TWO NEW ALGORITHMS FOR THE LINEAR ASSIGNMENT PROBLEM

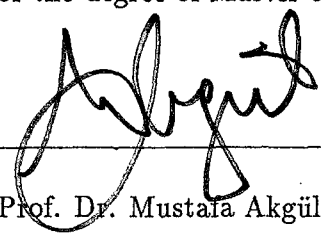
A THESIS
SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Oya Ekin

October 1990

I certify that I have read this thesis and that, in my opinion, it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

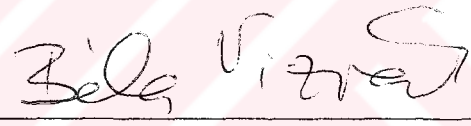


Associate Prof. Dr. Mustafa Akgül (Principal Advisor)

I certify that I have read this thesis and that, in my opinion, it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Associate Prof. Dr. Osman Oğuz

I certify that I have read this thesis and that, in my opinion, it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Associate Prof. Dr. Béla Vizvári

I certify that I have read this thesis and that, in my opinion, it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Associate Prof. Dr. Peter Kas

Approved for the Institute of Engineering and Sciences:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Sciences

To my mother

ABSTRACT

TWO NEW ALGORITHMS FOR THE LINEAR ASSIGNMENT PROBLEM

Oya Ekin
M.S. in Operations Research
Supervisor: Assoc. Prof. Mustafa Akgül
October 1990

The linear assignment problem (AP) being among the first linear programming problems to be studied extensively, is a fundamental problem in combinatorial optimization and network flow theory. AP arises in numerous applications of assigning personnel to jobs, assigning facilities to locations, sequencing jobs, scheduling flights, project planning and a variety of other practical problems in logistics planning. In this thesis work, we seek for new approaches for solving the linear assignment problem. The main concern is to develop solution methods that exhibit some sort of parallelism. We present two new approaches for solving the assignment problem : A dual-feasible signature guided forest algorithm and a criss-cross like algorithm.

Keywords: Assignment problem, signature, stronly feasible tree.

ÖZET

DOĞRUSAL ATAMA PROBLEMİNİN ÇÖZÜMÜNDE İKİ YENİ ALGORİTMA

Oya Ekin
Yöneylem Araştırması Bölümü Yüksek Lisans
Tez Yöneticisi: Doç. Mustafa Akgül
Ekim 1990

Doğrusal atama problemi, birleşi (combinatorial) eniyileme ve serim akım teorisinde en temel problemlerden biri olarak literatürde geniş kapsamda çalışılmıştır. Bu tez çalışmasında, atama problemini çözmek için iki yeni yaklaşım önerilmiştir. Birinci algoritma ikil uyarlılığı sürekli sağlar ve ağaçlar topluluğu ile çalışır. Çizgedeki batak noktaların derece dizimi belli bir özelliğe ulaşınca durulur. İkinci algoritma ise pivotsal bir algoritmadır.

Anahtar Kelimeler: Atama problemi, ikil uyumlu ağaçlar.

ACKNOWLEDGMENTS

I welcome this opportunity to express my gratitude towards Assoc. Prof. Mustafa Akgül for his supervision, guidance, continual interest, encouragement and patience throughout the development of this thesis.

I am also indebted to the members of the thesis committee: Assoc. Prof. Osman Oğuz, Assoc. Prof. Béla Vizvári and Assoc. Prof. Peter Kas for their advice and support.

My special thanks are due to Ezhan Karaşan for his morale support and encouragement in all phases of the study.

Contents

Chapter 1

INTRODUCTION

The linear assignment problem (AP) being among the first linear programming problems to be studied extensively, is a fundamental problem in combinatorial optimization and network flow theory. AP has received a great deal of attention in the literature and justifiably so, since it arises in numerous applications of assigning personnel to jobs, assigning facilities to locations, sequencing jobs, wiring computers, scheduling flights, project planning and a variety of other practical problems in logistics planning. In some of these applications, an AP directly provides a solution to the entire optimization problem, and in others, provides solutions for embedded subproblems on which the optimization of the complete system depends (e.g. the traveling salesman problem). Moreover, historically, the assignment problem has served as a useful test problem for developing instrumental computational ideas for solving more general network optimization problems [9]. A classic example is Kuhn's Hungarian method and its influence on developing the blossom algorithm for solving weighted matching problems in undirected network and on developing the primal-dual method for general linear programs [9].

Many studies of assignment problem methods have been launched over the past three decades. However, as in other areas of network optimization, the most dramatic gains (especially in the design of special list structures and processing techniques) have been made in the last ten years [19].

In this study, we seek for new approaches for solving the linear assignment problem. The main concern is to develop solution techniques that exhibit some sort of parallelism. The two algorithms constructed during this thesis work are capable of finding the optimal solution of a problem instance given the optimal solutions to its underlying independent subproblems. The work presented throughout this study should be viewed as a basic step for a further research in parallel algorithms for the assignment problem.

We now give an overview of the thesis. Following the introduction, we give a review of the related literature in Chapter 2. In this chapter, the emphasis is given to the well

known primal-dual, primal and dual algorithms for the assignment problem together with some others that we use in order to set up the background for the algorithms resulting from this study. The notation used throughout the thesis will also be presented in this chapter. Chapter 3 introduces our first algorithm which is a dual-feasible forest algorithm for the assignment problem. The algorithm, being an improved variant of Paparrizos'[27] algorithm, is guided by the signature of a strongly feasible tree and terminates with such a tree. A criss-cross like algorithm is presented in Chapter 4. As the name implies, at some stages of the algorithm, primal pivots are made to obtain a primal-feasible solution and at others, dual pivots are made to achieve a dual-feasible solution. Thus, the algorithm can not be categorized as primal, dual or primal-dual. In Chapter 5, we provide concluding remarks about parallelism in assignment problems.

Chapter 2

REVIEW OF THE RELATED LITERATURE

Being a fundamental problem in linear programming and network flow theory, the assignment problem has been extensively studied and numerous special algorithms have been developed to solve it. Solution procedures vary from primal-dual [24, 9, 25, 31], dual [2, 5], network flows [17, 16], cost parametric [29], recursive [30], relaxation [15, 22] to primal methods [6]. Assignment problem has been generalized to bottleneck, quadratic and algebraic cases [10, 11].

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 [24] using successive shortest paths [12, 23],
- a relaxation method due to Hung and Rom [22],
- an efficient version of Balinski and Gomory's [6] primal algorithm due to Cunningham and Marsh [14], primal simplex algorithm of Akgül [3],
- signature methods of Balinski [5] and Goldfarb [17],
- the dual forest algorithm of Paparrizos at al. [1],
- Dual-simplex algorithms of Akgül [2] and Balinski [5] and,
- the two algorithms we are going to present in Chapters 3 and 4.

Needless to say, we are going to touch only a few of the mass of AP algorithms that exists in the literature. Those we are going to mention include the well known primal-dual, primal, and dual algorithms together with some others that we use in order to set up the background for algorithms in Chapters 3 and 4.

We categorize the algorithms under four major headings. In Section 2.1, we are going to describe the Hungarian algorithm as a prototype primal-dual algorithm. Primal network simplex algorithms are mentioned in Section 2.2. Section 2.3 is reserved for signature guided algorithms and 2.4 covers some dual-simplex algorithms.

2.1 Primal-Dual Algorithms

In what follows, we are going to describe the 'Hungarian Algorithm' which is a prototype primal-dual algorithm that solves the assignment problem. First we give some preliminaries:

Definition 2.1 *A* **bipartite graph** $G = (V, E)$ *is one whose node set* V *can be partitioned into two sets* R *(row or source nodes) and* C *(column or sink nodes) so that no edge of* G *has both ends in the same set. When* G *is directed, each edge has its tail in* R *and its head in* C*. Often,* G *will be represented as* $G = (R, C, E)$*.*

Definition 2.2 *A* **matching** M *in* G *is a subset of its edges such that no two meet the same node. A* **perfect matching** *in* G *is a subset of its edges such that exactly one member meets each node.*

Definition 2.3 *Given any bipartite graph* $G = (V, E)$ *with real numerical weight* w_e *for each edge* $e \in E$*, the* **optimum assignment problem** *is to find a perfect matching* M *which minimizes* $\sum_{e \in M} w_e$*.*

Consider a directed bipartite graph $G = (R, C, E)$ with $V = R \cup C$ as its node set and E as its edge set. Let the edges be directed from source(row) nodes towards column(sink) nodes.

The optimum assignment problem (weighted bipartite matching problem) can be represented with the following simple algebraic formulation

$$\begin{array}{lll} \min & \sum_{(i,j)\in E} w_{ij}x_{ij} & \\ s.t & -\sum\{x_{ij} : (i,j) \in E\} = -1 & i \in R \\ & \sum\{x_{ij} : (i,j) \in E\} = 1 & j \in C \\ & x_{ij} \geq 0 & (i,j) \in E \end{array}$$

where it is assumed that $\mid R \mid = \mid C \mid = n$.

Here, $x_{ij} = 1$ means that the edge (i, j) is included in the matching, whereas $x_{ij} = 0$ means that it is not.

More compactly, we can write the optimum assignment problem as:

$$\begin{array}{ll} \min & wx \\ & Ax = b \quad (LP) \\ & x \geq 0 \end{array}$$

where A is the node-edge incidence matrix of G. The dual LP is then:

$$\begin{array}{ll} \max & \pi b \qquad (DLP) \\ & \pi A \leq w \end{array}$$

or more explicitly

$$\begin{array}{ll} \max & \sum_i \pi_i b_i \\ & \pi_{h(e)} - \pi_{t(e)} \leq w_e \quad \forall e = (t(e), h(e)) \in E \end{array}$$

The complementary slackness (CS) conditions say that

$$\text{if} \quad x_j > 0 \quad \text{then} \quad \pi a^j = w_j$$

where a^j is the j^{th} column of A and x_j and w_j are the j^{th} components of vectors x and w respectively.

So, if we have an x-vector feasible in LP and a π-vector feasible in DLP satisfying the CS conditions, then x and π are both optimal.

The generic primal-dual algorithms proceed as follows:

Starting with a dual-feasible π, let $E(\pi) = \{e : w_e = \pi_{h(e)} - \pi_{t(e)}\}$. That is to say, $E(\pi)$ is the set of indices of an LP solution vector x, that are allowed to be positive if π is optimal. We search for such an x by solving an auxiliary problem, called the **restricted primal** $LP(\pi)$ determined by the π we are working with. $LP(\pi)$ is

$$\begin{array}{l} Ax = b \\ x_j = 0 \quad \forall j \notin E(\pi) \end{array}$$

or more compactly

$$\begin{array}{l} A^\pi x^\pi = b \\ x^\pi \geq 0 \ . \end{array}$$

If the feasibility problem $LP(\pi)$ is successful i.e. $\bar{x}^\pi$ solves $LP(\pi)$, then $\bar{x} = (\bar{x}^\pi, 0)$ and π are both optimal in LP and DLP respectively.

If $LP(\pi)$ has no solution, then there exists a vector σ solving

$$\begin{array}{l} \sigma a^j \leq 0 \quad j \in E(\pi) \\ \sigma b > 0 \end{array}$$

Let

$$\bar{t} = \min\{\frac{w_j - \pi a^j}{\sigma a^j} : \sigma a^j > 0\}$$

If $\bar{t} = +\infty$, then dual LP is unbounded and hence LP is infeasible. In the case when $\bar{t}$ is finite, increment π with $\bar{t}\sigma$ and continue with the recently obtained dual-feasible π.

In what follows, some information on matching in general and implementation details of the 'Hungarian Algorithm' will be presented.

Any tree of G rooted at say, source node r, can be described by parent pointers. That is to say, if $e = (i, j)$ is an edge between source i and sink j then, $parent(j) = p(j) = i$. Let M be any given matching in G. $Mate$ is an array of size $\mid V \mid$ which stores the adjacency information defined by M:

$$e = (i, j) \in M \Leftrightarrow mate(i) = j \Leftrightarrow mate(j) = i$$

Definition 2.4 *If $mate(i) = 0$, then node i is defined to be* **free** *(exposed) node i.e. node i meets no edge of M. Else, node i is a* **saturated** *node.*

Let $\overline{M} = E \backslash M$ be the set of unmatched edges.

Definition 2.5 *An* **alternating path** *with respect to matching M is one on which edges alternate between matched and unmatched. An* **augmenting path** *is an alternating path joining two exposed vertices. There exists one more edge of $\overline{M}$ than of M in an augmenting path. Let P be the set of edges on an augmenting path in a graph G with respect to matching M. Then, an* **augmentation** *is the replacement of M with $M' = M \oplus P$ where $\oplus$ denotes the symmetric difference operation.*

Theorem 2.1 (Berge) *A matching M in G is not of maximum cardinality if and only if (G, M) contains an augmenting path with respect to M.*

Definition 2.6 *An* **alternating tree**, *T, with respect to matching M, rooted at source node, say r, has the following properties.*

- *r is a free node*
- *$deg(j) = 2$ $\forall j \in C \cap T$ i.e. there exists two edges incident with each sink node in T*
- *$\forall v \in T$ $(v \neq r)$, the unique path from r to v is alternating*
- *All leaf nodes are source nodes*
- *$\mid V(T) \mid = 2k + 1$, $\mid T \cap M \mid = k$, $\mid T \cap R \mid = k + 1$, $\mid T \cap C \mid = k$ for some positive integer $k, k < n$.*

Definition 2.7 *A maximal alternating tree is called a* **Hungarian tree.** *A subset H of nodes in G is called* **Hungarian** *relative to G, if no two of them are joined by an edge of G and if the set of neighbors of H i.e. $N(H)$ has fewer members than H, i.e. $\mid N(H) \mid < \mid H \mid$. So, for a Hungarian tree, $T \cap R$ is Hungarian.*

Theorem 2.2 (Hall) *A directed bipartite graph $G = (R, C, E)$ has a perfect matching if and only if $\forall S \subset R$, $\mid N(S) \mid \geq \mid S \mid$ i.e. if and only if subset R contains no Hungarian set.*

Let

$$\delta^+(S) = \{e \in E(G) : e = (u, v), u \in S, v \notin S\}$$

Then an alternating tree, T, is Hungarian $\Leftrightarrow \delta^+(V(T)) = \emptyset$.

2.1.1 The Hungarian Algorithm

Given any dual-feasible π, let $G(\pi) = (V, E(\pi))$ denote the **equality subgraph** of G relative to π i.e. $G(\pi)$ consists of all nodes of G and those edges $e \in E$ such that $\pi_{h(e)} - \pi_{t(e)} = w_e$ where $e = (h(e), t(e))$. Letting $\bar{w}_e = w_e - \pi_{h(e)} + \pi_{t(e)}$ $\forall e \in E$, $E(\pi) = \{e \in E : \bar{w}_e = 0\}$. M is any given matching in $E(\pi)$ and F is the set of free nodes. *Label* is an array of size $\mid V \mid$ which contains the label information of nodes. Each node can have label '-', '+', or '0'. Root node has label '+' in the alternating tree and all nodes having odd distance from root have labels '-' (odd or inner nodes) and all those having even distance from root have label '+' (even or outer nodes). '0' label means, that specific node is is not currently in the alternating tree.

Let L be the list of edges emanating from the current alternating tree T. Since G is directed, $L = \{e = (u, v) : e \in \delta^+(V(T))$ and u is an even node $\}$.

procedure Aug (r,G,M)
begin
$T \leftarrow \emptyset$
$label(r) \leftarrow$ '+'
$L \leftarrow \delta^+(r)$
$found \leftarrow false$
while $L \neq \emptyset$ **and not** found **do**
begin
take $e = (u, v) \in L$
$L \leftarrow L \backslash e$
if $label(v) =$ '0'
then if $mate(v) = 0$

then begin
augment
$found \leftarrow true$
end
else grow tree
end{while}
end{aug}

Depending on whether node v is free or saturated, two things can happen in **procedure Aug.**

(1) **augment** If v is free, the matching M is augmented, the existing tree T is thrown away, another free source node is chosen and the matching algorithm is continued.

(2)**grow tree** If v is saturated and say $mate(v) = 'w'$, then T is enlarged by adding two edges and two nodes and hence maintaining the alternating structure. The arrays are altered as follows:
$label(w) \leftarrow '+'$
$parent(w) \leftarrow 'v'$
$parent(v) \leftarrow 'u'$
$L \leftarrow L \cup \delta^+(w)$
$T \leftarrow T \cup (u,v) \cup (v,w)$

The prototype weighted matching algorithm given dual-feasible π and feasible M is then:

begin
choose $r \in F \cap R$
Aug$(r, G(\pi), M)$
if found=false
then begin *(make dual-variable changes)*
let $\epsilon = \min\{\bar{w}_e : e \in \delta^+(V(T))\}$
$\pi_v \leftarrow \pi_v - \epsilon \ \forall v \in T$
Aug $(r, G(\pi), M)$
end
end. {Hungarian}

Let us call the computation between two successive augmentations a **stage.** Thus, a stage consists of a search for an augmenting path in the bipartite graph made up of admissible edges ($e \in E(\pi)$), interleaved with dual variable modifications that change the set of admissible edges. To modify the dual variables, we need to calculate ϵ as

mentioned above. To recompute this quantity every time that we modify dual variables needs comparing the n^2 candidates which could be very costly. If the Hungarian algorithm is implemented in a brute way, the cost will be $O(n^4)$. However, any good implementation will solve the shortest path problem on the residual graph $RG(\pi, M)$ (the graph obtained from G by reversing the orientation of edges in M) which will improve the cost to $O(n^3)$.

2.2 (Primal) Network Simplex Algorithms and Notation

In the rest of this Chapter, we are going to stick to the notation used in [3]. We will view the assignment problem (AP) as an instance of a transshipment problem. AP can be represented as a flow problem over a finite directed graph $G = (V, E)$, with $V = R \cup C$, where R is the set of row(source) nodes and C is the set of column(sink) nodes. Each edge $e \in E$ is directed from its tail $t(e) \in R$ to its head $h(e) \in C$, has flow x_e and cost w_e. Clearly $\mid V \mid = 2n$ and let $\mid E \mid = m$.

For graph G and disjoint sets $X, Y \subset V$, we let

$$\gamma(X) = \{e \in E : t(e) \in X, h(e) \in X\},$$

$$G[X] = (X, \gamma(X)) \text{ (the node induced subgraph of } G),$$

$$\delta(X, Y) = \{e \in E : t(e) \in X, h(e) \in Y\},$$

$$\delta^-(Y) = \delta(\overline{Y}, Y),$$

$$\delta^+(Y) = \delta(Y, \overline{Y}) \text{ where } \overline{Y} = V - Y$$

For $v \in V$, $d(v) = d_T(v)$ is the degree of node v in tree T. For a subgraph H of G, $N(H)$ and $E(H)$ represent the node set and the edge set of H. We use $+$ and $-$ to denote set union and set difference, when it is convenient.

The transshipment problem can then be formulated as

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e \\ s.t. \quad & \sum(x_e : v = h(e)) - \sum(x_e : v = t(e)) = b_v, \quad v \in V \\ & x_e \geq 0, e \in E \end{aligned}$$

which can be written more compactly as

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

where A is the node-edge incidence matrix, and

$$b_r = -1, \quad r \in R$$

$$b_c = +1, \quad c \in C$$

The dual of (AP) is

$$\max\{\pi b : \pi_{h(e)} - \pi_{t(e)} \leq w_e, e \in E\}$$

Reduced cost of edge $e = (i, j)$ is

$$\bar{w}_e = \bar{w}_{ij} = c_e - \pi_j + \pi_i$$

The network simplex method is a specialization of the primal simplex method of general linear programming to transshipment problem. It is well-known that any basis of (AP) consists of n positive variables (nondegenerate) and $n - 1$ degenerate variables which together correspond to a spanning tree T of G.

Given any T, the flow values $x_e, e \in T$ can be uniquely determined for each compatible 'supply' vector b i.e. $\sum b_v = 0$ is a necessary condition for (AP) to have a solution. Moreover, the complementary dual basic solution is also uniquely determined once one of the node-potentials $(\pi's)$ is fixed arbitrarily. Optimality conditions for (AP) are primal feasibility, dual feasibility and complementary slackness. The simplex method, while maintaining primal feasibility and complementary slackness, proceeds towards attaining dual feasibility.

Definition 2.8 *For every co-tree edge $e \in \overline{T} = E - T$, $T + e$ contains a unique cycle $C(T, e)$ named as the* **fundamental cycle**. *We can partition $C(T, e)$ into $C^+(T, e)$ containing edges having the same orientation with e and $C^-(T, e)$ consisting of the remaining edges in the fundamental cycle.*

In the classical network simplex method, the leaving edge f is determined by

$$f = \arg\min\{x_j : j \in C^-(T, e)\}$$

If $x_f = \theta$, the new flow values will be

$$x_{j'} = \begin{cases} x_j + \theta & j \in C^+(T, e) \\ x_j - \theta & j \in C^-(T, e) \\ x_j & j \notin C(T, e) \end{cases}$$

$T - f$ has exactly 2 components, say X and $\overline{X} = V - E$, where $t(e) \in X$. The dual variable changes will then be

$$\pi_v = \begin{cases} \pi_v & v \in \overline{X} \\ \pi_v + \epsilon & v \in X \end{cases}$$

where $\epsilon = -\bar{w}_e$ is the amount of dual-infeasibility of edge e.

Definition 2.9 *The set of edges with one end in X and the other in $\overline{X}$ is defined as the* **fundamental cocycle** *which can be partitioned into $D^+(T,f)$ and $D^-(T,f)$ where*

$$D^+(T,f) = \{j \in E : t(j) \in X, h(j) \in \overline{X}\}$$

and

$$D^-(T,f) = D(T,f) - D^+(T,f)$$

Hence, f and e both belong to $D^+(T,f)$

A primal simplex algorithm for the assignment problem proceeds from one feasible basis to another via a selection rule specifying the choice of one dual-infeasible co-tree edge e (nonbasic variable) and the choice of $f \in T$ (basic variable). Primal simplex algorithms differ from each other in the way the entering and leaving variables are chosen.

Provided that cycling does not occur, the classical simplex algorithm is finite and practically efficient. If the linear program is nondegenerate, simplex method is finite. However, instances of (AP) are highly degenerate, especially when b is an integer vector.

Cunningham [13] introduced a simple combinatorial modification to prevent cycling in the primal simplex algorithm for the transshipment problem. The method involves keeping 'strongly feasible' bases, which arise from spanning trees of G providing basic feasible solutions and whose edges satisfy certain additional orientation requirements. Barr et al. [7] independently introduced a specialization of strongly feasible tree to assignment problem in order to circumvent and exploit degeneracy. When restricted to AP, the two definitions in [7] and [13] are dual to each other; root is chosen as a source in [7], and as a sink in [13] and the roles of 'forward' and 'reverse' edges are interchanged.

Using the notation of [7], let r be a specific source node chosen as root and $dis_T(v)$ be the distance of node v from r in T i.e. number of edges in the unique path form r to v.

Definition 2.10 *$e \in T$ is a* **reverse edge** *($e \in$ **Re**) if $dis_T(t(e)) = dis_T(h(e)) + 1$, otherwise it is a* **forward edge** *($e \in F$).*

Definition 2.11 *A rooted tree is* **strongly feasible (SFT)** *if $\forall f \in T$, $x_f = 0$ implies f is a reverse edge.*

Definition 2.12 *Co-tree edge $e \in \bar{T}$ is a* **forward edge** *($e \in$ **F**) if $t(e)$ lies on the unique path from r to $h(e)$ in T. e is a* **reverse edge** *($e \in$ **Re**) if $h(e)$ lies on the path from r to $t(e)$. Otherwise, co-tree edge is called a* **cross edge** *($e \in$ **Cr**).*

Definition 2.13 *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, $e = (u,v) \in \mathbf{F}$ if $u = \mathbf{NCA}(u,v)$, $e \in \mathbf{Re}$ if $v = \mathbf{NCA}(u,v)$ and $e \in \mathbf{Cr}$ otherwise.

When rooted at a source node, the SFT of an assignment problem resembles the alternating tree of the bipartite matching algorithm mentioned in Section 2.1 and the unique path from r to any $v \in V$ is an alternating path.

Lemma 2.1 *The SFT has the following properties:*

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

(ii) every source node except root has degree 2; root has degree 1

(iii) let $e \in \overline{T}, f \in C(T,e)$ and $t(e) = t(f)$. Then, the selection of f as the departing variable is valid and maintains strong feasibility

(iv) for $e \in \overline{T}$, pivot specified by (iii) is nondegenerate $\Leftrightarrow e \in \mathbf{F} \Leftrightarrow f \in T \cap \mathbf{F}$

(v) for e, f same as in (iv), the pivot is nondegenerate $\Leftrightarrow r \in X$ (component of $T - f$ containing $t(e)$).

The alternating basis (AB) algorithm of Barr et al. [7] starts with any feasible AP basis i.e. SFT for the assignment problem. It then successively applies the simplex pivot step keeping the root node fixed and picking the leaving variable according to Lemma 2.1(iii). Cunningham essentially does the same in his simplex algorithm [13].

The simplex algorithm presented by Hung [21] specifies that in selecting the entering arcs, those arcs that are reverse or cross are to be chosen first. An AP-basis (SFT) is 'degenerate pivot free' if every reverse or cross nonbasic arc has a nonnegative reduced cost i.e. is dual feasible. So, Hung's algorithm performs degenerate pivots at an extreme point until a degenerate-pivot-free basis is found and only then moves to a new extreme point. The entering arc(e) selection rule in Hung's algorithm is the "Modified row most negative" (MRMN) rule which is:

Let $e = (i_1, j)$ be the most recent entering arc that belongs to basis T. Select $e^* = (i_1 + 1, j^*)$ as the current entering arc if $\bar{w}_{e^*} < 0$, e^* is either a reverse or a cross arc on T and $\bar{w}_{e^*} = \min_{\bar{e} \in S} \bar{w}_{\bar{e}}$ where $S = \{e = (i_1 + 1, j) \ \forall j \in \mathbf{Cr}$ and e is either reverse or cross on T $\}$. If $\bar{w}_{\bar{e}} \geq 0 \ \forall \bar{e} \in S$, then rows $i_1 + 2, i_1 + 3, ..., n, 2, 3, ..., i_1$ are scanned respectively in a similar manner. The leaving arc f is chosen according to Lemma 2.1(iii) as is done in [7].

In his simplex algorithm, Akgül [3] starts with a problem of size 1 and sequentially solves problems of size 2,3,4,...,n. The algorithm utilizes degeneracy by working with strongly feasible trees and employs Dantzig's rule for entering edges of the subproblems.

The full basis of the original problem is carried throughout the algorithm and this makes it a purely primal simplex algorithm. Instead of evaluating the change in the objective function value, the structure of cutsets i.e. set of nodes on which dual variable changes are made, generated during the solution of the current subproblem is studied. Some facts about these sets are:

- cutsets are disjoint,
- edges emanating from a cutset are dual-feasible once for the subgraph under consideration,
- the tails of dual-infeasible edges undergo no dual variable change
- each node is subject to at most one dual variable change.

Now, we sum up the computational complexities of some primal simplex algorithms for the assignment problem.

Although Barr et al.'s [7] algorithm works well in practice, it is theoretically an exponential algorithm. Roohey-Laleh [28] exhibits a family of problems with exponentially long nondegenerate pivot sequences.

Hung's algorithm [21], being a polynomial primal simplex method requires $O(n^3 \log \Delta)$ pivots, where Δ is the gap in the objective function value between an initial extreme point. Although, Hung's algorithm achieves polynomial convergence, he states that the algorithm is probably less efficient than the primal simplex algorithm of Barr et al.[7].

Orlin [26], using a perturbation technique which is equivalent to using strongly feasible trees and applying Dantzig's rule, reduced the bound on the number of pivots to $O(n^2 \log \Delta)$ which then is reduced to $O(n^2 m \log n)$ again by him.

Cunningham and Roohey-Laleh [28] developed a genuinely polynomial primal simplex algorithm that needs $O(n^3)$ pivots and $O(n^5)$ time in the worst case.

The mentioned algorithm of Akgül [3] solves the assignment problem in $\frac{1}{2}n(n+3)-4$ pivots. The algorithm can be implemented to run in $O(n^3)$ time for dense graphs and $O(n^2 \log n + nm)$ time for sparse graphs using state of the art data structures.

2.3 Signature Guided Algorithms

Following the notation presented in Section 2.2, we now give an overview of signature guided algorithms. Signature methods were first introduced by Balinski [4]. We discuss his algorithm together with efficient variants of it introduced by Goldfarb [20]

Definition 2.14 *The* **signature** *of a tree T is the vector of its column(row) node degrees* $d = (d_1, ..., d_n)$, $\sum_i d_i = 2n - 1$, $d_i \geq 1$.

Lemma 2.2 *Let T be a spanning tree for the AP rooted at a sink node. Then, the following are equivalent.*

- *T is a SFT*
- *T is primal feasible. In particular,*

$$x_e = 1, e \in T \Leftrightarrow e \in \mathbf{Re}$$

and

$$x_e = 0, e \in T \Leftrightarrow e \in \mathbf{F}$$

- *Column signature of T is* $(2, 2, 2, ..., 1)$.

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., $(2, 2, 2, ..., 2, 1)$.

Definition 2.15 *A tree is in* **level** *k if its signature has exactly k ones.*

Balinski's algorithm begins with a tree at 'level $n - 1$' and ends when 'level 1' is reached. The method is entirely guided by the signatures.

The initial dual-feasible tree T (Balinski tree) is constructed as follows:

$$(1, j) \in T \ \forall j \in C$$

$$\pi_i = \begin{cases} 0 & \text{if } i \in R \cap \{1\} \\ w_{1i} & \text{if } i \in C \end{cases}$$

$$\pi_i = \min_k (w_{1k} - w_{ik}) \quad 1 \neq i \in R$$

and

$$(i, j) \in T \text{ for } j = \arg\min_k (w_{1k} - w_{ik}).$$

During the course of the algorithm, the first level k tree T encountered has by construction the row signature $(k + 1, 2, ..., 2, 1, ..., 1)$: source node 1 has degree $k + 1$, some k row nodes have degree 1 (leaf nodes) and the remaining $n - k - 1$ row nodes have degree 2. One of the leaf nodes, say t, is singled out as the 'target'. During the pivot, the leaving edge f is chosen as edge $(1, l)$ which is on the unique path from root (source node 1) to the target t. The entering edge e is chosen so as to maintain the dual-feasibility. If

we let T^l be the component of T containing t after the removal of f, e is chosen as the edge having minimum reduced cost among edges in $\delta^+(T^l)$. The dual variable changes are done accordingly i.e.

$$\pi_v \leftarrow \pi_v - \epsilon \ \forall v \in T^l$$

where ϵ is the reduced cost of edge e.

Let $e = (g, h)$.

(i) If g was a leaf node in T, level $k-1$ is achieved, and the **basic step** is complete

(ii) If this is not the case (g has degree 2 in T), take g as the source node in the new tree (tree after the pivot) and repeat, pivoting on (g, l') on the path joining g to t.

Case (ii) can occur at most $n-k-1$ times before a case (i) occurs, so the method encounters at most $n-k$ 'level k' trees giving rise to a pivot bound of $\frac{(n-1)(n-2)}{2}$.

This variant of signature method is 'purely' dual: it pays no attention to the primal problem. Thus, the method is not a simplex method in that a pivot can be made by deleting an edge (i,j) with $x_{ij} = 0$. However, there is a cost to ignoring the primal problem: prior to its termination, the method may encounter a dual-feasible basis that already admits an optimal assignment without noticing it. As Balinski states [4], this difficulty is easily overcome with some additional accounting.

The obvious way of obtaining a worst case count is to notice that each pivot step involves at most $O(n^2)$ operations (comparisons to find $\epsilon = \bar{w}_e$). This count yields $O(n^4)$ total work for the signature method.

Balinski [5] and Goldfarb [20] show how each basic signature step can be accomplished using only $O(n^2)$ operations. Since Balinski's method require only $n-2$ basic steps, this results in an $O(n^3)$ dual algorithm.

The key to the efficiency of Goldfarb's variants of the signature method is the use of labels s_i on all row nodes in T^l. These enable the algorithms to use information already computed on previous pivots during the current basic signature step.

To be specific, the computation of

$$\bar{w}_e = \min\{\bar{w}_{ij} : (i,j) \in \delta^+(T^l)\} = \epsilon$$

that is required to determine the arc e to enter the basis on a simplex type pivot can be performed as

$$\epsilon = \min_{i \in T^l \cap R} \{s_i\} = s_g$$

where

$$s_i = \min_{k \in T^l \cap C} \{\bar{w}_{ik} : i \in T^l \cap R\}$$

and $\overline{T}^l = T - T^l$.

The column index which gives the minimum for each $i \in T^l \cap R$ is also needed so, let

$$nb(i) = j \ \ if \ \ \bar{w}_{ij} = s_i$$

Let Q be the set of row nodes in T^l and Q' the corresponding set at the next tree, P the set of column nodes in $\overline{T}^l$, and P' the corresponding set at the next tree. By construction, $Q' \subset Q$ and $P \subset P'$.

After a pivot,

$$s_i \leftarrow \min\{s_i - \epsilon, \min_{j \in P'-P}\{\bar{w}_{ij}\}\}, \quad i \in Q'$$

Consequently, no matter how many pivots are carried out during a basic signature step, each reduced cost $\bar{w}_{ij}$ 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, each level step can be accomplished using only $O(n^2)$ operations.

Goldfarb [20] further presents a signature algorithm whose worst-case computational bound is slightly better than the bound for his variant described 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 with basis tree T having exactly one terminal row node, T is first extended to the $(k \times 1) \times (k \times 1)$ problem in such a way as to maintain dual feasibility. If necessary, one basic signature step is executed to reduce the number of leaf row nodes from two down 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.

2.4 Dual Simplex Algorithms guided by Dual Strongly Feasible Trees

In this section, we are going to present Balinski's [5] purely dual-simplex algorithm together with Akgül's sequential dual-simplex algorithm. We stick to notation used in [2] together with that described in Section 2.2.

The dual simplex method for the transshipment problem starts with a dual-feasible tree. If $x_f \geq 0 \ \ \forall f \in E(T)$, then T is optimal (primal feasibility, dual feasibility and CS are satisfied). Otherwise, the algorithm chooses an $f \in E(T)$ with $x_f < 0$, as the leaving edge, and chooses a co-tree edge as the pivot (entering) edge to satisfy the dual feasibility

via

$$\epsilon = \bar{w}_e = \min\{\bar{w}_{ij} : (i,j) \in \delta^-(X)\}$$

where X is the component of $T - f$ containing $t(f)$. 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, increase the reduced costs of the edges in $\delta^+(X)$ by ϵ, and decrease that of edges in $\delta^-(X)$ by ϵ.

Since a tree has one less edge than the number of nodes, there is a natural $1-1$ correspondence between $N(T) - r$ and $E(T)$, i.e., between the non-root nodes and edges of the tree. When T is reoriented as a branching $\vec{T}$ with root r (all edges are directed away from r in $\vec{T}$), the mapping is :

$$g : N(T) - r \rightarrow E(T), \;\; g(v) = (p(v), v),$$

where $p(v)$ is the parent of node $v \in N(T), v \neq r$. Let $L \subset V - r$ be the set of leaf nodes of T, $\tilde{L} = \{v \in L : (v,r) \in E(t)\}$, $E(L) = \delta^-(L)$, $E(\tilde{L}) = \delta(L,r)$ and $L_R = L \cap R$.

Definition 2.16 *A dual feasible tree is a* **Dual Strongly Feasible Tree (DSFT)** *for AP if*

(i) $f \in \mathbf{Re} - E(\tilde{L}) \Rightarrow x_f \leq 0$

(ii) $f \in \mathbf{F} \Rightarrow x_f \geq 1$

Note that automatically we have $f \in E(L) \Rightarrow x_f = 1$.

A tree which is both a SFT and a DSFT is optimal (with possibly different roots). Akgül's definition is slightly different from that of Balinski: the roles of forward and reverse edges are interchanged and T is rooted at a sink node.

Balinski [5] proved the following Lemma.

Lemma 2.3 *Let T be a DSFT rooted at a sink node r. Consider $u \in R$, (hence $g(u) \in$ **Re**), with $d(u) \geq 3$. Then,*

(i) $x_{g(u)} \leq -1$

(ii) the selection of $g(u)$ as the leaving edge maintains DSFT.

In other words, if the selection of leaving edges is restricted to those $f \in T \cap R$ and $d(t(f)) \geq 3$, dual strongly feasibility of T will be maintained.

Balinski's dual simplex algorithm [5] works in **stages**. Let $S = \{u \in R : d(u) \geq 3\}$. The algorithm for a **stage** (a signature step or a level), for $s \in S$ can be described as:

procedure A1 (s)
while $d(s) \geq 3$ do
  begin
    cut $f = g(s)$, and let $e \in \bar{T}$ be the entering edge via
      $\epsilon = \bar{w}_e = \min\{\bar{w}_{ij} : (i,j) \in \delta^-(X)\}$
    $T \leftarrow T + e - f$
    $s \leftarrow t(e)$
  end while

In a stage, the algorithm starts with $s \in S$ and performs dual-simplex pivots until it reaches a node in L_R. Since X's are monotonically increasing, the number of pivots in a stage is bounded by $\mid R - L_R \mid$. When $S = \emptyset$ or $\mid L_R \mid = 1$, T is optimal (since SFT). Clearly, the total number of pivots is bounded by $\frac{1}{2}(n-1)(n-2)$ and this bound is shown to be sharp [5].

We now describe Akgül's algorithm.

He solves a sequence of perturbed AP's over an increasing sequence of graphs $G_0, G_1, ..., G_n$. Each G_k defines an AP, say AP_k. Let $V = \{v_1, ..., v_n\}$ be an arbitrary ordering of sink nodes, $r \equiv v_0$ be a dummy sink node, and let $G^{\#} \equiv (R, V + r, E + \{(u,r) : u \in R\})$. When the original graph $G = (R, C, E)$ is complete bipartite graph, then so is $G^{\#}$. G_k is defined as $G^{\#}[R + \{v_0, v_1, ..., v_k\}]$. Actually, AP_k is not strictly an assignment problem since $b_u = -1, \quad u \in R, \quad b_{v_j} = 1, \quad j \in [1..k], \quad b_r = n - k$. For artificial edges, set $w_{ur} = K$, for arbitrarily large K. Let $\pi_r = K, \quad \pi_u = 0, \quad u \in R$ for AP_0. Clearly, G_0 is a feasible and hence an optimal tree for AP_0. The optimal solution of AP_n will give the required solution.

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

$$\pi_v \equiv \min\{w_{uv} + \pi_u : (u,v) \in E\} = w_{\bar{u}v} + \pi_{\bar{u}}$$

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

procedure A2
while $d(\bar{u}) = 3$ **do**
begin
cut $f = g(\bar{u})$, and let $e \in \bar{T}$ be the pivot edge via
$$\epsilon = \bar{w}_e = \min\{\bar{w}_{ij} : (i,j) \in \delta^-(X)\}$$
$T \leftarrow T + e - f$
$\bar{u} \leftarrow t(e)$
end{while}

The pivot bound of this algorithm is $\frac{1}{2}n(n-1)$ and the increase in the number of pivots is due to the dummy sink node. Both Balinski's and Akgül's algorithms have $O(n^3)$ complexities.

Chapter 3

A DUAL FEASIBLE FOREST ALGORITHM

In the following, we present a dual-feasible forest algorithm for the assignment problem. The algorithm, being a modification of Paparrizos' [27] algorithm, is guided by the signature of a strongly feasible tree and terminates with such a tree. It has the time complexity $O(n^3)$ for dense problems using elementary data structures. For sparse graphs, using Fibonacci-heaps of Fredman and Tarjan, the complexity drops down to $O(n^2 \log n + nm)$. Throughout this chapter, the notation presented in Section 2.1 will be used.

3.1 The Algorithm

First, we will describe Paparrizos's [27] 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., 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 root is identified. If there is no such node, then T is SFT and hence is optimal. Let $v \in C$ 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
$e \leftarrow \arg\min\{\bar{w}_e : e \in \delta(T_-, T_+)\}$
$\epsilon \leftarrow \bar{w}_e$
Let $t(e) \in T_k$
then
$\pi_v \leftarrow \pi_v - \epsilon \ \forall v \in T_k$
$T_- \leftarrow T_- \setminus T_k$
$T_+ \leftarrow T_+ + T_k + e$
end {**while**}

The main invariant during **link** is that 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 **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.

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 $t(e) = u \in T_q$
then
$\pi'_z \leftarrow \pi_z - \epsilon \ \forall z \in T_-$
$T'_+ \leftarrow T_+ + T_q + e$
$T'_- \leftarrow T_- \setminus T_q$
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.**

Let $d(v)$ represent 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 subtree(s) 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 3.1 *The new forest $T' = (T'_+, T'_-)$ is dual-feasible.*

Proof: It suffices to show that with respect to dual variables π', 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 π', edge e has zero reduced cost.

Therefore, T' is dual feasible.□

Clearly, **decompose** routine does not affect dual-feasibility. As a result, the forest of the $(k+1)^{st}$ iteration is dual-feasible.

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

Now, we bound the total number of pivots.

Theorem 3.1 *The algorithm requires at most $(n-1)(n-2)/2$ pivots.*

Proof: Let $\sigma'_i = \sigma_i(T_+), i = 1, 2, 3$ at the beginning of a **stage**. A **stage** ends with a pivot of **type 2** or **3**. A pivot of **type 1** will decrease σ'_2 by 1. Hence, the number

of pivots during a **stage** is bounded by $\sigma_2' + 1 \leq \sigma_2 + 1 = n - \sigma_1 - \sigma_3 + 1 \leq n - \sigma_1$ since $\sigma_3 \geq 1$. Because σ_1 is at most $n-1$ and at least 2 at the beginning of a **stage**, the maximum number of pivots is

$$\sum_{\sigma_1=2}^{n-1}(n-\sigma_1) = \sum_{j=1}^{n-2} j = \frac{(n-1)(n-2)}{2} \quad \square.$$

Now, we give the time complexity of the algorithm.

Theorem 3.2 *The algorithm can be implemented so that it has $O(n^3)$ time complexity for dense graphs and $O(n^2 \log n + nm)$ for sparse graphs.*

Proof: It suffices to show that a **stage** can be implemented at $O(n^2)$ and $O(n \log n + m)$ time for dense and sparse graphs respectively. First we consider the dense case. Clearly, other than the selection of link-edge, everything else in a pivot can be performed in $O(n)$ time per **stage**. To achieve $O(n^2)$ bound per **stage**, we need to analyze the cost of selection of link-edge altogether in a **stage**. Since, each such edge has its head in T_+, we store enough information attached to these nodes. Specifically, let

$$s(v) = \min\{\bar{w}_{iv} : i \in T_-\} \ \forall v \in T_+ \cap C$$

$$nb(v) = j \quad \text{if} \quad \bar{w}_{jv} = s(v)$$

In other words, $s(v)$ is the smallest reduced cost among the edges in $\delta(T_-, v)$ and $nb(v)$ is the tail of such an edge. After a pivot, since the dual variables of all the nodes in T_- decrease by ϵ, we have

$$s(k) \leftarrow s(k) - \epsilon, \quad k \in T_+ \cap C,$$

For a **type 1** pivot, at least one source node, say node v, is transferred from T_+ to T_-. Let T'' be the subtree obtained by deleting edge $(p(v), v)$. We visit the source nodes in T'', for each edge e in $\delta(T'', T_+ \setminus T'')$ compute reduced cost of e, compare with $s(h(e))$ and update $s(h(e))$ and $nb(h(e))$ if necessary. Thus, during a **stage**, each edge is examined at most once for the computation of $s(v)$ and $nb(v)$. We maintain a list representing $T_+ \cap C$. Sink nodes in T'' are deleted from the list.

For a **type 2** or **type 3** pivot, a **stage** is over. After updating dual variables and $s(v)'s$ as above, we compute afresh $s(v)'s$ for sink nodes added to T_+ during the last pivot.

In order to determine pivot or link edge, we compute $\min\{s(v) : v \in T_+ \cap C\}$ and the pivot edge is $(nb(v), v)$ for a minimizing v. This completes the dense case.

For the sparse case, we store $s(v)$'s in Fibonacci heaps [18] . Thus, the cost of updating $s(v)$'s and selection of pivot edges will be $O(n \log n)$. Since we may have to

examine every edge at least once during a **stage**, total cost of these operations will be $O(n \log n + m)$ per **stage**. Since, in any **stage**, we can perform $O(n)$ pivots, updating dual variables after each pivot is not acceptable. As is shown in Akgül [2], and Goldfarb [20], the total cost of dual updates and tree/forest operations in a **stage** can be bounded in $O(n)$ time. The basic idea is to maintain an offset between actual reduced costs and those stored in $s(v)$'s, and compute ϵ with respect to $\min\{s(v) : v \in T_+ \cap C\}$ and offset. Dual variables can be updated when a **stage** is over.

3.2 Variations

1. It is not necessary to start with 'Balinski tree'; the algorithm works as long as sink nodes of degree 1 are incident with the root. We can work with trees and dual variables obtained by familiar row-minimum, column-minimum method. For each $i \in R$, let $w_{ij(i)} = \min\{w_{ij} : j \in C\}$. The edges $E_0 = \{(i, j(i)), i \in R\}$ form a part of the initial forest. Let $Q \subset C$ be the set of isolated sink nodes in (R, C, E_0).

   Setting $\pi_i = -w_{ij(i)}$, $\forall i \in R$, and $\pi_i = 0$, $\forall i \in C$ we obtain a dual-feasible solution for which edges in E_0 have zero reduced costs. Let r be a new (artificial) source node. By adding artificial edges (r, j) $\forall j \in C$ we obtain an initial tree. We set $\pi_r = -K$, and assign cost K to all artificial edges. One can add a new source node, say i_0, and edge (i_0, r) with cost 0 to the initial tree formed. (This last step is not necessary, it is introduced so that the new graph has a matching provided that the old graph has one). One may also apply column-minimum operation to nodes in Q to obtain better dual variables, but there is no need to add any edges to E_0.

   The value of K is not important, e.g., one can set $K = 0$. During the course of the algorithm r (and i_0) will not have any dual-variable changes, and none of the artificial edges will be a link-edge. So, once the initial subtrees T_- and T_+ are constructed, the artificial edges may be deleted. In this version, T_+ will be a forest of subtrees not necessarily SFTs as opposed to being a single SFT. Let $e = (u, v)$ be the link-edge and $d(v)$ represent the degree of v in $T_+{}'$ i.e. forest obtained after the link operation. If $d(v) = 2$, we add the subtree rooted at v in $T_+{}'$ to $T_-{}'$. If $d(v) = 3$, we apply a regular **type 1** pivot and delete edge $(p(v), v)$. In the case when $d(v) = 1$, again a **stage** is over. We apply **decompose** and add the resulting subtrees if any to the collection T'_- (a **type 2** pivot) or we just continue (a **type 1** pivot). **Decompose** and **link** routines are the same as before.

   This version of our algorithm shows some similarities with the algorithm of Paparrizos et al. [1] which is a valency related algorithm that is neither primal-dual nor does it preserve dual or primal feasibility throughout. The iterations of this algorithm are guided by the valency structure of special forests called 'superforests'. By definition [1], a spanning forest F of G is called a 'superforest' provided the following

two conditions hold:

(i) Every connected component of F contains precisely one root which is a sink node,

(ii) Every node in C which is not a root has degree 2.

The algorithm starts with a dual-feasible superforest F_0 and constructs a sequence $F_0, ..., F_k$ of superforests by a special pivoting strategy such that F_k is dual-feasible and contains an optimal assignment. Dual-feasibility, however, is relaxed during the intermediate steps of the algorithm.

The two algorithms mentioned above differ in the way they handle the subtrees of the forests and they make the dual-variable changes. In Paparrizos et al.'s algorithm, the dual-variable changes are made so that they only guarantee the reduced cost of the link-edge being 0. Our dual-variable changes, however, guarantee the dual-feasibility of the whole forest. Although our algorithm applies **decompose** considering the degree of v (the head of the link-edge) only, theirs pay attention to keep the forest a superforest by making changes in the roots of the subtrees where necessary.

2. The idea of performing dual-variable changes over all subtrees in T_- can be applied to algorithms in [1] and [27] so that modified algorithms become dual-feasible forest algorithms.

Chapter 4

A CRISS-CROSS LIKE ALGORITHM

The criss-cross type algorithm is a polynomially bounded nonsimplex method for solving assignment problems. It begins by finding the optimum solution for a problem defined from the first row, then finding the optimum for a problem defined from rows one and two, etc., continuing sequentially until it solves the problem consisting of all the rows. At some stages, dual simplex type pivots are made to obtain a dual-feasible solution for a problem defined from a subset of rows only. At others, primal simplex type pivots are made to obtain a SFT. The algorithm switches between dual and primal pivots and eventually finds the optimal solution.

In order to convey the idea of out algorithm in a simple fashion, we let the following SFT T be the starting tree. Later, we will give more concrete initial starting solutions.

$$(1,j) \in T \quad \forall j \in C$$

$$(i,j) \in T \quad \text{where } i = j+1 \ \ \forall j \neq n \in C$$

The initial dual variables of nodes in T are

$$\pi_i = \begin{cases} 0 & if \quad i \in R \cap 1 \\ w_{1,i} & if \quad i \in C \\ w_{1,i-1} - w_{i,i-1} & if \quad i \neq 1 \in R \end{cases}$$

By construction, all edges emanating from source node 1 are dual-feasible. Column node n is the unique sink node with degree 1 in T. Then, T is a primal feasible (since SFT) tree rooted at sink node n, i.e., it looks like Fig. 4.1.

Our algorithm has two phase, a **dual phase** and a **primal phase**. During k^{th} dual phase, the dual-feasibility of all edges emanating from source node k is attained. In doing this, we retain the dual-feasibility of the so far attained edges. However, primal feasibility may be ruined. If so, we move to the primal-phase where we do pivots in order to get

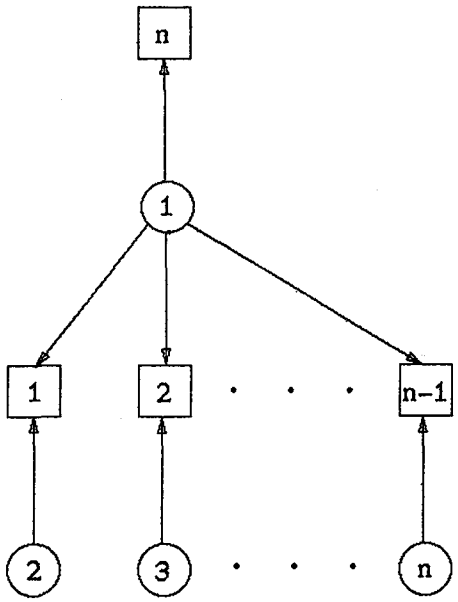


Figure 4.1:

back a primal feasible SFT. So, our algorithm looks like

begin
$i \leftarrow 1$
while $i \leq n-1$ **do**
begin
$i \leftarrow i+1$
i^{th} **dual phase:**
let $\epsilon_i = \min_j\{\bar{w}_{ij} : j \neq p(i) \text{ and } j \in C\} = \bar{w}_{ih}$
if $\epsilon_i < 0$
then begin
$\pi_i \leftarrow \pi_i - \epsilon_i$
cut $(p(i), i)$
link (i, h)
if $deg(h) = 3$
then begin
cut $(p(h), h)$
let T_h be the subtree rooted at h
go to primal phase
end
else go to dual phase
end
else go to dual phase
primal phase:
let $\epsilon = \min\{\bar{w}_{gl} : g \leq i \text{ and } g \in T_h, l \neq T_h\} = \bar{w}_{gk}$

$\pi_v \leftarrow \pi_v - \epsilon \quad \forall v \in T_h$
link (g,k)
if $deg(k) = 3$
then begin
cut $(p(k),k)$
let T_h be the subtree rooted at k
go to primal phase
end
else go to dual phase
end {while}
end {algorithm}

Theorem 4.1 *The dual-feasibility of all edges emanating from source nodes in* $[1..k]$ *is retained through the primal phase starting with* $i = k$.

Proof: (by induction)

base case $(k = 2)$

After the dual variable change in the second dual phase, all edges emanating from source node 2 are dual-feasible. Since no change is made in the dual-variable values of sink nodes, there is no change in the dual-feasibility of edges emanating from source node 1; by construction these edges were in the initial SFT. Assume the entering edge chosen during dual phase is not $(2,n)$. If so, the algorithm moves to the 3^{rd} dual phase and the statement is obvious. Otherwise, T_h contains source node 2 together with another source node other than 1, and a sink node other than n. In primal phase, we consider all edges emanating from source node 2 that are not in T_h. Due to the preceding dual phase, the reduced costs of all such edges are ≥ 0. Since the dual value of the sink node in T_h is decreased by a nonnegative amount, the dual feasibility of all edges emanating from source node 1 is also retained. Because of the way the link-edge is chosen during the primal phase, all edges emanating from source node 2 are still dual-feasible. In the worst case, we can repeat the primal phase so many times that T_h grows up to include all the source nodes (except 1) and all the sink nodes (except n). But, since the pivots up to now have not ruined the dual-feasibility of edges emanating from source nodes 1 and 2 simply because of the way the link-edge is chosen, the pivot in this step i.e. an edge incident with n entering, will retain the dual-feasibility of edges emanating from source nodes in $[1..k](k = 2)$ as well. So, base case holds.

inductive case

Assume the statement holds $\forall k \in [1..m]$. We are going to show that it also holds for $k = m + 1$.

By the inductive hypothesis, the dual phase starts with all edges emanating from source nodes $k \leq m$ as dual-feasible. During the dual phase, we find the edge with minimum reduced cost emanating from source node $m+1$. Say, this edge has negative reduced cost (If not, we simply move directly to the next dual phase which does not ruin the dual-feasibility of edges emanating from nodes in $[1..m+1] \cap R$ at all). Since no change is made in the dual values of sink nodes, the inductive hypothesis holds at the end of the dual-phase. During the pivot in the primal phase, the edge with minimum reduced cost among all edges in $\delta(T_h)$ emanating from nodes in $[1..m+1] \cap R$ is chosen. So, the pivot and the dual variable changes do not ruin the dual-feasibility of edges in $\delta(T_h)$ with tails in $[1..m+1]$. If a source node in $[1..m+1]$ is not in T_h, then all the edges emanating from this node with head nodes in T_h will have increased reduced costs (since pivot edge has nonnegative reduced cost) and thus their dual-feasibility will be retained.

So, however many pivots we do during the primal phase, the dual-feasibility of edges emanating from source nodes in $[1..m+1]$ will be retained due to the way the link-edge is chosen. Thus, the statement holds for $k = m+1$ as well. □

Corollary 4.1 *At the beginning of each dual phase, say i^{th} one, there is a SFT at hand and all edges emanating from source nodes in $[1..i-1]$ are dual-feasible.*

Corollary 4.2 *After $n-1$ repetitions of the 'while-loop', the optimal SFT is attained.*

Proof:

Follows trivially form Theorem 4.1. At the last iteration ($i = n$), the dual phase guarantees the dual-feasibility of all edges emanating from source nodes in $[1..n]$ and the primal phase constructs the optimal SFT. □

Theorem 4.2 *Each while loop of the algorithm can be accomplished in $O(n)$ pivots.*

Proof:

In the worst case, following each pivot in the dual phase, $n-3$ primal pivots are required to construct the SFT. This adds up to a total of $n-2$ pivots in each iteration of the algorithm. □

Corollary 4.3 *Worst case pivot bound of the algorithm is $(n-1)(n-2)$.*

Theorem 4.3 *The total work done while executing the algorithm is $O(n^3)$.*

Proof: It suffices to show that each *iteration* of the algorithm can be implemented at $O(n^2)$ time for dense graphs. The choice of the link-edge during a dual phase can be performed in $O(n)$ time per *iteration*. To achieve $O(n^2)$ bound per *iteration*, say i^{th} one, we need to analyze the cost of selection of link-edge during the primal phase of the iteration. Since each potential link-edge has its head in $\overline{T_h}$, we store enough information attached to these nodes. Specifically, let

$$s(v) = \min\{\bar{w}_{gv} : g \leq i, g \in T_h\} \quad \forall v \in \overline{T_h} \cap C$$

$$nb(v) = j \quad \text{if} \quad \bar{w}_{jv} = s(v)$$

In other words, $s(v)$ is the smallest reduced cost among the edges in $\delta(T_h, v) \quad \forall v \in \overline{T_h} \cap C$ and $nb(v)$ is the tail of such an edge. After a primal pivot, we have

$$s(k) \leftarrow s(k) - \epsilon \quad k \in \overline{T_h} \cap C,$$

and we update $s(k)'s$ accordingly.

Say, the link-edge in primal phase is (g, k) and k has degree 3. Then, the updated T_h includes the subtree rooted at k before the link operation. So, at least one source node, say node v, is transferred from $\overline{T_h}$ to T_h. We visit the source nodes in T_k, for each edge e in $\delta(T_k, \overline{T_h} \setminus T_k)$, compute reduced cost of e, compare with $s(h(e))$ and update $s(h(e))$ and $nb(h(e))$ if necessary. Thus, during a primal phase, each edge is examined at most once for the computation of $s(v)$ and $nb(v)$. We maintain a list representing $T_h \cap C$ with sink nodes in T_k deleted from this list.

In order to determine the link-edge of the primal phase, we compute $\min\{s(v) : v \in \overline{T_h} \cap C\}$ and the pivot-edge is $(nb(v), v)$ for a minimizing v. □

Arguing as in the proof of Theorem 3.2, this algorithm also can be implemented with $O(n^2 \log n + nm)$ time complexity for sparse graphs with n nodes and m edges via the use of Fibonacci heaps [18].

4.1 Variations

It is not necessary to start with the suggested initial tree. We now give two further suggestions in order to construct the initial SFT.

1. At some problem instances, there is a near-optimal primal-feasible solution available. In these cases, the algorithm can start with the SFT at hand. In this version, the row nodes should be traversed in a postorder manner. So, firstly, a re-ordering of row nodes from 1 to n is made and the algorithm is applied starting with $i = 1$

instead of 2. If at the i^{th} dual-phase, (i,h) is the link-edge, then both $(p(i),i)$ and (i,j) $\forall j$ such that $p(j)=i$ are deleted. Consequently, in this version, $T \setminus T_h$ rather than being a SFT throughout the algorithm, is a collection of SFTs. The primal phase of the algorithm is not altered. For this version, the average pivot bound is believed to be less than the original, although, $(n-1)(n-2)$ stays as the worst case pivot bound. The time complexity of this version is again $O(n^3)$.

2. If nothing is known about the problem instance, it is best to start with the maximum attainable dual-feasibility. To do this, we construct the initial SFT, T as follows:

$$(1,j) \in T \quad \forall j \in C$$

The initial dual variables of nodes in T are

$$\pi_i = \begin{cases} 0 & if \quad i \in R \cap 1 \\ w_{1,i} & if \quad i \in C \end{cases}$$

```
Let R* = {1}
for i := 2 to n do
    begin
        let j* = arg min_j {w_ij - w_1j}
        if deg(j*) = 1 in T
        then begin
                T ← T ∪ (i, j*)
                π_i ← π_j* - w_ij*
                R* ← R* ∪ i
             end
    end {for }
i ← 2
for j := 1 to n do
    begin
        if deg(j) = 1 in T
        then begin
                found ← false
                while i ≤ n and not found do
                   begin
                      if deg(i) = 0 in T
                      then begin
                              found ← true
                              T ← T ∪ (i, j)
```

$\pi_i \leftarrow \pi_j - w_{ij}$
end
$i \leftarrow i + 1$
end
end
end.

The first 'for-loop' adds edges to T in such a way that the edges emanating from row nodes currently in T are dual-feasible. The second 'for-loop' adds the rest of the row nodes in a manner that does not destroy strong feasibility of T. By construction, all edges emanating from source nodes in R^* are dual-feasible. Let r be the unique sink node with degree 1 in T. Then, T is a primal feasible (since SFT) tree rooted at sink r.

Let $\overline{R^*} = R \setminus R^*$. Renumber the nodes in R^* from 1 to $\mid R^* \mid$ and change the numbering of the remaining row nodes accordingly i.e. from $\mid R^* \mid +1$ to n. The algorithm starts with iteration $i = \mid R^* \mid +1$ instead of $i = 2$. If $\mid R^* \mid > 1$, the pivot bound reduces to $(n - \mid R^* \mid)(n-2)$. However, the complexity is again $O(n^3)$.

Chapter 5

SOME REMARKS ON PARALLELISM IN ASSIGNMENT PROBLEMS

Most large-scale optimization problems arising from real-world applications can be decomposed into quasi-independent subproblems (corresponding to time periods, geographic districts, physical or logical commodities, etc.), allowing the possibility of attack via iterative algorithms that exhibit a high degree of parallelism. Theoretical research into decomposition methods for large-scale optimization dates back to Dantzig and Wolfe (1960), but the absence of computer hardware capable of exploiting the parallelism inherent in these methods has long discouraged potential research in this area. With the invention of multicomputers and multiprocessors, research into new decomposition methods is not merely stimulated but is made inevitable by the goal of achieving the speedups made possible by such architectures [8].

Our major motivation in starting this research was the design of algorithms for the linear assignment problem that can exhibit parallelism. Apart from being different techniques for solving AP with worst case bounds as good as the best available in the literature, the two algorithms presented will provide essential guidance towards attaining our ultimate goal. Both algorithms have the flexibility of working on forests instead of a single tree. Given two (or more) dual-feasible trees, the dual-feasible forest algorithm is capable of combining the given information in order to find the optimal solution of the underlying AP. Similarly, our criss-cross like algorithm can combine a group of SFTs into an optimal SFT. So, grouping a given instance of AP into certain simple independent subproblems and solving these embedded subproblems optimally (with perhaps more efficient techniques available in the literature), we can then apply either of our algorithms to find the optimal solution of the main problem.

As a complementary research, we are planning to exhibit an efficiency test for out algorithms. To do this, certain well known efficient methods will be implemented together with ours using the same data structures and implementation techniques. We believe that

only then we will be able to give information about the efficiency of the two algorithms presented. Although it may sound like we have attained our goal of designing iterative algorithms, there is yet a lot to be done. Another further research could be the design of improved variants of the two algorithms discussed, that will exhibit parallelism in APs in a more efficient fashion.

Bibliography

[1] H. Achatz, P. Klenschmidt and K. Paparrizos, "A dual forest algorithm for the assignment problem", *Submitted* (1989).

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

[3] M. Akgül, "A genuinely polynomial primal simplex algorithm for the assignment problem", SERC Report IEOR 87-07, Bilkent University, (1987).

[4] M.L. Balinski, "Signature method for the assignment problem", *Operations Research* 33, 527-536 (1985).

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

[6] M.L. Balinski and R.E. Gomory, "A primal method for the assignment and transportation problems", *Management Science* 10, 578-598 (1964). assignment problem", *Mathematical Programming* 34, 125-141 (1986).

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

[8] R.J. Chen and R.R. Meyer, "Parallel optimization for traffic assignment", *Mathematical Programming* 42, 327-345 (1988).

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

[10] R.E. Burkard, "Traveling salesman and assignment problems: a survey", *Annals of Discrete Mathematics* 4, 192-215 (1979).

[11] R.E. Burkard W. Hahn, and W. Zimmerman, "An algebraic approach to assignment problem", *Mathematical Programming* 12, 318-327 (1977).

[12] G. Carpaneto and P. Toth, "Primal dual algorithms for the assignment problem", *Discrete Appl. Math.* 18, 137-153 (1987).

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

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

[15] E.A. Dinic and M.A. Kronrad, "An algorithm for the solution of the assignment problem", *Soviet Mathematics Doklady* 10, 1324-1326 (1969).

[16] J. Edmonds and R.M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems", *JACM* 19, 248-264 (1972).

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

[18] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms", *Journal of ACM* 34, 596-615 (1987).

[19] F. Glover, R. Glover and D. Klingman, "Threshold Assignment Algorithm", *Research Report*, (1982).

[20] D. Goldfarb, "Efficient dual simplex algorithms for the assignment problem", *Mathematical Programming* 33, 187-203 (1985).

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

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

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

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

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

[26] J.B. Orlin, "On the simplex algorithm for networks and generalization networks", *Mathematical Programming Study* 25, 166-178 (1985).

[27] 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, 269-289 (1988).

[28] Roohy-Laleh, "Improvements to the theoretical efficiency of the simplex method", Ph.D. Thesis, University of Charleton, Ottawa (1981). Dissertation Abstract International vol. 43, 448B (August 1982).

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

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

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