network flow problems. This efficiency comes mainly from the fact that the network simplex algorithm works combinatorially over trees rather than algebraically over the matrices.

There are several efficient primal simplex algorithms for the assignment problem, either especially designed for the assignment problem [18] or designed for the transshipment problem [46, 75]. Naturally, they all work reasonably well in practice, but theoretically they are exponential algorithms.

When the network simplex method is specialized to assignment problems, degeneracy comes into picture. For an $n \times n$ assignment problem there are $n - 1$ degenerate and n non-degenerate edges in any basis. It has been observed by several researchers that about %90 percent of pivots are degenerate in an assignment problem. Roohy-Laleh [72] exhibits a family of problems with exponentially long non-degenerate pivot sequences.

Cunningham [29, 30] while devising a network simplex method which does not cycle

introduced the concept of *strongly feasible tree*. Barr, Glover and Klingman, independently and simultaneously introduced the alternating basis tree. It turns out that, a strongly feasible tree for an assignment problem is exactly an alternating basis tree. An alternating basis tree resembles the alternating tree of the primal-dual algorithm. In fact, an alternating tree becomes a strongly feasible tree after an augmentation if rerooted at the free sink node causing augmentation.

In a primal simplex algorithm, if $\bar{w}_e \geq 0$, $\forall\, e \in E$ then T is optimal. Otherwise an edge $e \in T^{\perp} = E \setminus T$ is chosen with $\bar{w}_e < 0$ as the pivot edge. Then a flow of value θ is sent through $C(T,e)$ in the direction of e. The cut edge, f, θ and the flow update can be described as:

$$\left.\begin{array}{l} \theta = x_f = \min\{\, x_j \,:\, j \in C^-(T,e)\,\} \\ \\ x_j = \left\{ \begin{array}{ll} x_j + \theta & j \in C^+(T,e) \\ x_j - \theta & j \in C^-(T,e) \\ x_j & \text{otherwise} \end{array} \right. \\ \\ T = T + e - f \end{array}\right\} \,. \tag{12}$$

y is updated so that $\bar{w}_e = 0$. For more information on the simplex method see, e.g. [28, 29, 54, 22, 71, 3]. Following the convention in [6, 18], we will choose the root of the SFT as a source node and use SFT' to differentiate from the previous one. Then T is SFT' if $\forall f \in T$, $x_f = 0$ implies f is a reverse edge. We need to classify co-tree edges as **forward, reverse** and **cross.** $e \in T^{\perp}$ is a forward edge if $t(e)$ lies on the unique path from r to $h(e)$ and a reverse edge if $h(e)$ lies on the path from r to $t(e)$. Otherwise a co-tree edge is called a cross edge. For nodes u and v, the **nearest common ancestor** NCA(u,v) is the last node common to paths from r to u and v respectively. Then, $e = (u,v) \in E$ is forward if $u = NCA(u,v)$, reverse if $v = NCA(u,v)$, otherwise e is a cross edge.

When rooted at a source node, a SFT' has the following properties:

Lemma 2

i) *Every forward edge has flow value* 1, *and every reverse edge has flow value* 0.

ii) *The root has degree* 1, *every other source node has degree* 2.

iii) *If* e, f *satisfy*

$$e \in T^{\perp}\,,\; f \in C(T,e)\,,\; t(e) = t(f)\,, \tag{13}$$

then the selection of f *as the departing variable is valid and maintains strong feasibility.*

iv) *For $e \in T^{\perp}$, the pivot determined by e and (13) is nondegenerate iff $e \in T^{\perp}$ is forward iff $f \in T$ is forward.*

v) *For e, f satisfying (13) the pivot is nondegenerate iff $r \in X$.*

vi) *For e, $f = (u, w)$ satisfying (13): the pivot is degenerate iff $X = \vec{T}(u)$, and the pivot is nondegenerate iff $X = N \setminus \vec{T}(w)$.* □

Cunningham and Roohy-Laleh [72] developed a genuinely polynomial primal simplex algorithm. The algorithm needs $O(n^3)$ pivots and $O(n^5)$ time in the worst case. The algorithm uses strongly feasible trees.

Hung [51] gave a polynomial primal simplex method that requires $O(n^3 \log \Delta)$ pivots, where $\Delta = \Delta_0$ and $\Delta_k = wx^k - wx^*$ is the difference in the objective function value between the current solution x^k and an optimal solution x^*, and x^o is the initial basic solution. Let $\beta_k = \min\{\bar{w}_e : e \in E\} < 0$, be the most negative reduced cost at the iteration k (Dantzig's rule). From the equation $w\,x = \bar{w}\,x + w\,\bar{x}$, one obtains

$$\Delta_k = w\,x^k - w\,x^* = -\sum \bar{w}_e\,x^*_e \leq -\beta_k n\ .$$

Suppose the pivot with reduced cost β_k is non-degenerate, then

$$\Delta_{k+1} = \Delta_k + \beta_k \leq \Delta_k - \frac{\Delta_k}{n} = \Delta_k(1 - \frac{1}{n})\ .$$

Thus after k non-degenerate pivots with Dantzig's rule, we have

$$\Delta_k \leq \Delta_o(1 - \frac{1}{n})^k\ .$$

Assuming integral w, when $\Delta_k < 1$ the current solution x^k is optimal. Thus, the number of non-degenerate pivots by Dantzig's rule is bounded by $O(n \log \Delta)$. Cunningham [30] earlier bounded the number of degenerate pivots at an extreme point by $n(n-1)$ by utilizing strongly feasible trees and a certain pivot rule. Hung performs all available degenerate pivots ($O(n^2)$) to ensure that the first available non-degenerate pivot has the largest reduced cost. Combining these, one obtains the given bound.

Cunningham [29], Orlin [63] and Srinivasan-Thomson [76] observed the relation between strongly feasible trees and a classical perturbation technique. For ϵ small enough, consider the perturbed b vector

$$b'_i = -1 + \epsilon,\ i \in U, \qquad b'_r = 1 - n\epsilon,\ b'_v = 1,\ v \in V,\ v \neq r, \tag{14}$$

where r is the root of the tree. Then any basic feasible solution of $Ax = b'$, $x \geq 0$ is non-degenerate, and the resulting tree is strongly feasible tree for the unperturbed system.

Orlin [63] using this perturbation technique reduced the bound on the number of pivots to $O(n^2 \log \Delta)$. To see this, notice that for the perturbed system every pivot is non-degenerate, (for each tree edge e we have $x_e \geq \epsilon$). Thus, for each pivot with Dantzig's rule we have

$$\Delta_{k+1} \leq \Delta_k + \beta_k \epsilon,$$

and for $\epsilon = \frac{1}{2n}$, we obtain

$$\Delta_k \leq \Delta_o (1 - \frac{1}{2n^2})^k,$$

which implies $O(n^2 \log \Delta)$ pivot bound. He later reduced the bound to $O(n^2 m \log n)$ where m is the number of edges and n is the number of nodes in the graph by showing that there exists an equivalent network with cost coefficients bounded by $4^m (m!)^2$; and hence proving $\log \Delta$ is $O(m \log n)$. The above algorithms, [51, 63] at least implicitly, are influenced by the ellipsoidal algorithm . Their common feature is the reduction of the objective function value by a fraction depending on n or m, independent of the rest of the problem parameters. In ellipsoidal algorithm this ratio is $\exp(\frac{-1}{m^2})$, whereas, say, in Orlin's algorithm $\exp(\frac{-1}{n^2})$. It is worth mentioning that for totally unimodular linear programs the ellipsoidal algorithm needs $O(m^2 \log (m||c|| \, ||b||)$ iterations [5, ch 4], where as before m is the number of variables, and $||x||$ denotes the (euclidean) norm of the vector x.

In [6] we presented a primal simplex algorithm with $O(n^2)$ pivot and $O(n^3)$ time bound. We cast the problem as an instance of transshipment problem and work on a directed graph. The algorithm has three features. We consider an increasing sequence of subgraphs, the last of which is the graph of the original problem, and each one differs from the previous one by addition of some of the edges incident with one node. In matrix terms, we solve the subproblems defined by principal minors of the cost matrix. The motivation for this approach came from the author's work on the shortest path problem [8]. Moreover, we restrict the feasible basis to strongly feasible trees. Interestingly, degeneracy together with strongly feasible trees is very helpful, at least theoretically. The third component of the algorithm is the use of Dantzig's rule restricted to the current subgraph. Our algorithm is a purely primal simplex algorithm, because we carry a full basis of the original problem all the time. We do not attempt to evaluate the change in the objective function value. Instead, we study the structure of the set of nodes on which we make dual variable changes during the solution of the current subproblem. We call these sets cutsets. It turns out that : i) cutsets are disjoint, ii) edges originating from a cutset are dual-feasible once for all for the subgraph under consideration, iii) dual infeasible edges have the property that their tails have no dual variable change and iv) each

node is subject to at most one dual variable change. Thus passing from a subproblem of size $k \times k$ to a subproblem of size $(k+1) \times (k+1)$ can be done with at most $k+2$ pivots. Hence, we have the bound $\frac{1}{2}n(n+3) - 4$ for the number of pivots. The total number of non-degenerate pivots is bounded by $n-1$. The total number of consecutive degenerate pivots is bounded by $\frac{1}{2}(n+2)(n-1)$. All of these bounds are sharp.

Ahuja and Orlin [2] presented a new primal simplex algorithm with $O(n^2 \log W)$ pivot and $O(nm \log W)$ time bound where W is an upper bound on w_{ij}. Their pivot rule is a variation of Dantzig's rule and employs scaling. Initially, the parameter $\alpha = W$, and any edge with $\bar{w}_e \leq -\frac{1}{2}\alpha$ is a valid pivot edge. When there is no valid edge then α is replaced with $\frac{\alpha}{2}$.

5 Signature Methods

The dual simplex algorithm for the transshipment problem starts with a dual feasible tree. If $x_f \geq 0$, $\forall f \in T$, then T is optimal. Otherwise the algorithm chooses an $f \in T$ with $x_f < 0$, as the leaving edge (cut-edge), and chooses a co-tree edge $e \in T^{\perp}$ as the entering (pivot) edge to satisfy dual-feasibility via

$$\epsilon = \bar{w}_e = \min\{\, \bar{w}_j \,:\, j \in D^-(T,f)\,\}\,. \tag{15}$$

Thus the result of a pivot is the new tree $T' = T + e - f$. A pivot will increase flows on the edges $C^+(T,e)$ by $\theta = -x_f$, decrease flows on $C^-(T,e)$ by θ, and increase the reduced cost of the edges in $D^+(T,f)$ by ϵ and decrease that of the edges in $D^-(T,f)$ by ϵ. Note that for Y being the component of $T-f$ containing $t(f)$ we have the equalities $\delta^+(Y) = D^+(T,f)$ and $\delta^-(Y) = D^-(T,f)$.

Since a SFT is automatically primal feasible, a dual feasible SFT tree is optimal. Balinski [15] starts with a specially structured dual-feasible tree and tries to obtain a SFT. Balinski's algorithm performs essentially dual-simplex pivots, but the algorithm *never* evaluates or updates flow values explicitly. Its behaviour is dictated by the degree structure or signature of the tree. Even though Balinski starts with what is known as the Balinski tree, one can start with any dual feasible tree [48]. Let $V_+ = \{v \in V \,:\, d(v) \geq 3\}$, $V_- = \{v \in V \,:\, d(v) = 1\}$. A SFT can be characterized by either $|V_+| = 0$, or equivalently by $|V_-| = 1$.

Balinski defines the **level** of a tree as the cardinality of V_-. By **stage** we mean the total work involved in reducing level by 1. Then Balinski's algorithm for a stage can be described as:

Algorithm A4 /* stage */

Input: $\bar{s} \in V_+$, $t \in V_-$

reroot T at t, $s \leftarrow \bar{s}$

while $s \notin V_-$ **do**

$f = (p(s), s)$ /* cut-edge */

$e = \text{argmin}\{\bar{w}_j : j \in D^-(T, f)\}$ /* link-edge */

$T \leftarrow T + e - f$

$s \leftarrow t(e)$

endwhile

Let $s_1, s_2, \cdots, s_k$ be the nodes encountered at the while loop, and $f_1, \cdots, f_k$, and $e_1, \cdots, e_k$ be the corresponding cut-edges and link-edges respectively. Let X_i be the component of $T_i - f_i$ containing s_i. Thus $f_i \in \delta^-(X_i)$ and $e_i \in \delta^+(X_i)$. Since $s_{i+1} = t(e_i) \in X_{i+1}$, and $s_{i+1} \notin X_i$, it follows that $X_{i+1} \supset X_i$ and s_i's are distinct, and $\{s_1, s_2, \cdots, s_i\} \subset X_i$. Since the while loop breaks once $s_i \in V_-$, the number of iterations in a stage is bounded by $|V \setminus V_-|$. So, if the current level is k, then one can have at most $n - k$ pivots. Thus the total number of pivots is bounded by

$$\sum_{k=2}^{n-1}(n-k) = \sum_{j=1}^{n-2} j = \binom{n-1}{2}$$

Goldfarb [48] developed a sequential version of the signature method which starts with 1×1 problem, and solves $k+1 \times k+1$ problem using an optimal SFT solution to $k \times k$ problem, for $k = 1, \cdots, n-1$. Given a SFT T' for the $k \times k$ problem of the graph $G' = (U', V')$ rooted at $r \in V'$, with dual vector y, T' and y is extended for the $k+1 \times k+1$ problem of $\hat{G} = (U' + u, V' + v)$ as:

$$y_v = \min\{y_i + w_{iv} : i \in U'\} = y_{\bar{u}} + w_{\bar{u}v}$$

and

$$y_u = \max\{y_j - w_{uj} : j \in V' \cup v\} = y_{\bar{v}} - w_{u\bar{v}}$$

and the new tree as

$$T \leftarrow T' + (\bar{u}, v) + (u, \bar{v}) .$$

If $\bar{v} = v$ or $\bar{v} = r$ then T is a SFT with root r or root v, and the solution of the new subproblem is at hand. Otherwise $d(\bar{v}) = 3$ and $d(r) = d(v) = 1$. Then for $\bar{s} = \bar{v}$ and $t = r$ or $t = v$ the previous algorithm requires at most k pivots.

6 Purely Dual-Simplex Algorithms

The above algorithms, strictly speaking, are not dual simplex algorithms for they may cut an edge with zero flow (Balinski tree) or with positive flow (arbitrary dual-feasible tree or Goldfarb's variant).

Balinski [16] later introduced a notion of Dual Strongly Feasible Tree (DSFT) for the assignment problem with very strong properties. Paparrizos [69] extended this concept to the transportation problem somehow, but the resulting algorithm for the transportation problem is pseudo-polynomial.

Let T be a dual-feasible tree for AP rooted at a sink node r. Let $\tilde{L}$ be the set of edges in T attached to r and to a leaf, i.e. $\tilde{L} = \{(u,r) \in T : d(u) = 1\}$. Then T is DSFT if:

i) $f \in T \setminus \tilde{L}$, reverse $\Longrightarrow x_f \leq 0$,

ii) $f \in T$, forward $\Longrightarrow x_f \geq 1$.

Notice that for $f \in \tilde{L}$ we have $x_f = 1$. Let $U_+ = \{u \in U : d(u) \geq 3\}$ and $U_- = \{u \in U : d(u) = 1\}$. The relevant properties of DSFT are given in [16] as:

Lemma 3 *Let T be a DSFT rooted at $r \in V$. Let $u \in U_+$ and let $f = (u, p(u))$. Then*

i) $x_f \leq -1$, and

ii) the selection of f as the cut-edge of a dual-simplex pivot maintains DSFT.

Thus DSFT is maintained as long as the cut-edge is a reverse edge $f = (u,v)$, with $d(u) \geq 3$.

Balinski's dual-simplex algorithm for a stage is:

Algorithm A5 $(\bar{s})$ /* stage */

$s \leftarrow \bar{s}$

while $d(s) \geq 3$ **do**

let $f = (s, p(s))$ and $e \in T^{\perp}$ via (15)

$T \leftarrow T + e - f$

$s \leftarrow t(e)$

endwhile

Letting e_i, f_i, T_i, and Y_i be respectively pivot edge, cut-edge, tree and the component of $T_i - f_i$ containing $s_i = t(f_i)$, with $T_{i+1} = T_i + e_i - f_i$, it follows easily that $Y_i \subset Y_{i+1}$,

s_i's are distinct and $\{s_1, \cdots, s_i\} \subset Y_i$. Thus the number of pivots in a stage is bounded by $|U \setminus U_-|$ measured at the beginning of the stage. Hence we obtain the same bound for the total number of pivots as before.

We should point out that the final SFT is rooted at a source node. This is so because the algorithm works with row signatures instead of column signatures (this is our version of Balinski's algorithm).

Akgül's Sequential Algorithm

Akgül [7] presented a sequential algorithm which, starting with trivial problem (of a perturbed system) AP_o, solves $AP_1, \cdots AP_n$ and from the last one, obtains an optimal solution of the original problem. The final solution need not be a tree solution, but a collection of strongly feasible trees each rooted at a source node. Let $V = \{v_1, v_2, \cdots, v_n\}$ be an arbitrary ordering of sink nodes, and let $r \equiv v_o$ be a dummy sink node. Form the new graph $G^{\#} \equiv (U, V, E + \{(u, r),\ u \in U\})$, and define G_k as $G_k = G^{\#}[U + \{v_o, \cdots, v_k\}]$, and let AP_k be the transshipment problem over G_k with $b_u = -1$, $u \in U$, $b_{v_j} = 1$, $1 \leq j \leq k$, and $b_r = n - k$. Assign $w_{ru} = K$, for $u \in U$ (for the artificial edges). Clearly $y_r = K$, $y_u = 0$ for $u \in U$ is feasible for AP_o, and G_o is a feasible hence optimal tree for AP_o. Here K is a large constant.

Let T_k^* be an optimal DSFT for AP_k. Then $T_k^* - r$ will be disjoint union of (primal) SFT's each rooted at a source node together with $n - k$ isolated source nodes. Letting $v \equiv v_{k+1}$, G_{k+1} contains, in addition to G_k, the node v and the edges $\delta(U, v)$. Given T_k^* and v, the dual vector y is extended to the node v and a new edge is added to T_k^* to obtain T, a DSFT for G_{k+1} via:

$$y_v \equiv \min\{w_{uv} + y_u \ :\ (u, v) \in E\} = w_{\bar{s}v} + y_{\bar{s}}$$

and $T = T_k^* + (\bar{s}, v)$. If $d(\bar{s}) = 2$ then T is optimal. Otherwise, $d(\bar{s}) = 3$, and all the reverse edges from r to $\bar{s}$ have flow value -1. Even though a dual simplex algorithm can choose any one of them as a cut-edge, there is a *unique* cut-edge which maintains DSFT, namely $f = (\bar{s}, p(\bar{s}))$. Solving AP_{k+1} starting with the above T will be referred as *stage* $k + 1$. The sequential algorithm for solving AP_{k+1} is as follows:

Algorithm A6 $(\bar{s})$ /* stage */

$s \leftarrow \bar{s}$

while $d(s) = 3$ **do**

let $f = (s, p(s))$ and $e \in T^{\perp}$ via (15)

$T \leftarrow T + e - f$
$s \leftarrow t(e)$
endwhile

The arguments given for the Balinski's algorithm remain valid. One can easily show that the total number of pivots is bounded by $\binom{n}{2}$.

The $O(n^2)$ bound on the number of pivots will not translate into $O(n^3)$ time bound for the dense case. For this one needs to utilize the nested structure of various X_i's or Y_i's. One can implement these algorithms so that the total work in a stage takes $O(n^2)$ for the dense case with simple data structures and $O(m + n \log n)$ for the sparse case using Fibonacci heaps. For details see [16, 48, 7, 9].

Paparrizos [68] developed a sequential dual simplex algorithm similar to ours. He starts with a Balinski tree and from that tree he drives the sequence of problems to be solved. The solution of the subproblems are essentially the same with ours.

7 Signature Guided Algorithms

Paparrizos [66] introduced a non-dual signature method which solves the n by n assignment problem in at most $O(n^2)$ pivots and $O(n^4)$ time.

In [9], a modification of Paparrizos' algorithm, is given: it is a dual-feasible signature-guided forest algorithm which terminates with a strongly feasible tree.

First, we will describe Paparrizos'[66] algorithm in our notation. His algorithm works with, what we call, **layers**. Initial tree is dual-feasible and is rooted at a source node and all sink nodes of degree 1 are attached to this source node, i.e., a Balinski tree. A **layer** consists of two parts: **decompose** and **link**. To **decompose** a tree, a sink node of degree ≥ 3 which is minimal in distance to the root is identified. If there is no such node, then T is SFT and hence it is optimal. Let $v \in V$ be such a node. Then the edge $(p(v), v)$ is deleted and the cutoff subtree rooted at v is identified as a 'candidate tree', and is denoted as say, T_v. The process is continued until the tree rooted at r contains no sink nodes of degree ≥ 3. The tree rooted at r is called T_+ and T_- is the collection of candidate trees. The **link** part of the algorithm is as follows.

while $T_- \neq \emptyset$ **do** begin
$e \leftarrow \text{argmin}\{\bar{w}_e : e \in \delta(T_-, T_+)\}$
$\epsilon \leftarrow \bar{w}_e$, let $t(e) \in T_k$

$y_v \leftarrow y_v - \epsilon \ \forall v \in T_k$
$T_- \leftarrow T_- \setminus T_k$
$T_+ \leftarrow T_+ + T_k + e$
endwhile

The main invariant during **link** is that the subtree T_+ is dual-feasible, i.e., edges in $\gamma(T_+)$ are dual-feasible. Consequently, when a **layer** is finished, the new tree is dual-feasible. Since the **layer algorithm** is continued until T is SFT, the algorithm stops with an optimal tree. The pivot bound is $O(n^2)$ but the number of layers also has the same bound. This results in an $O(n^4)$ algorithm. Moreover, during a layer, dual-feasibility may be violated.

In the new algorithm, the layer concept is abandoned altogether. After linking a subtree to T_+ via sink node v, instead of linking other trees in T_- to T_+, *decompose* is applied if possible. So the algorithm performs a simpler form of *link* and decompose alternatively (some decompose could be vacuous). The whole process is divided into **stages** which will facilitate an efficient implementation of the algorithm.

We also make dual variable changes on the whole T_- rather than on a subtree of it. Consequently, we obtain a dual feasible algorithm with the state of the art complexity.

Now, we describe the new algorithm.

For a tree (forest) T, let $\sigma_1 = \sigma_1(T), \sigma_2, \sigma_3$ be the number of sink nodes of degree 1, degree 2 and degree at least 3 respectively. Hence, T is SFT if and only if $\sigma_1 = 1, \sigma_2 = n-1, \sigma_3 = 0$. The **level** of a tree is $\sigma_1(T)$. Our algorithm works with **stages** through each of which σ_1 is reduced by 1. The computational cost of a stage will be $O(n^2)$ for the dense case and $O(n \log n + m)$ for the sparse case.

We start with the well-known 'Balinski-tree' rooted at a source node r. We then apply **decompose**. Thus, we obtain T_+, and $T_- = \bigcup_{i=1}^{l} T_i$ and $l \leq \sigma_3$.

Our **link** routine (at say k^{th} iteration) is as follows:
begin
$e = (u, v) = \arg\min\{\bar{w}_e : e \in \delta(T_-, T_+)\}$
let $\epsilon = \bar{w}_e$ and $t(e) = u \in T_q$
$y'_z \leftarrow y_z - \epsilon \ \forall z \in T_-$
$T'_+ \leftarrow T_+ + T_q + e$
$T'_- \leftarrow T_- \setminus T_q$
end

where $T = T_- \cup T_+$ is the forest at the k^{th} iteration and $T' = T'_- \cup T'_+$ is the forest obtained after the k^{th} link.

A *link* followed by a, possibly vacuous, *decompose* is called a **pivot**. Let $d(v)$ be the degree of v in T'_+. Depending on $d(v)$ where $v = h(e)$ (e is the link-edge at k^{th} link), we identify 3 types of pivots.

$d(v) = 3$: In this case, we cut the edge $(p(v), v)$ from T'_+, and add the cutoff subtree rooted at v to T'_-. This is called a **type 1** pivot.

$d(v) = 2$: In this case, a stage is over. Here, we check whether the subtree of T'_+ rooted at v, which is $T_q + e$ contains any sink node(s) of degree ≥ 3. If so, we apply decompose and add the resulting subtrees to the collection T'_-. Otherwise, we just continue. The former case is called **type 2** pivot and the latter **type 3** pivot. In **type 2** pivots, the number of subtrees in T'_- may increase by more than one. In **type 1** pivots, the number of subtrees in T'_- is the same as that of T_-, and in **type 3** pivots the number of subtrees in T'_- is one less than that of T_-.

The algorithm continues until $T_- = \emptyset$ and terminates with a strongly feasible and hence an optimal tree T_+.

Lemma 4 *The new forest $T' = (T'_+, T'_-)$ is dual-feasible.*

Proof: It suffices to show that with respect to dual variables y', forest T is dual-feasible and the reduced cost of the link-edge e is zero. Clearly, the reduced costs of the edges in $\gamma(T_-)$ and $\gamma(T_+)$ do not change. The reduced costs of the edges in $\delta(T_-, T_+)$ decrease by ϵ and those in $\delta(T_+, T_-)$ increase by ϵ. Since $\epsilon \geq 0$, edges in $\delta(T_+, T_-)$ remain dual-feasible. Edges in $\delta(T_-, T_+)$ are also dual-feasible simply because of the way link-edge e is chosen. With respect to y', edge e has zero reduced cost. Therefore, $T + e$ is dual feasible. Clearly, decompose routine does not affect dual-feasibility. As a result, T' is dual-feasible. □

Since the algorithm maintains dual-feasibility and stops with SFT, it is valid.

The total number of pivots is bounded by $(n-1)(n-2)/2$.

8 Forest Algorithms

Here we present a forest version of the classical primal-dual algorithm of Kuhn implemented in the spirit of successive shortest paths. Strictly speaking, we successively solve a shortest path problem over the residual graph whose arc costs are reduced costs, until

optimality. We grow a forest of alternating trees each rooted at a free source node, and allow strongly feasible trees each rooted at a sink node 'float' around. We do not necessarily stop when an augmenting path is found. When an augmentation happens we do not discard the whole alternating forest. We reroot the tree subject to augmentation on the free sink node causing augmentation and obtain a strongly feasible tree rooted at that sink node. When we grow an alternating tree with an edge whose head lies in a non-trivial strongly feasible tree, if necessary we decompose that tree and append a maximal subtree to the alternating tree. The subtree may contain more than one matched edges, which we call as 'block pivot'.

Block Pivots

The key to our 'block pivot' is the relationship between an alternating tree and a strongly feasible tree. Recall that if T is an alternating tree rooted at a source node r and T is subject to augmentation with edge $e = (u, v)$, then $T' = T + (u, v)$ is a strongly feasible tree when rerooted at the (previously free) sink node v. Recall also that it is very easy to identify the matched edges in a SFT. Thus after an augmentation we reroot the current alternating tree and retain it as a SFT.

Let $\tilde{T}$ be an alternating tree rooted at r, and T be a SFT rooted at $q \in V$, and suppose in the primal-dual algorithm we apply grow_tree step with the edge $e = (u, v)$, where $u \in \tilde{T}$ and $v \in T$. Ordinarily, we grow the alternating tree by adding edges e and $(v, mate(v))$ to $\tilde{T}$. In a block pivot, we add e and T_v the subtree of T rooted at v to $\tilde{T}$ $(N(T_v) = N(\vec{T}_v))$. Thus, if $v = q$ then we have $\tilde{T} \leftarrow \tilde{T} + (u, v) + T$, otherwise we delete the edge $f = (p(v), v)$ from T obtaining, say, T_q and T_v and let $\tilde{T} \leftarrow \tilde{T} + (u, v) + T_v$. Clearly, in both cases $\tilde{T}$ will be an alternating tree, and in the latter case T_q will remain a SFT.

Like other primal-dual/successive shortest path algorithms, the new algorithm works in **stages** which involve finding a set of cheapest augmentations, updating matching and dual variables and continues until an optimal perfect matching is found.

Let us set up some notation. We use U_F, V_F to denote set of free source/sink nodes. We will maintain a set of alternating trees each rooted at a free source node, possibly a trivial tree consisting of a root. This collection will be called **Planted Forest** (PF). Since each isolated node is a trivial tree, the set V_F will be called, alternately, **Trivial Forest** (TF). Moreover, we will have several SFT's containing equal number of source nodes, sink nodes and matched edges. Each such tree is rooted at a sink node of degree

1. The collection of such trees will be called **Matched Forest** (MF). The union of TF and MF will be called as **Floating Forest** (FF). PF, MF and TF will be maintained via a circular list containing roots of the trees in each forest.

Q denotes set of nodes that can be appended to planted forest. It is initially identical with node set of floating forest, but may differ slightly later. For $j \in Q \cap V$, $\pi(j)$ holds the minimum reduced cost of the edges whose tail lies in Planted Forest and whose head is j and tail of such an edge is stored in $nb(j)$. In order to facilitate multiple augmentations, we carry the field **sroot** (source root), which identifies for each node in the planted forest, root of the alternating tree which contains that node. We also carry two scalars **labeled** and **augmented** which counts number of nodes in V_F which are reached and which will be subjected to augmentation. $\mathcal{T}$ denotes any tree, and $\mathcal{T}_i$ denotes the tree containing the node i. y denotes the cumulative dual vector and π denotes dual vector for the shortest path problem.

The Algorithm A7
Input: $(y,\ M)$, PF, FF
global: $y,\ \pi,\ \epsilon$, sroot, augmented, labeled
while $V_F \neq \emptyset$ **do** begin /* shortest path */
  Initialize_shortest_path
  repeat /* solve shortest path */
    $k = \text{argmin}\{\pi(i) : i \in Q \cap V\}$ /* **findmin** */
    $\epsilon = \pi(k),\ u = nb(k),\ \ell = |sroot(u)|$
    If $k \in V_F$ then /* k is free sink node */
      Count_labeled_and_augmented
    else /* k is not free */
    Grow_Tree
  until labeled and/or augmented is large enough
    Augment
    Extend_Dual
    $y \leftarrow y + \pi$
endwhile

Count_labeled_and_augmented $(k,\ u,\ \ell)$
  $Q \leftarrow Q \setminus k,\ \ labeled \leftarrow labeled + 1$
  If $sroot(\ell) = \ell$ then /* augmentation */
    $sroot(k) = \ell,\ p(k) = u$

```
    sroot(ℓ) = −ℓ /* mark ℓ as used */
    augmented ← augmented + 1
  endif
endCount_labeled_and_augmented
```

```
Grow_Tree (k, u, ℓ)
  If p(k) = 0 then delete T_k from MF
  else
    r = p(k)
    delete k among children of r
  endif
  parent(k) = u
  for j ∈ N(T_k) do
    π(j) = ϵ,  Q ← Q − j
  if j ∈ U then Scan(j), sroot(j) = ℓ,
endGrow_Tree
```

```
Initialize_shortest_path
  labeled ← 0, augmented ← 0
  Q ← N(FF)
  for j ∈ Q, π(j) = +∞, nb(j) = 0,
  for T ∈ PF
  for j ∈ T π(j) = 0,
  if j ∈ U scan(u),
  endfor
endInitialize_shortest_path
```

```
  Augment
for v ∈ V_F do
  if sroot(v) ≠ 0 then
    remove v from V_F
    reroot T_v making v root
    transfer T_v from PF to MF
    endif
endfor
endAugment

  Extend_Dual
begin
  for j ∈ Q
    π(j) = ϵ        /* the last ϵ */
  endfor
endExtend_Dual

  Scan(i)        /* i ∈ U */
  for (i, j) ∈ E and j ∈ Q  do
    temp = w_ij − y_j + y_i + π(i)
    if temp < π(j) then
      π(j) = temp, nb(j) = i
    endif
  endfor
endScan
```

Augment does not actually perform augmentations, but instead reroots the alternating trees on the free sink node causing augmentation. Thus each such tree becomes a SFT and transferred to MF. Matching is defined via *parent* pointers of the source nodes. **Initialize_shortest_path** calculates $\pi(j)$, $nb(j)$, for $j \in V \cap FF$ by scanning source nodes in planted forest. Clearly, there is some freedom in ending a stage: from labeling a free sink node to labeling all free sink nodes.

A Faster Version of Hong-Rom Algorithm

We now apply the ideas presented in the above algorithm to semi-assignment algorithms of Dinic-Kronrod and Hung-Rom [39, 52].

Our **Initialize** routine is the classical row minimum routine followed by a slight variation of column minimum applied to free sink nodes. At initialization we allow formation of stars rooted at source nodes as well as at sink nodes. Our initial forest $\mathcal{F}$ decomposes into 3 parts: $\mathcal{F}_-$, $\mathcal{F}_0$, $\mathcal{F}_+$ where $\mathcal{F}_0$ is a collection of matched edges, $\mathcal{F}_-$ is a collection of stars rooted at sink nodes, and $\mathcal{F}_+$ is a collection of stars rooted at source nodes. Each star in $\mathcal{F}_-$ has deficit of sink nodes and each star in $\mathcal{F}_+$ has surplus of sink nodes. Initially the root of a star in $\mathcal{F}_- \cup \mathcal{F}_+$ has degree ≥ 2. When degree of such a root decreases to 1, the tree rooted at that node is transferred directly into forest containing $\mathcal{F}_0$. We let

$$N(\mathcal{F}_-) = (U_-,\ V_-) \text{ and } N(\mathcal{F}_+) = (U_+,\ V_+). \tag{16}$$

When we grow forest and perform augmentations, structures of $\mathcal{F}_-$, $\mathcal{F}_0$, $\mathcal{F}_+$ will change and identities in (16) will be maintained at the beginning of each stage. $\mathcal{F}_-$ will become the **Planted Forest** (PF), a collection of trees each rooted at node in V_-. When nodes in V_- are deleted, resulting collection of trees will be *alternating trees* rooted at nodes in U_-. We would like to view PF as a collection of alternating trees each rooted at a node in U_- (which is true for PF$\backslash V_-$). $\mathcal{F}_0$ wil become a collection of SFT's rooted at sink nodes. This forest will be called **Matched Forest** (MF). $\mathcal{F}_+$ will be called **Surplus Forest** (SF) and will be treated as a collection of stars each rooted at a node in U_+, except some isolated sink nodes (SF will replace TF the trivial forest). Only isolated nodes in the current forest could be in V_+ which is in SF. The union of MF and SF will be called **Floating Forest** (FF).

Q, y, π, nb, *sroot, labeled and augmented* will be the same as before. Actually, the *main* routine will be the same. Only routines *Grow_Tree, Count_labeled_and_augmented* and *Augment* will change slightly to handle stars.

The main operation in a primal-dual/successive shortest path algorithm is *findmin* followed by *Grow_Tree* or *Augment*. Our **findmin** is

$$e = (u, k) = \text{argmin}\{\bar{w}_{ij} \ :\ i \in PF,\ j \in Q\}\,. \tag{17}$$

Normally, for k as defined by (17), $k \in SF$ means an augmentation. However, this is no longer true in our algorithm since we are allowing multiple augmentations. We check whether root ℓ of the subtree containing u, with $\ell = |sroot(u)|$, is marked for

augmentation. If ℓ is marked before, we remove k from Q and increase *labeled* by 1. Node k is temporarily taken from SF but kept in V_+ as an isolated node for later stages. Otherwise, we mark ℓ as augmented via $sroot(\ell) = -\ell$ and delete edges (l, s) and (k, q) where $s = p(\ell)$, $q = p(k)$ from $\mathcal{F}_-$ and $\mathcal{F}_+$ respectively. Furthermore, we check degrees of s and q. If $d(q) = 1$, we move the tree consisting of (q, r) to MF with root r, where r is the only child of q after deletion of k. If the new degree of s is 1, we mark u' the only neighbour of s in $\mathcal{F}$ as augmented, but keep in PF. At the end of augmentation we move such trees from PF to MF. We also increase *labeled* and *augmented* by 1. This is what **Count_labeled_and_augmented** does.

If $k \notin SF$ we call **Grow_Tree**. If k is not root of a SFT, we delete $(k, p(k))$. Then we append SFT rooted at k to PF, and for each $i \in N(\vec{T}_k)$ we set $\pi(i) = \pi(k)$, delete i from Q, and perform **Scan**(i) if i is a source node. Details are given in [11].

Paparrizos [67] developed a pivotal algorithm which he calls 'exterior point' algorithm. The algorithm as presented attains primal feasibility and dual feasibly only at optimality. The selection of pivot edge (co-tree edge) is done in the spirit of dual simplex algorithm, and cut-edge is selected from the fundamental cycle using signature guided considerations. It can be made dual feasible quite easily via lemma 4. More importantly, it can be realized as a variant of the above algorithm.

Achatz, Kleinschmidt and Paparrizos [1] presented another dual simplex based forest algorithm where pivot selection is guided by signature of a SFT. It is very similar in principle to our algorithm A7 and can be made dual feasible via lemma 4.

9 A Few Other Algorithms

In this section we will discuss some old and some new algorithms with different motivations and different characteristics.

Balinski-Gomory Primal Algorithm

We start with the primal algorithm of Balinski-Gomory [17]. It maintains a matching M and a (non-basic) dual vector y and the *invariant* that they are *complementary* throughout the algorithm. y and M are complementary if $e \in M \Longrightarrow \bar{w}_e(y) = 0$. Let us define $E_+ = E_+(y) = \{e \in E : \bar{w}_e(y) > 0\}$, and similarly E_- and E_o. Clearly $E_o(y) = E(y)$ in our terminology. A second *invariant* of the algorithm is that as y is updated to y', we have

$E_+(y) \cup E_o(y) \subset E_+(y') \cup E_o(y')$; in other words once an edge is dual feasible, it remains dual feasible throughout the algorithm. If $E_-(y) = \emptyset$ then the pair (y, M) is optimal. A *stage* chooses $e_o = (u, v) \in E_-$, and ends with new y and M with $e_o \in E_+ \cup E_o$. Since $e_o \in E_-$, $e_o \notin M$. Let $(r, v) \in M$. The algorithm grows an alternating tree rooted at r in the graph $(N, E_+ \cup E_o)$ (with edges in M reversed in orientation). As usual it performs grow_tree and dual-update operations and changes the matching if the edge e_o is on a negative alternating cycle.

An equivalent description of the algorithm for a stage is the following. Let $\epsilon_o = \bar{w}_{uv}$ at the beginning of the stage. Solve the single source shortest path problem for root r over the edges $E_+(y) \cup E_o(y)$ with node v deleted and edges in $M \setminus (r, v)$ reversed in orientation. Let G' be the resulting graph, T be resulting shortest path tree, and π be the dual vector for SP. To describe the new y and M we need to look at 3 cases:

i) $u \notin T$, i.e. u is not reached, which means $\delta^+_{G'}(T) = \emptyset$. $\delta^+(T) \cap E_+(y) = \emptyset$ is obvious. $\delta^+ \cap E_o(y) = \emptyset$ follows from the facts that T is alternating and every node in G' except r is matched. Define

$$y'_i = \begin{cases} y_i + \pi_i & i \in T \\ y_i + \epsilon & i \notin T,\ i \neq v \\ y_i & i = v \end{cases} \tag{18}$$

where ϵ is the last label. Clearly $T \subset E_o(y')$, and $E_+(y') \supset E_+(y)$. Let $\epsilon' = -\bar{w}_{uv}(y') \leq \epsilon_o$. If $\epsilon' \leq 0$ then $e_o \notin E_-(y')$ and we are done. Otherwise, let

$$\hat{y}_i = \begin{cases} y'_i - \epsilon' & i = v,\ i \in T \\ y'_i & \text{otherwise} \end{cases}.$$

Notice that for $e \in \delta^-(T+v)$, $\bar{w}_e(\hat{y}) = \bar{w}_e(y') + \epsilon'$, and in particular $\bar{w}_{uv}(\hat{y}) = 0$. Since $\delta^+_{G'}(T) = \emptyset$, we have the invariant $E_-(\hat{y}) \subset E_-(y')$. Then $(\hat{y}, M)$ is complementary pair and $e_o = (u, v) \notin E_-$, i.e. the stage is over.

ii) $u \in T$ and $\pi(u) \geq \epsilon_o$. Let y' be as in (18). Then $\bar{w}_{uv}(y') = \bar{w}_{uv}(y) + \pi(u) \geq 0$, and $y'_r = y_r$, $y'_v = y_v$ implies $(r, v) \in E_o(y')$. Thus (y', M) is a complementary pair and $e_o = (u, v) \notin E_-(y')$.

iii) $u \in T$ and $\pi(u) < \epsilon_o$. In this case, the path P from r to u in T and the edges (r, v), (u, v) form a negative alternating cycle C. Let M' be the matching obtained from M by switching matching and free edges in C, i.e., $M' = M \oplus C$. Let y' as in (18) and define

$$\tilde{y}_i = \begin{cases} y'_i & i \neq v \\ y'_i - \epsilon_o + \pi(u) & i = v \end{cases}.$$

Clearly, $\bar{w}_{uv}(\tilde{y}) = 0$, $(\tilde{y},\ M')$ is compatible and the stage is over.

By choosing edge (u, v) as $\text{argmin}\{\bar{w}_{iv} : i \in U\}$ and $(u, v) \in E_-$ we can bound the number of stage by n. This algorithm is generalized to non-bipartite matching by Cunningham and Marsh [31]. Balinski-Gomory also presented a sequential version of their algorithm.

Cost Operator Algorithms

Srinivasan and Thompson [76] presented the so called 'cell cost operator' and 'area cost operator' algorithms for the assignment problem which require $O(n^2)$ pivots. We conjecture that with proper design they can be implemented in $O(n^3)$ time. The algorithms maintain a (non-basic) dual feasible y and SFT which defines the matching. They came to the notion of SFT in an attempt to handle degeneracy, and of course they do not use the term SFT. Let us define $E_+(y) = \{e \in T : \bar{w}_e(y) > 0,\ x_e > 0\}$. Since they introduce a perturbation so that for $e \in T$ we have $x_e > 0$, the latter condition is not needed in the definition of $E_+(y)$.

The **cell cost operator algorithm** can be described as:

while $E_+(y) \neq \emptyset$ **do**
  $e_o \in E_+(y)$, $e_o = (p, q)$,
  while $e_o \in E_+(y)$ **do** /* stage */
    Let X be the component of $T - e_o$ containing p
    $e = \text{argmin}\{\bar{w}_j : j \in \delta^+(X)\}$, $\epsilon = \bar{w}_e(y)$
    if $\epsilon = \bar{w}_{e_o}(y)$ **then**
      Update y via (9); **break**
    else
      Update x, y and T via (12), (9)
  endwhile
endwhile

Since y is unrelated to T, the algorithm is not a pivotal algorithm in LP sense; though each pivot looks as a combination of primal pivot and semi-dual pivot. A similar pivot appears in Paparrizos [67]. By using the classical perturbation technique (14), it is easy to see that each stage requires $O(n)$ pivots, since $e_o \in C^-(T, e)$ for each e in a stage. Thus the total number of pivots is $O(n^2)$. Actually, one can work with the combinatorial definition of SFT and using essentially the algorithm of [6], one can show that a stage requires at most $|X_o \cap U|$ pivots, where X_o is the X set at the beginning of the stage.

Area cost operator algorithm inherits some of the notions of cell cost operator algorithm, and introduces some more. Let y, T, $E_+(y)$ as before. Let χ be the characteristic vector of E_+, and π be a basic solution of

$$\pi_j - \pi_i = \chi_{ij}\,, \quad (i,j) \in T\,. \tag{19}$$

Let $\tilde{E} = \{\,(i,j) \in E \,:\, \chi_{ij} < \pi_j - \pi_i\,\}$ and define

$$\mu = \min\left\{ \frac{\bar{w}_e(y)}{\bar{\chi}_e(\pi)} \,:\, e \in \tilde{E} \right\}\,. \tag{20}$$

Let $e = e(\mu)$ be the edge defining μ, if any. Letting $\pi_r = 0$ and T be a SFT rooted at a source node, π_i is equal to the algebraic sum of the edges of $E_+(y)$ on the path P from r to i: each non-degenerate edge in $E_+(y) \cap P$ contributes 1, and each degenerate edge in $E_+(y) \cap P$ contributes -1. Since for $e \in T$, $\bar{\chi}_e(\pi) = 0$, and $\chi_e = 0$, for $e \in T^\perp$; $\tilde{E}$ can be described alternately as $\tilde{E} = \{\,(i,j) \in T^\perp \,:\, \pi_i < \pi_j\}$.

The algorithm can be described as:

while $E_+(y) \neq \emptyset$ **do**
  Compute μ and $e(\mu)$ via (20)
  $\mu' = \min\{\bar{w}_j \,:\, j \in E_+(y)\}$
  $\epsilon = \min\{\mu,\, \mu'\}$
  $y \leftarrow y + \epsilon\pi$
  if $e(\mu)$ **defined** Pivot via (12) using $e(\mu)$
endwhile

Letting $y' = y + \epsilon\pi$, we have for $e \in E_+(y)$, $\bar{w}_e(y') = \bar{w}_e(y) - \epsilon$. Thus, when $\mu = \infty$, we have $\epsilon = \mu'$, and the edge defining μ' will leave $E_+(y)$. By working with the classical perturbation (14), each pivot with $\epsilon = \mu'$ will decrease the sum $\sum\{x_j \,:\, j \in E_+(y)\,\}$ by $\theta \geq \frac{1}{2n}$. Thus the total number of pivots can be bounded by $O(n^2)$.

Again, y is non-basic and we do not see any stage concept. For details see [76] and the references given there.

A Criss-Cross Algorithm

Criss-Cross method of Terlaky[78] performs 'primal' and 'dual' pivots with a well-defined pivot selection rule, but does not maintain primal or dual-feasibility. The Criss-Cross algorithm of [78] stops with primal-infeasibility, dual-infeasibility or finite-optimum for the given linear program. A pivot in a network-simplex method performs 'link' and 'cut' operations, which changes a tree to a new one.

Akgül-Ekin [12] present a new algorithm for AP, termed *Criss-Cross*, which performs essentially primal pivots and dual pivots. Like many algorithms, it works in stages, at the end of which a measure towards optimality is improved. In the basic model, it start with a specially constructed strongly feasible tree. At the end of stage i, primal strong feasibility is regained and dual-feasibility of the set of edges incident with first i source nodes is obtained. In a non-trivial stage, the algorithm first destroys the tree by cutting an edge and linking another one and gain dual-feasibility for the desired set of edges. This part of the algorithm is termed as the **dual-phase**. Then a strongly feasible tree is obtained by performing essentially dual-simplex pivots. This part of the algorithm is called a **primal-phase**. Later, a modified algorithm is given which is faster, and can start with any strongly feasible tree.

The algorithm has the pivot bound of $O(n^2)$, and time bound of $O(n^3)$ for dense graphs and $O(n^2 \log n + nm)$ for sparse graphs using Fibonacci heaps of Fredman and Tarjan[44].

In [79], Thompson presented a *Recursive* algorithm which may look similar to criss-cross [12] at the first sight. His subproblems are similar to that of the basic model in [12]. However, inner working of the algorithm is quite different: in [79], first an optimal primal solution is obtained, then strong feasibility of the basis tree is restored with degenerate pivots. The complexity of both algorithms is the same.

10 Implementation

We now discuss briefly some implementation details. To maintain a tree we use *parent* (p(.)), *first (child), left(sibling)* and *right (sibling)* pointers for each node. Children of a node are maintained as a doubly linked circular list using left and right sibling pointers. Since a doubly linked circular list allows addition and deletion of an element in O(1) time, each *cut, link* can be performed in constant time. Thus all tree operations in a *stage* takes O(n) time. All dual updates in a stage also takes O(n) time. We maintain roots of trees in Planted Forest, MF, TF and SF in a doubly linked circular list. Q is maintained in two ways simultaneously: once as a doubly linked circular list and once as a 0–1 array. For dense graphs we maintain π in an array and for sparse graphs in a fibonacci heap. The array *mate* is not needed since that information is available in *parent* and *first* fields. For more information about data structures see, e.g. [77, 44, 3, 4, 13, 35].

References

[1] H. Achatz, P. Kleinschmidt, and K. Paparrizos, *A dual forest algorithm for the assignment problem*, Tech. Rep., Universität Passau, Germany, 1989.

[2] R.K. Ahuja and J.B. Orlin, *Improved primal simplex algorithms for shortest path, assignment and minimum cost flow problems*, Working Paper OR 189-88, M. I. T. 1988

[3] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows*, in, **Optimization**, Handbooks in Operations Research and Management Science Vol. 1, Eds., G.L. Nemhauser, A.H.G. Rinnoy Kan, and M.J. Todd, (Nort-Holland, New York, 1989).

[4] R.K. Ahuja, K. Mehlhorn, J.B. Orlin, and R.E. Tarjan, *Faster Algorithms for the Shortest Path Problem*, Journal of ACM **37** (1990) 213–223.

[5] M. Akgül, *Topics in Relaxation and Ellipsoidal Methods*, (Pitman, London, 1984. Research Notes in Mathematics Vol. 97.)

[6] M. Akgül, *A genuinely polynomial primal simplex algorithm for the assignment problem*, Technical Report, North Carolina State University, 1985, and Working Paper IEOR 87-07, Bilkent University, 1987. (To appear in *Discrete Applied Mathematics*)

[7] M. Akgül, *A sequential dual simplex algorithm for the linear assignment problem*, Operations Research Letters **7** (1988) 155–158.

[8] M. Akgül, *Shortest paths and the simplex method*, Working Paper IEOR-8804, Bilkent University, 1988.

[9] M. Akgül and O. Ekin, *A dual feasible forest algorithm for the assignment problem*, Working Paper IEOR 90-11, Bilkent University, 1990. (to appear in RAIRO)

[10] M. Akgül , *A forest primal-dual algorithm for the assignment problem*, Working Paper IEOR 90-14, Bilkent University, 1990.

[11] M. Akgül , *A faster version of Hung-Rom algorithm for the assignment problem*, Working Paper IEOR 90-16, Bilkent University, 1990.

[12] M. Akgül and O. Ekin, *A criss-cross algorithm for the assignment problem*, Working Paper IEOR 90-22, Bilkent University, 1990.

[13] A.I. Ali, R.V. Helgason, J.L. Kennington, and H.S. Lall, *Primal simplex network codes: state-of-the-art implementation technology*, Networks **8** (1978) 315–339.

[14] J. Araoz and J. Edmonds, *A case of non-convergent dual changes in assignment problems*, Discrete Applied Mathematics **11** (1985) 95–102

[15] M.L. Balinski, *Signature method for the assignment problem*, Operations Research **33** (1985) 527–536. Presented at Mathematical Programming Symposium, Bonn 1982.

[16] M.L. Balinski, *A competitive (dual) simplex method for the assignment problem*, Mathematical Programming **34** (1986) 125–141.

[17] M.L. Balinski and R.E. Gomory, *A primal method for the assignment and transportation problems*, Management Science **10** (1964) 578–598.

[18] R.S. Barr, F. Glover, and D. Klingman, *The alternating basis algorithm for assignment problems*, Mathematical Programming **13** (1977) 1–13.

[19] D. Bertsekas, *A new algorithm for the assignment problem*, Mathematical Programming **21** (1981) 152–157.

[20] D. Bertsekas, P.A. Hosein, and P. Tseng, *Relaxation methods for network flow problems*, SIAM J. Control & Optimization **25** (1987) 1219–1243.

[21] D. Bertsekas, *The auction algorithm: a distributed relaxation method for the assignment problem*, Annals of Operations Research **14** (1988) 105–123.

[22] R.B. Bixby, *Matroids and operations research*, in Advanced Techniques in the Practice of Operations Research, H.J. Greenberg et. al., ed., (North-Holland, 1982) 333–459.

[23] G.H. Bradley G.G. Brown G.W. Graves, *Design and implementation of large scale primal transshipment algorithms*, Management Science **24** (1977) 1–34.

[24] R.E. Burkard, *Travelling salesman and assignment problems: a survey*, Annals of Discrete Mathematics **4** (1979) 193–215.

[25] R.E. Burkard, W. Hahn, and W. Zimmermann, *An algebraic approach to assignment problems*, Mathematical Programming **12** (1977) 318–327.

[26] G. Carpaneto and P. Toth, *Primal-dual algorithms for the assignment problem*, Discrete Applied Mathematics **18** (1987) 137–153.

[27] P. Carraresi and C. Sodini, *An efficient algorithm for the bipartite matching problem*, EJOR **23** (1986) 86–93.

[28] V. Chvátal, *Linear Programming*, (Freeman, San Francisco, 1983.)

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

[30] W.H. Cunningham, *Theoretical properties of the network simplex method*, Mathematics of Operations Research **4** (1979) 196–208.

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

[32] G.B. Dantzig, *Linear Programming and Extensions*, (Princeton University Press, Princeton, 1963.)

[33] E. Denardo and B. Fox, *Shortest-route methods: 1. reaching, pruning, buckets*, Operations Research **27** (1979) 161–186.

[34] N. Deo and C. Pang, *Shortest path algorithms: taxonomy and annotation*, Networks **14** (1984) 275–323.

[35] U. Derigs, *The shortest augmenting path method for solving assignment problems–motivation and computational experience*, Annals of Operations Research **4** (1985/6) 57–102.

[36] R. Dial, *Algorithm 360: shortest path forest with topological ordering*, Communications of ACM **12** (1965) 632–633.

[37] R. Dial, F. Glover, D. Karney, and D. Klingman, *A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees*, Networks **9** (1979) 215–248.

[38] E.W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik **1** (1959) 269–271.

[39] E.A. Dinic and M.A. Kronrod, *An algorithm for the solution of the assignment problem*, Soviet Mathematics Doklady **10** (1969) 1324–1326.

[40] S. Dreyfus, *An appraisal of some shortest path algorithms*, Operations Research **17** (1969) 396–412.

[41] J. Edmonds and R.M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, Journal of ACM **19** (1972) 248–264.

[42] M. Engquist, *A successive shortest path algorithm for the assignment problem*, Infor **20** (1982) 370–384.

[43] L.R. Ford and D.R. Fulkerson, *Flows in Networks*, (Princeton University Press, Princeton, 1962.)

[44] M.L. Fredman and R.E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, Journal of ACM **34** (1987) 596–615 Also in Proc. 25'th FOCS (1984) 338-346.

[45] J. Gilsinn and C. Witzgall, *A performance comparison of labeling algorithms for calculating shortest path trees*, Tech. Rep. 772, Washington, D.C., 1973.

[46] F. Glover and D. Karney, *Implementation and computational comparisons of primal , dual, primal-dual codes for minimum cost network flow problems*, Networks **4** (1977) 191–212.

[47] F. Glover, R. Glover, and D. Klingman, *Threshold assignment algorithm*, Mathematical Programming Study **25** (1986) 12–37.

[48] D. Goldfarb, *Efficient dual simplex algorithms for the assignment problem*, Mathematical Programming **37** (1985) 187–203.

[49] D. Goldfarb, J. Hao, and S. Kai, *Efficient Shortest Path Simplex Algorithms*, Operations Research **38** (1990) 624–628.

[50] M. Gondran and M. Minoux, *Graphs and Algorithms*, (Wiley, New York, 1984.)

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

[52] M.S. Hung and W.O. Rom, *Solving the assignment problem by relaxation*, Operations Research **27** (1980) 969–982.

[53] E. Johnson, *On shortest paths and sorting*, in Proc. 25'th conference of ACM, Boston, 1972, 529–539.

[54] E.L. Johnson, *Flows in networks*, in Handbook of Operations Research, J.J. Moder & S.E. Elmaghraby, ed., Von Nostrand, 1978.

[55] R. Jonker and A. Volgenant, *Improving the Hungarian assignment algorithm*, Operations Research Letters **58** (1986) 171–176.

[56] R. Jonker and A. Volgenant, *A shortest path algorithm for dense and sparse linear assignment problems*, Computing **38** (1987) 325–340.

[57] P. Kleinschmidt, C. Lee, and H. Schannath, *Transportation problems which can be solved by the use hirsch-paths for the dual problems*, Mathematical Programming **37** (1987), 153–168.

[58] H.W. Kuhn, *The hungarian method for the assignment problem*, Naval Research Logistics Quarterly **2** (1955) 83–97.

[59] J. Munkres, *Algorithms for the assignment and transportation problems*, SIAM Journal **5** (1957) 32–38.

[60] W.M. Nawijn and B. Dorhout, *On the expected number of assignments in reduced matrices for the linear assignment problem*, Operations research Letters **8** (1989) 329–336.

[61] E. Lawler, *Combinatorial Optimization: Networks and matroids*, (Holt, Rinehart and Winston, New York, 1976).

[62] E. Lawler, *Shortest path and network algorithms*, Annals of Discrete Mathematics **4** (1979) 251–265.

[63] J.B. Orlin, *On the simplex algorithm for networks and generalized networks*, Mathematical Programming Study **24** (1985) 166–178.

[64] J.B. Orlin and R.K. Ahuja, *New scaling algorithms for the assignment and minimum cycle mean problems*, Working Paper OR 178-88, M. I. T. 1988.

[65] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms & Complexity*, (Prentice-Hall, Englewood Cliffs, 1982.)

[66] K. Paparrizos, *A non-dual signature method for the assignment problem and a generalization of the dual simplex method for the transportation problem*, RAIRO Operations Research **22** (1988) 269–289.

[67] K. Paparrizos, *An infeasible (exterior point) simplex algorithm for assignment problems*, Mathematical Programming (*to appear*)

[68] K. Paparrizos, *A relaxation Column signature method for assignment problems*, EJOR **50** (1991) 211–219.

[69] K. Paparrizos, *Generalization of a signature method to transportation problems*, Technical Report, Democritus University of Thrace, 1990.

[70] A. Pierce, *Bibliography on algorithms for shortest path, shortest spanning tree and related circuit routing problems*, Networks **5** (1975) 129–143.

[71] R.T. Rockafellar, *Network Flows and Monotropic Optimization*, (Wiley, New York, 1984.)

[72] Roohy-Laleh, *Improvements to the theoretical efficiency of the simplex method*, PhD thesis, University of Charleton, Ottawa, 1981. Dissertation Abstract International vol.43, August 1982 p.448B.

[73] D.D. Sleator and R.E. Tarjan, *A data structure for dynamic trees*, Journal of Computer and System Sciences **26** (1983) 362–391.

[74] V. Srinivasan and G.L. Thompson, *Accelerated algorithms for labeling and relabeling of trees, with applications to distribution problems*, Journal of ACM **19** (1972) 712–726.

[75] V. Srinivasan and G.L. Thompson, *Benefit-cost analysis for coding techniques for the primal transportation problem*, Journal of ACM **20** (1973) 184–213.

[76] V. Srinivasan and G.L. Thompson, *Cost operator algorithms for the transportation problem*, Mathematical Programming **12** (1977) 372–391.

[77] R.E. Tarjan, *Data Structures and Network Algorithms*, (SIAM, Philadelphia, 1983.)

[78] T. Terlaky, *A convergent criss-cross method*, Mathematische Operationsfurchung und Statistics ser. Optimization **16** (1985), 683–690.

[79] G.L. Thompson, *A recursive method for the assignment problems*, Annals of Discrete Mathematics **11** (1981) 319–343.

[80] C. Witzgall, *On labeling algorithms for determining shortest paths in networks*, Tech. Rep. 9840, Washington, D.C., 1968.

[81] N. Tomizawa, *On some techniques useful for solution of transportation network problems*, Networks **1** (1972) 173–194.