NETWORK FLOWS AND COMBINATORIAL OPTIMIZATION

R. Chandrasekaran
UT Dallas

July 8, 2004

Chapter 1

Matching and Related Topics

This chapter includes several related problems that we have chosen to label as *matching*. There is a book on the topic by **Lovasz** and **Plummer**. There are several perspectives on this problem, and depending on the view point, we get different versions of this problem.

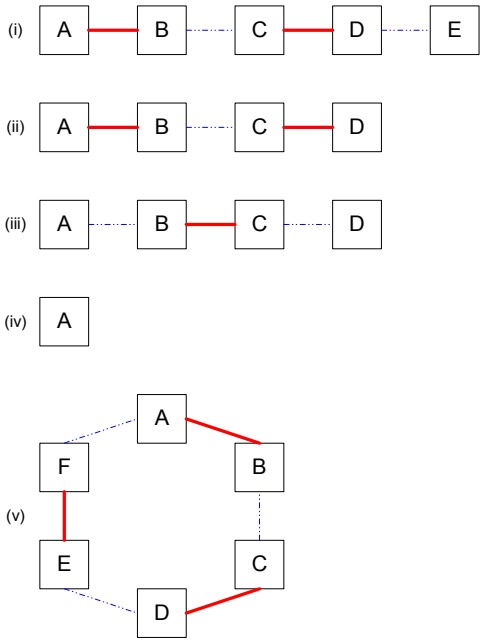
1.1 Problem:

Let $G = [N, E]$ be an undirected graph. $M \subset E$ is a *matching* if no two edges in M meet. In other words, no node in N has more than one arc of M incident at it. Given weights c_e for $e \in E$, the problem of $\max_M \sum_{e \in M} c_e$ is called the *weighted matching* problem or simply the matching problem. When $c_e = 1 \; \forall \; e \in E$, the problem is known as the *cardinality matching* problem. Often, the method to solve cardinality problem is used as a subroutine for the general problem in a primal dual format.

A node is *free (exposed)* relative to a matching M if no edge of M is incident at it. A matching is said to be *perfect* if it leaves no node exposed. A necessary condition for a perfect matching to exist in a graph is that $|N|$ is even. A *path* is said to be *alternating with respect to* M if its alternate edges belong to M and others do not. Thus, if the path consists of edges $e_1, e_2, ..., e_n$ and if $e_1 \in M$, then all odd numbered edges are in M and even ones are not; if $e_1 \notin M$, then all odd ones are not in M and even ones are in M. A path is *augmenting* if it is a simple alternating path with its end nodes exposed.

Theorem 1 *A matching is of maximum cardinality iff* $\exists$ *no augmenting path relative to it.*

Proof: "Only if" is clear. To prove the if part: assume M and P are any two matchings. Consider the graph $G^{\Delta} = [N, P\Delta M]$ where $P\Delta M = (P - M) \cup (M - P)$. Let solid lines indicate edges of $(P - M)$ and dashed lines the edges of $(M - P)$. Then, since there can be no more than one edge of M and one of P incident at any node, there can be no more than two at any node in G^{Δ}. Thus, this graph consists of components each of which is a cycle or a path. In either case, the edges are alternating in and out of M and P. Thus, each component looks like one of the following diagrams:



In cases (i), (iv), and (v), the number of edges in $(P - M)$ and $(M - P)$ are equal. In case (ii), $(P - M)$ has one more edge, and in case (iii), $(M - P)$ has one more. If $|P| > |M|$, then we must have at least one occurrence of case (ii), and this is an augmenting path relative to M.□

Although this theorem is a characterization, it is far from being useful at this stage. **J. Edmonds'** contribution was in making the process of finding such a path efficient.

1.2 Weighted Matching

If $|N|$ is even and $c_e \geq 0 \; \forall \; e \in E$ (we are assuming without loss of generality that the graph is complete now), then it should be clear that $\exists$ an optimal solution that is a perfect matching. If we require the matching to be perfect

in the weighted matching problem, then it is called a *perfect matching problem*. While the WMP is always feasible, PMP may not be. Since feasibility of PMP requires $|N|$ to be even, we will assume this to be the case when dealing with PMP. Since the number of edges in any perfect matching is $|N|/2$ (a constant for a given graph), the problem with $c'_e = c_e + K$ for $\forall\ e \in E$ is equivalent to the original problem. Thus, we may assume in PMP that $c_e > 0\ \forall\ e \in E$. Thus, PMP can be converted to WMP. To show that the converse is also possible, we first drop the edges with negative weights. Now add those edges not present with a weight equal to 0 to make a complete graph (and add a vertex if necessary to make $|N|$ even) and now do a PMP on the resulting structure. Note the resulting PMP is feasible. Since the two problems are equivalent, we will only consider PMP from now on.

Integer Programming Formulation of PMP:

$$\begin{gathered} x_e = \begin{cases} 1 & e \in M \\ 0 & e \notin M \end{cases} \\ \textstyle\sum_{e:e=(i,j)} x_e = 1\ \forall\ i \in N \\ \max \textstyle\sum_{e \in E} c_e x_e \end{gathered} \tag{1.1}$$

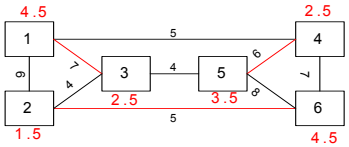
The constraint matrix is *totally unimodular* iff the graph is *bipartite*. In this case, the problem is known in the literature as the *assignment problem* and can be solved as a linear program. The LP dual of the above problem (the dual of the above problem ignoring integrality requirements) is:

$$\begin{gathered} y_i + y_j \geq c_e\ \forall\ e \in E \text{ where } e = (i,j) \\ \min \textstyle\sum y_i \end{gathered} \tag{1.2}$$

Lemma 2 *For any feasible solution to the LP dual and any perfect matching of G (i.e. an integer feasible solution to the primal), we have $\sum c_e \leq \sum y_i$. If equality holds for such a pair of solutions, then they are optimal to the respective problems.*

This is actually weaker than the usual weak duality result because of the integrality requirement. The following example shows we may not always be able to find such a pair where equality holds.

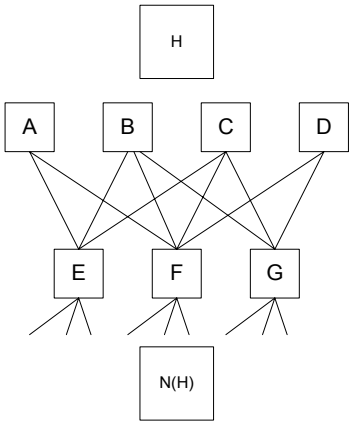
Example:



In this example, $M^* = \{(1,3),(2,6),(4,5)\}$ and $\sum_{e \in M^*} c_e = 18$; the optimal dual solution is $y^* = [4.5, 1.5, 2.5, 2.5, 3.5, 4.5]$ and $\sum y_i^* = 19$.

For bipartite graphs, this equality holds, and this is shown by the *Hungarian algorithm* which is a primal dual algorithm for this problem. Before we proceed to describe this algorithm, we need a definition.

A subset H of nodes is said to be *Hungarian* relative to a graph G if: (i) no two nodes of H are connected in G and (ii) the set $N_G(H)$, of neighbors of H in G (which is the set of nodes of G that are connected to some node of H) satisfies the condition $|N_G(H)| < |H|$.



Clearly, if a graph G contains a Hungarian set, then it can have no perfect matching (whether or not the graph G is bipartite). The algorithm will show that, for bipartite graphs, the converse is also true. However, the converse is not true for the complete graph on three points or for that matter any odd cycle. Actually a stronger version is true for bipartite graphs as in:

Theorem 3 *For any bipartite graph $G = [S, T; A]$, the following statements are equivalent:*

(i) $\exists$ *no perfect matching in G.*

(ii) $\exists$ *a Hungarian set H wholly contained in S or T.*

For any edge weighted undirected bipartite graph G that contains a perfect matching, the maximum in the primal is equal to the minimum in the LP dual.

Corollary 4 *If c_e are integral in addition, $\exists$ an integral solution that is optimal to the LP dual if $\exists$ an optimal solution to this problem.*

Corollary 5 *If G has no perfect matching, then the LP dual is unbounded.*

All of these results are shown by an algorithm which will be generalized for nonbipartite graphs. This algorithm is a primal dual algorithm and starts with an (easily found) dual solution which is integral if c_e are integral.

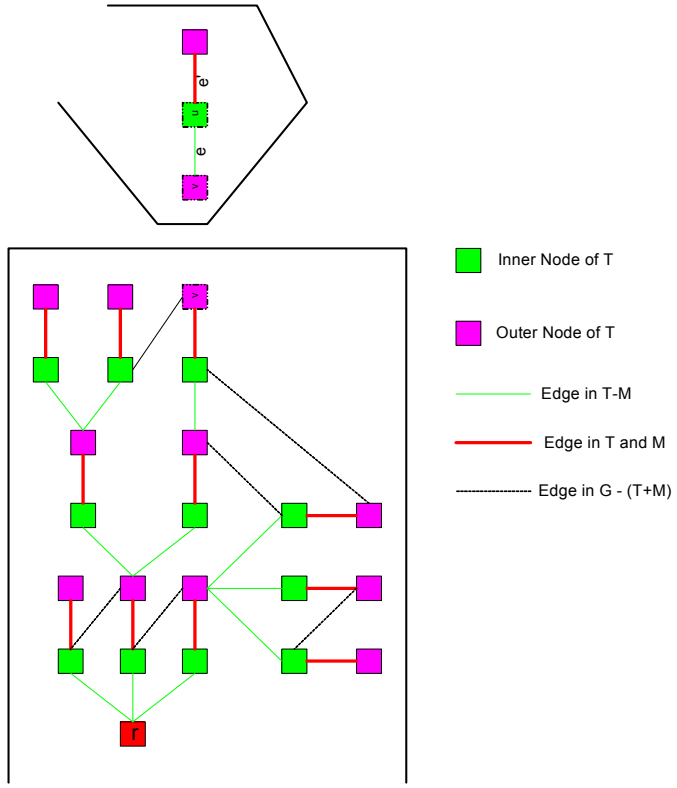
1.2.1 Algorithm I: (bipartite version)

Step 1: Find a feasible solution to the LP dual. If all c_e are integers, then choose y to be integral (this is always possible). For example, let $y_1 = \max c_e$ where the maximum is taken over all edges e incident at node 1. Let $y_2 = \max\ c_e$ where the maximum is taken over all edges incident at node 2 that are not incident at node 1, and so on.
Step 2: Define G^{eq}, the *current equality graph* as follows: $G^{eq} = [S, T; A']$ where $A' = \{e \in A : f_e(y) = y_i + y_j = c_e \text{ with } e = (i, j)\}$.
Step 3: Use subroutine R_1 described below which terminates with either a perfect matching M all of whose edges are in G^{eq} or with a set H of nodes wholly contained in S or T which is Hungarian with respect to G^{eq}. In the former case, the perfect matching solves the primal and the current y solves the dual LP and, therefore, the algorithm terminates. In the latter case, go to step 4.
Step 4: If H is Hungarian in G, stop; $\exists$ no perfect matching in the graph G, and the dual LP is unbounded. Else go to step 5.
Step 5: Use subroutine R_2 to change the dual variables (maintaining integrality of the dual solutions if all c_e are integral).

This will give an improved dual solution, and the equality graph changes ("*for the better*") and we go to step 3 with a new equality graph. Much of the work done in R_1 is unchanged as we will show in R_1. The only termination conditions are in steps 3 and 4. In step 3, we get optimal solutions, and in step 4 we get infeasibility. Now we describe subroutines R_1 and R_2. Up to this point, what we have described is nothing more than what is known in LP as primal dual algorithm's superstructure. R_1 describes the algorithm for solving the so called *restricted primal* problem. R_2 is the usual dual variable change found in primal dual algorithms.

R_1: We have a graph G^{eq} [which is bipartite with the same partition as G because G is]. Let M be any matching in G^{eq} (one could start with the ϕ for example). If the current M is perfect, then we stop as in step 3. If not, let r be an exposed node relative to M in G^{eq}. We will either augment the current matching M so that all its nonexposed nodes remain nonexposed and r and another exposed node are no longer exposed thereby increasing the size of the matching by one or exhibit a Hungarian set of the type promised containing r. For this purpose, we start growing a tree with r as its root. The entire tree will be in G^{eq} and have its alternating edges in M. To begin with, the tree consists only of r which is declared to be an *outer node* of the tree. We look for an edge that connects an outer node of the tree to a node outside the tree through an arc of G^{eq}. If such an edge is found, there are two possibilities: (i) the end of this edge that is not in the tree is an exposed node or (ii) it is not. In the first case, we have an augmenting path connecting the root of the tree to this node, and we augment the matching by deleting from the matching those edges in this path that are in M and including those that are not. The tree is *dismantled*,

and we repeat R_1 with some node that is an exposed node relative to this new matching. In case (ii), we grow the tree by including this edge as well as the edge that is in M that meets this node. Nodes are declared to be *alternatively outer and inner* as we go along any path in the tree traversing it from the root. We now continue to repeat this process until the tree cannot grow any further, and we do not get case (i). In general, the picture looks like:



All outer nodes must be in one side of the partition of the nodes in G and so do the inner nodes which are in the other and $|outernodes| = |innernodes|+1$. If we cannot grow the tree and cannot improve the matching, then the set of outer nodes forms a Hungarian set relative to G^{eq}, and as promised, it is completely contained in S or T. Clearly the tree cannot grow for more than n steps, and the matching cannot grow for more than $n/2$ steps and, hence, we must either get a perfect matching or a Hungarian set in polynomial time. When we find a Hungarian set in G^{eq}, we go to R_2.

R_2: If H is the Hungarian set in G^{eq}, then there are two possibilities: (i) it is also Hungarian in G, in which case G does not have any perfect matching and we will show that the dual is unbounded; (ii) H is not Hungarian in G;i.e. $N_G(H) \supset N_{G^{eq}}(H)$. Hence, $\exists$ edges in G that are not in G^{eq} joining an outer

node of tree T to a node outside the tree. The inclusion of such an edge in G^{eq} will allow the subroutine R_1 to grow the tree further or increase the size of the matching. We now change the dual solution as follows:

Let $\epsilon = \min[f_e(y) - c_e]$ where $e = (i, j)$ where the minimum is taken over all edges e with one end as an outer node of the tree and the other outside the tree. Thus, $\epsilon > 0$ and is integral if all c_e are. If $N_G(H) = N_{G^{eq}}(H)$, then $\epsilon = \infty$, and this will show that the dual is unbounded in this case. The new dual solution is given by:

$$y'_i = \begin{cases} y_i + \epsilon & i \text{ inner node of } T \\ y_i - \epsilon & i \text{ outer node of } T \end{cases} \tag{1.3}$$

It should be clear that y' is feasible to the LP dual and all edges whose both ends are in T are retained in the new G^{eq} as are those in the current matching. At least one edge is included in the new G^{eq} so that the tree can grow. The only edges that might drop out are those with one end as an inner node and the other end not in the tree; these are neither in M nor in the tree and, hence, do not matter for our construction. Thus, *we can retain the old M and the old tree as is.* Please note the solution y' is better than y for the dual because $\sum y'_i = \sum y_i - \epsilon$ since there are more outer nodes in the tree than inner nodes. This also shows the unboundedness of the LP dual if $\epsilon = \infty$. All the theorems and corollaries have been proven. We have also shown the following theorem:

Theorem 6 *G is bipartite $\iff$ the extreme points of P_G are precisely the incidence vectors of perfect matchings in G where P_G is defined as:*

$$P_G = \{x | x \geq 0; \sum_{e:e=(i,j)} x_e = 1\} \tag{1.4}$$

This completes our description of the algorithm for bipartite graphs. We now turn our attention to the general case.

1.3 General Graph:

The previous formulation as an integer program is still valid. However, if the problem is solved as an LP, we are no longer guaranteed integrality. P_G has fractional extreme points (indeed they are multiples of 1/2). We, therefore, have to introduce inequalities that are satisfied by all integral but not by some fractional solutions. These constraints are called *cuts* and were used by **R.E. Gomory**. While **Gomory's** algorithm would solve the LP and then add one or a few cuts at a time, **J. Edmonds'** idea is to add all the necessary cuts (and a few redundant ones) at one time a priori. Thus, he gives a direct characterization of IP_G, the convex hull of integral points in P_G.

$$IP_G = \{x : \begin{cases} x_e \geq 0; \sum_{e:e=(i,j)} x_e = 1 \ \forall \ i \in N \\ \sum_{e:e=(i,j);i,j \in S} [x_e] \leq \lfloor S \rfloor / 2 = q_S \ \forall \ S \ \subset N \end{cases}\} \tag{1.5}$$

This is the first such instance where the integer hull has been completely described and is different from the set of feasible solutions to the corresponding LP.

It should be clear that 0/1 vectors satisfying the first two conditions of 1.5 also satisfy the third. Hence, IP_G as defined by 1.5 does include incidence vectors of all perfect matchings in G. Now it remains to show that the extreme points of IP_G as defined by these relations are all integral. This is done by showing that any linear function is maximized at an integer point of IP_G. The means used for showing this is an algorithm that is strongly polynomial n.

Dual of IP_G :

$$y(S) \geq 0 \; \forall \; S \subset N$$
$$f_e(y) = y_i + y_j + \sum_{S:i,j \in S} y(S) \geq c_e \forall \; (i,j) = e \in E$$
$$\min[\sum_i y_i + \sum_S q_S y(S)$$

A solution $(\{y_i\}, \{y(S)\})$ is *feasible to the dual* if the constraints of this LP are satisfied. The following result is the weak duality theorem applied to these two problems;

Lemma 7 *For any perfect matching M and any feasible y,*

$$\sum_{e \in M} c_e \leq \sum_i y_i + \sum_S q_S y(S). \tag{1.6}$$

If equality holds for a pair that they are optimal solutions to the two problems respectively also follows from LP duality.

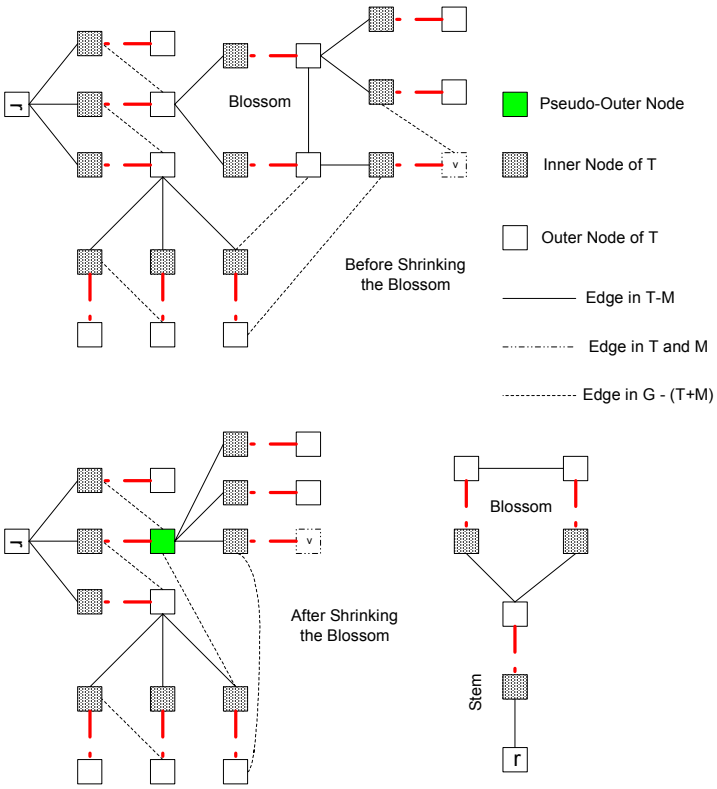
Now we are ready to describe the primal dual algorithm for this case.

1.3.1 Algorithm II:(General Graph)

Step 1: Find a feasible solution to the dual (this is easy). For example, let $y(S) = 0 \; \forall \; S$; and let y_i be chosen as in the bipartite case.
Step 2: Define the equality graph G^{eq} as before using the relation $e \in G^{eq} \iff f_e(y) = c_e$.
Step 3: Using R_1 try to find a perfect matching in G^{eq}. If one is found, then it is optimal and the y is optimal to the dual. We now describe R_1 before stating R_2 . We need this in order to have some terms clarified.
R_1 : We start with an arbitrary matching in G^{eq} as before. We try to grow a rooted tree as before starting with an exposed node r as its root. The tree still consists of edges of which alternate ones are in M and the remaining not in M as one traverses from the root. Alternate nodes are declared outer and inner as before with the root being an outer node. We look for an edge in G^{eq} that connects an outer node of the tree to a node outside the tree. If we find such

an edge, then there are the same two possibilities as before, and we follow the same procedures as before in each case. Thus, in one of these, we augment the matching, and in the other, the tree grows by two edges and two nodes, one of which is inner and the other is an outer node. If the matching is augmented, then as before we discard the tree and start growing another from a node that is exposed relative to the new matching.

All of this is the same as the algorithm for the bipartite case. What is different is that there is another case to consider now. This is because we may have the situation in which we are unable to grow the tree, unable to augment the matching and the outer nodes of the current tree do not form a Hungarian set in the equality graph. This is caused by the possibility that two outer nodes of the current tree may be connected by an edge of the equality graph. In this case, the graph has an odd circuit which was not possible in the bipartite case and is now.



If two outer nodes of the tree are connected by an edge in G^{eq}, then the odd circuit created by adding this edge to the tree is called a *blossom*, and the path from the root to the node in this circuit closest to the root in the tree is called the *stem*. These two are together called a *flower*. Please note that the

node common to the stem and the blossom is an outer node of the tree. We now do an operation called *shrinking* which is done **simultaneously on the blossom, the tree, the matching, the equality graph and the original graph** (at least conceptually). This operation is what we called *condensing of nodes* in multi terminal flows. The resulting node of the tree is called a *pseudo node.* Thus, the blossom condenses to a single outer node which is a pseudo node.

Thus, at any stage of the algorithm, we have a shrunken graph G_s, a shrunken equality graph G_s^{eq}, a shrunken matching M_s, and a shrunken tree T_s as well their original counterparts. There are *real* and pseudo nodes which may have in them real and/or pseudo nodes. However, the number of real nodes inside a pseudo node is odd at all times. The term *current* pseudo nodes refers to the outermost (or equivalently the largest) of the pseudo nodes in the nested family. Thus, there are pseudo nodes inside pseudo nodes and these are not current. Also, we may not *unshrink* a pseudo node unless we are *forced* to do so, and because of this, there may, at any time, be some pseudo nodes that are inner nodes of the current tree. There are only two instances in the algorithm that force the unshrinking of a pseudo node: (i) we find a perfect matching in the shrunken equality graph and want to expand this to a perfect matching in the equality (unshrunken) graph, and (ii) dual variable of a pseudo node goes down to 0 (this will only happen if this pseudo node is an inner node of the tree). One could also unshrink whenever the matching is augmented; while this is conducive to a simpler explanation of the algorithm, it is inefficient for the algorithm itself and, therefore, we will not do this.

If at some stage we are unable to augment the matching, or grow the tree, or shrink some blossom, then the outer nodes (real and pseudo) form a Hungarian set in the shrunken equality graph G_s^{eq}. Now we go into the subroutine R_2 that makes the change in dual variables. If this set is Hungarian in G_s, the shrunken graph, then we will show that G does not have a perfect matching, and the dual problem is unbounded.

R_2: Choose ϵ as large as possible subject to the following conditions:

I For every edge e of $G_s - G_s^{eq}$ whose one end is an outer node of T and the other is not in T , $\epsilon \leq f_e(y) - c_e$.

II For every edge e of $G_s - G_s^{eq}$ both of whose ends are outer nodes of T $\epsilon \leq (f_e(y) - c_e)/2$.

III For every S which represents the set of all real nodes corresponding to a current inner pseudo node of T, $\epsilon \leq y(S)/2$.

Now change the dual variables as follows:

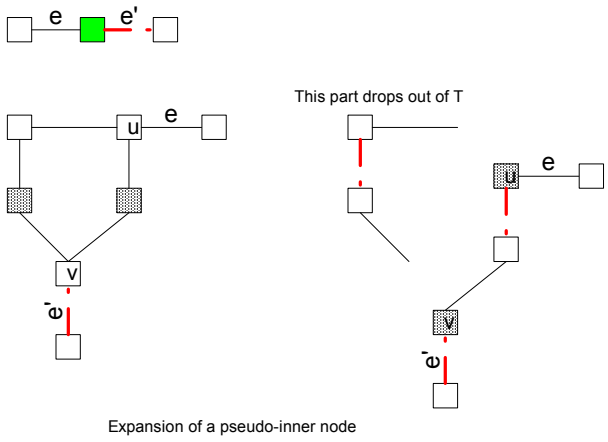
$$y_i' = \begin{cases} y_i - \epsilon & i \text{ real outer or } \in \text{pseudo outer node of } T \\ y_i + \epsilon & i \text{ real inner or } \in \text{pseudo inner node of } T \end{cases} \tag{1.7}$$

$$y'(S) = \begin{cases} y(S) + 2\epsilon & S \text{ a current pseudo outer node of } T \\ y(S) - 2\epsilon & S \text{ a current pseudo inner node of } T \end{cases} \tag{1.8}$$

Clearly the new y is feasible to the dual and $y'(S) \geq 0 \; \forall \; S \subset N$. Also we will assume that $y(S) > 0$ for all the current pseudo inner nodes of T. For, otherwise, ϵ may not be positive and $y' = y$.

Let us see the effect of these dual variable changes on various structures we have introduced so far. Clearly, edges both of whose ends are outside the tree will not be affected since $f_e(y)$ does not change for these. The same is true for edges both of whose ends is inside the same pseudo node (whether this is inner or outer or outside the tree). If an edge has both ends as outer nodes of the tree or inside distinct outer nodes of the tree, then $f_e(y)$ decreases by 2ϵ; however, ϵ was chosen so that the new solution is still feasible. One of these edges is added to G_s^{eq} if ϵ is determined by (II), and in this case, we will be able to shrink the tree further. If both ends of an edge are inner nodes or inside inner nodes of the tree, then this edge will drop out of G_s^{eq}; however, this is of no concern since this edge is not in M_s nor in the tree. If one end is an outer node or inside one and the other is an inner node or inside one, then $f_e(y)$ is unchanged and, hence, is not affected. Edges with one end an inner node of the tree or inside one and the other outside the tree will have an increase in $f_e(y)$ and, hence, may drop out; but again this is of no concern. Edges with one end an outer node of the tree or inside one and the other outside the tree will have a decrease in $f_e(y)$, and one of these may be added to G_s^{eq} if ϵ is determined by (I), and in this case, the tree will be able to grow.

While changing the dual variables if ϵ is determined by (III), then $y'(S) = 0$ for some pseudo inner node, and this is one instance where we are forced to unshrink this pseudo node; we, however, do this only for one *layer*. By this, we mean we do not unshrink any pseudo node inside this pseudo node. We will explain this process now.



Expansion of a pseudo-inner node

Let e_1 be the edge of the matching that is incident at the pseudo node S. Let e_2 be the edge incident at S on the path from S to the root of the tree. (This exists because S is inner). Let the blossom that induced the creation of pseudo node S be B. *Unshrinking the blossom* is the following process: edges incident at the node in the blossom that meets e_1 are not used in the matching's completion within the blossom; the ones adjacent to these are, and the remaining ones are alternatively in and out of the matching. The even path in the blossom between the two nodes of the blossom that meet edges e_1 and e_2 is retained in the tree, and the odd one is dropped from the tree; the nodes are labeled as outer and inner in the usual manner. See the diagrams below. After this process, we may still have pseudo nodes some of which now become current. *Please note that only inner pseudo nodes are unshrunk*. For this purpose, we remember only the odd circuit of the blossom but not the original matching because the final one bears no relationship to the original. The edges of the blossom that drop out of the tree still remain in the equality graph. Now we go back to R_1 in step 3.

Termination of the algorithm: In step 3, we terminate if we find a perfect matching in G_s^{eq} by unshrinking one step at a time and recovering the perfect matching in G^{eq} just as we have described the process of unshrinking before. The other possibility is that at some stage of the algorithm we are unable to augment the matching, grow the tree, shrink the tree, and the set of outer nodes in the tree T_s is Hungarian in not only G_s^{eq} but also G_s. Further, there are no pseudo inner nodes to limit the value of ϵ. Thus, in this case $\epsilon = \infty$, and the dual is unbounded. Although we are not guaranteed a Hungarian set in G, there is no perfect matching in G because out of each pseudo node, at least one real node inside it will have to be matched to a node outside this set, and there are not enough to go around.

The number of times M can increase is at most $|N|/2$. The number of times we shrink a blossom (between two matching augmentations) is no more than $|N|/2$. Please note that shrinking results in a pseudo outer node. Unshrinking is done only on pseudo inner nodes, and this cannot be done more than $|N|/2$ times either (without augmenting the matching). Thus, in polynomial time, we must either find that the outer nodes form a Hungarian set in G_s or augment the matching. Thus, the whole algorithm is strongly polynomial and finds the optimal perfect matching when one exists and shows that none exists when this is the case.

This completes our description of the matching algorithm. We will discuss some "applications" of this problem and its extensions.

1.4 Applications:

1. **2-machine UET Shop**: In this problem, we have n jobs each of which has to be processed on one of two identical machines A and B. All the

processing times are equal (and hence taken to be 1). In addition, there are precedence constraints on the jobs. *It is assumed that no preemption is allowed.* We want to minimize the makespan which is the total time required to finish all the jobs. In [FKN] is a formulation of this problem as a cardinality matching problem as follows: Let a graph G be defined as $G = [N, E]$ where the nodes correspond to the jobs and edges to pairs of jobs which can be processed simultaneously (have no direct or implied precedents). Clearly, in the final schedule jobs that are processed in the same time slot (since all processing times are equal to 1, we can define these entities) will correspond to nodes that are joined by an edge in G. The length of the schedule is $|N|$ – the number of slots with two jobs. Hence, the problem of minimizing schedule length is equivalent to maximizing the number of slots with pairs of jobs. Clearly, there is a matching corresponding to any feasible schedule in that the edges correspond to pairs of jobs that occupy the same slot. If we show the converse, then the formulation will be complete. To do this, let M be a matching. If there are single jobs (those that are left exposed by the matching) which have no predecessors, then process them in the beginning of the schedule. If there is a pair of jobs (corresponding to an edge in the matching) both of which have no predecessors, then process these in the first time slot. If neither is true, then we modify the matching to another with the same cardinality in which one of these conditions is satisfied. This process can be repeated, and we get a corresponding schedule. For this purpose, let there be no single job or any pair corresponding to an edge in the matching without predecessors. Let job 1 be a job without predecessors ($\exists$ such a job if there is a feasible schedule), and because of the above conditions, let 1 be paired with 2 in the matching. Since 2 has predecessors, let 3 be the job obtained by following the chain of predecessors of 2 such that 3 has no predecessors (again $\exists$ such a job). Job 3 is paired with some other job, say 4. 4 is clearly different from 1 and 2. If 2 and 4 can be processed at the same time slot, then interchange the edges $(1, 2)$ and $(3, 4)$ to form a new matching $(1, 3)$ and $(2, 4)$. Now the pair $(1, 3)$ is without predecessors and can be processed first. If not, we get another pair$(5, 6)$ of this type and at the end of this process, we get the interchange of the type above. Thus, corresponding to any matching (after some interchanges) we get a schedule with as many pairs as there are edges in the matching. Thus, the formulation is correct.

2. **Odd/Even Shortest Routes**: Given an undirected graph $G = [N, E]$ whose edges have positive lengths, we want the shortest among paths between s and t that have an odd/even number of edges. To formulate the problem as a matching problem define a graph $H = [N \cup N', E \cup E' \cup W]$ where for each $i \in N$ $\exists$ its *clone* $i' \in N'$ and for each edge $e = (i, j)$ with $i, j \in N \exists$ its clone $e' = (i', j')$ and these together from E'. W consists of

all edges in H of the form (i, i'). These W edges have $c_e = 0$; all other edges have the same weight as their original lengths or the length of the edge of which it is a clone. Now depending on whether we are interested in paths with odd or even number of edges we remove the pair of nodes $\{s', t'\}$ or the pair $\{s', t\}$ from H and let the resulting graph be denoted by F. Now we solve the perfect matching problem in F to get the optimal path. This is done by deleting from this optimal matching edges of the type (i, i'). It is easy to see that because all lengths are positive, the edges in the optimal matching other than those that from the path of the desired type the only edges are of the type (i, i'). Thus, the equivalence is established.

3. **The Chinese Postman Problem**: This problem has a very interesting history and is indeed considered by many to be one of the first problems of graph theory. It deals with **Euler's** solution of the famous *Königsberg bridge problem*. Euler showed that in order to be able to traverse all the edges of an undirected graph in a continuous walk without repeating any edge the graph must have even degree at each node and be connected. Such graphs are now called *Eulerian graphs*. The corresponding condition for directed graphs is that the graph be strongly connected and the indegree at each node equal its outdegree. If an undirected graph has two nodes whose degrees are odd, then such a traversal is possible if we start at one of these nodes and are allowed to finish at the other. If this number exceeds two (and it is known that this number is always even), then in order to traverse all the edges, we will be forced to repeat some edges and the question is of repetition of minimal total length. Similarly, if such traversal is impossible for a directed graph, repetition is necessary and the optimal repetition, in this case, can be found by solving a transportation problem after some "preprocessing" using shortest route algorithms. The problem of repetition of some edges so as to make an undirected graph Eulerian with minimal repetition is what we solve by matching now. First, we solve the problem of determining the shortest paths between all pairs of nodes (for this purpose, we assume that lengths of all edges in the original graph to be positive). Now we define another graph H whose nodes are the nodes in G whose degree is odd. Each pair of these is connected in H, and the length of the edge $(i,\ j)$ in H is the length of the shortest path between i and j in G. In H, we solve the perfect matching problem. We repeat the paths in G that correspond to the edges in the optimal matching in H, and this is the required solution for the Chinese Postman problem in G. To prove this algorithm "works," we state the following results. No edge of G is traversed more than twice in the optimal traversal. Indeed, we can strengthen this result, too. If we let $Q = [N, D]$ where D is the set of edges that are repeated, then $\exists$ no cycle in Q. Thus, Q is *forest*. Further, the parity of the degree in G and Q is the same for all nodes (i.e.,

odd nodes in G are odd in Q and even ones are even). This forest can be decomposed into paths between odd nodes which are disjoint in the edges they use as well as their tip nodes. Thus, they form a matching in H. Hence, the result follows.

4. **Bidirected Flows**: Consider the problem:

$$[\min cx : x \geq 0; Ax = b; x \text{ integer}] \tag{1.9}$$

where A is a matrix whose entries $a_{i,j}$ integral. If A satisfies the condition $\sum_i |a_{i,j}| \leq 2 \; \forall \; j$, then the problem above is called a *bidirected flows problem*. Of course, this implies that the elements of $A \in \{0, \pm 1, \pm 2\}$, and there are, at most, two nonzeroes in every column of A, and if there are two, both of them must be ± 1. It is usual to assume that the variables have upper and lower bounds in addition. (Otherwise such bounds can be found on the optimal values of the variables). It is an easy exercise to show that it is possible to convert all entries to $+1$'s and satisfy these conditions at the same time. Problem of this type is called a b-matching problem. In many applications, variables in the b-matching problem will be between 0 and 1. In these cases, we can also make $b_i = 0$ or 1 by suitable transformations. All this without affecting the size in a nonpolynomial manner. At this point, we have a regular matching problem of the type we have considered so far. One good example of this type is the shortest route problem in undirected networks allowing negative edges but no negative undirected cycles. The usual trick of replacing an edge by two arcs with opposite directions will immediately create negative cycles in the corresponding directed version. We have no other means of solving this problem polynomially except through the use of matching algorithms. Here then is the formulation: Let

$$\begin{gathered} X_e = \begin{cases} 1 & e \in P_{s,t} \\ 0 & e \notin P_{s,t} \end{cases} \\ \sum_{e:e=(i,j)} x_e = \begin{cases} 1 & i = s,t \\ 2\alpha_i & else \end{cases} \\ 0 \leq \alpha_i \leq 1 \; \forall \; i \in N \\ \min \sum_e c_e x_e \end{gathered} \tag{1.10}$$

Please note that this is a bidirected flows problem and, hence, can be converted to matching since the bounds are $0, 1$.

5. T-joins and T-cuts: This application is very useful in multicommodity flows. In particular, much of the work on planar networks etc. is based on this concept. it is taken up in a separate section later on.

This completes our description of the applications although there is a problem in geometry dealing with decomposition of a simple polygon into minimal number of convex polygons (see Chazelle and Dobkin).

We now introduce the related problem of finding an independent set of nodes that has the maximum weight. Given an undirected graph $G = [N, E]$, a subset S of nodes is called an *independent set* if $\exists$ no edge connecting a pair of nodes in S.

1.4.1 Problem III:

Given $G = [N, E]$ and node weights v_i, find an independent set with maximum weight. This problem in general is *NP-hard*. If G is *perfect*, then it is nicely solvable. The connection of this to the matching problem is established in the following result.

Lemma 8 *The matching problem is a special case of the independent set problem.*

Proof: Given a weighted matching problem on $G = [N, E]$ define another graph $L(G) = [E.F]$, *(called the line graph of* G*)* whose nodes correspond to the edges of G and there is an edge in $L(G)$ for each pair of edges in G which have a common node. With this definition, it is clear that there is a one-to-one correspondence between matchings in G and independent sets in $L(G)$ and hence the equivalence of the two problems. Hence, if we know the original graph G of a line graph $L(G)$, then by solving the matching problem on G, we can solve the independence set problem on $L(G)$. Given a graph L, there is a polynomial algorithm that will test if this is a line graph of some graph and produce that graph if the answer is affirmative. Hence, this is nicely solvable case of the independent set problem. We now take up an extension to claw free graphs by **Minty.**

Chapter 2

Claw Free Graphs:

An undirected graph is said to be *claw-free* if there is no $K_{1,3}$ as a subgraph. The following result shows that this class of graphs includes line graphs.

Lemma 9 *Line graphs are claw-free.*

Thus, the independent set problem on claw-free graphs includes those on line graphs and hence matching problems. However, there are claw-free graphs that are not line graphs and hence this is a proper generalization of the matching problem. The methods used to solve this problem uses the algorithm of matching as a subroutine.

What we now describe is due to **G. Minty** and **N. Sbihi.**