The Demand Matching Problem

Bruce Shepherd[1] and Adrian Vetta[2]

[1] Bell Laboratories
bshep@research.bell-labs.com
[2] McGill University
vetta@math.mcgill.ca

April 18, 2006

Abstract. We examine formulations for the well-known b-matching problem in the presence of integer demands on the edges. A subset M of edges is feasible if for each node v, the total demand of edges in M incident to v is at most b_v. We examine the system of star inequalities for this problem. This system yields an exact linear description for b-matchings in bipartite graphs. For the demand version, we show that the integrality gap for this system is at least $2\frac{1}{2}$ and at most 2.764. For general graphs, the gap lies between 3 and 3.264. We also describe a 3-approximation algorithm (2.5 for bipartite graphs) for the cardinality version of the problem. A fully polynomial approximation scheme is also presented for the problem on a tree, thus generalising a well-known result for the knapsack problem. Recently, the notion of demand matching has arisen in the design of Clos network switches. [3]

1 Introduction

A combinatorial maximum packing problem is determined by a ground set V, each element u of which has an associated *profit*, denoted p_u, and a collection $\mathcal{F}$ of *feasible* subsets of V. The objective is to find a feasible set $F \in \mathcal{F}$ inducing a profit $p(F) \equiv \sum_{u \in F} p_u$ of maximum value. Feasibility for such problems is often determined by some set of resource capacities into which at most some bounded number of ground elements can be packed. For instance, in the b-matching problem on a graph, a subset M of its edges is feasible if each node v is incident to at most b_v edges of M. We are interested in understanding the relationship of such *base problems*, where ground elements in effect have a unit size, to their *demand version* where the elements each come with an integer demand value. For instance, in the case of b-matchings, each edge e could be additionally supplied with a demand d_e. A set M of edges is then feasible if for each node v, the total demand of edges in M incident to v does not exceed

[3] A preliminary version of this paper appeared in the Proceedings of the $9th$ International Conference on Integer Programming and Combinatorial Optimisation, Springer-Verlag, 2002.

the capacity b_v. Naturally, this gives rise to a completely different collection of feasible sets.

Analysis for such demand problems often requires new methods from those used for the base problem itself. This is largely because the whole demand must be satisfied before its profit may be reaped. Such "all-or-nothing" constraints arise quite naturally in practice; bandwidth trading in communication networks is one such example. In this paper our attention is restricted exclusively to the case of hard capacity constraints. The quintessential starting place for this study is the knapsack problem. Here an integrality gap of 2 for the standard linear programming (LP) formulation is well-known. Multiple knapsack problems, where the set of packable items is the same for each knapsack, have also been studied from the perspective of providing good approximation algorithms, see [5]. Our interest however lies in understanding how the structure of some base combinatorial problem interacts with its demand version. The demand b-matching problem, for instance, can be viewed as having many parallel knapsacks, one associated with each node. Coupling of the constraints only arises due to each edge being in two different knapsacks.

Theoretical work on the demand versions of combinatorial packing problems has been rather limited. We review now some of this work. One well-studied case of the generalized assignment problem is where each task has a size independent of its assignment. This problem is a special case of the demand matching problem. To see this, note that the base problem can be viewed as bipartite b-matching problem with the following property. Every node v on one side of the bipartition has a common demand value (associated with its task) on each of its incident edges, and its b_v-value is set to be equal to this demand value. The generalized assignment problem, in full generality where sizes may vary for a fixed task, was studied by Shmoys and Tardos [23], even though their focus was on congestion minimization. In fact, some of our techniques and findings bear resemblance to theirs. Their results are revisited and used in [5] where the maximization form is studied.

A demand version of network flows was studied by Cosares and Saniee (cf. the ring loading problem [8]) and by Kleinberg [16] who gave the first general and comprehensive study of the topic. In particular, Kleinberg introduced the term *unsplittable flow* and, relevant to the study in the present paper, he was the first to examine the maximization form of these problems. One example is the study [17] of the maximum single-source unsplittable flow problem. Here, the base combinatorial problem is that of packing paths into an edge-capacitated network. In the single-source setting, one is also given a source s and a collection of terminals $t_1, t_2, \dots, t_k$ with demands $d_1, d_2, \dots, d_k$, respectively. In addition, it is common to require a *no-bottleneck condition* so that each d_i is at most the minimum edge capacity in the network. The packing problem is to satisfy the maximum number of the demands subject to the edge capacity constraints. This problem may be viewed in the "demand" framework as follows. Let each $s - t_i$ path have the demand d_i. If we add a sink node t and edges $t_i t$ of capacity d_i, then the goal is to find a maximum packing of the weighted $s-t$ flow paths. This single-source problem has been further developed in [18, 9, 24].

Unsplittable flow with general sets of commodities has also been studied. For instance, in [19] the first approximation algorithm for general capacitated maximum unsplittable flow is given. Special classes of graphs have also been examined. Various special cases of the problem of packing subpaths (with demands) in an edge-capacitated path are studied in [12, 1, 3], although a constant factor approximation for the general problem first appeared in [4]. This was improved and extended to a constant factor (48) approximation for the maximum profit unsplittable flow problem where the underlying graph is a tree [6]. We mention that the maximum unsplittable flow problem on the tree, without the no-bottleneck assumption, also includes the demand matching problem, in the case where the tree is a star. Namely, the demand matching graph may be represented by demand edges between the leaf nodes of the star, and the node capacities are simulated by the edge capacities in the tree.

A traditional attack in solving general packing problems is to find a linear description for the convex hull of incidence vectors of feasible sets. A normal starting point is to formulate an integer program based on some $m \times n$ $0,1$ matrix A: $\max\{\mathbf{p} \cdot \mathbf{x} : A\mathbf{x} \leq \mathbf{b}, x_i \in \{0,1\}\}$ where any $0,1$ solution identifies a feasible set. For example, if G is bipartite, then its node-edge incidence matrix is totally unimodular. Hence this matrix immediately yields such a formulation for the b-matching problem [14] whose relaxation has integrality gap of 1. The demand version of the original $0,1$ packing problem is then: $\max\{\mathbf{p} \cdot \mathbf{x} : A^d\mathbf{x} \leq \mathbf{b}, x_i \in \{0,1\}\}$, where A^d is obtained from A by multiplying each column i by the value d_i. In [19], Kolliopoulos and Stein develop a framework for relating such demand packing problems to $0,1$ packing problems. They refer to a *column restricted packing integer program* as one arising from such a matrix with all non-zero entries in any column the same, and with each $d_i \leq b_j$ for each i, j. [4] They show that bounds on the integrality gap for column restricted packing problems can be directly tied to bounds on the integrality gaps for general $0,1$ packing problems. The key proof method is called *grouping and scaling*, whose origin is in their earlier paper [18]. In fact it follows from their work that a bound of Φ on the integrality gap for a family of $0,1$ packing problems, translates to an $O(\Phi)$ gap for the demand versions of the same family. This is stated explicitly in [6] for $0,1$ packing problems arising from an arbitrary *fixed* matrix A. Specifically, the following result is given, whose proof is implicit in the work of [19]. For any class of so-called *closed* objective functions $\mathcal{W}$, if the worst $0,1$ integrality gap arising from a packing problem for A, with any right hand side b, and some objective $w \in \mathcal{W}$, is Φ, then the worst gap for any such demand version is at most 11.54Φ. temming from a fixed A and set $\mathcal{W}$. Several refinements are stated in [6] for application to other problems, and in particular for giving a

[4] We mention that without the latter no-bottleneck assumption, there are examples where demand versions have arbitrarily large integrality gaps even if the starting matrix is total unimodular; this is described for instance in [4] for the problem of packing subpaths (with demands) into a capacitated path.

unified handling, and some improvements, of results in [3, 1, 4]. The potential for applying grouping and scaling was missed in these papers.

We largely ignore the congestion minimization form of all of these problems. Connections between the congestion minimization and maximization problems seem not to be fully understood and are worth exploring. For multi-commodity flow where the supply graph is a tree, for instance, the congestion problem is trivial. Finding a maximum routable (obeying capacities) subset of the commodities, however, is not. Even the case of unit profits and demands requires a nontrivial analysis [13]; as mentioned, a constant gap for the general version is given in [6].

We close by mentioning a link between demand matchings and the design of a class of communication switches. The switch is modelled as a graph with a bipartition of its nodes $I \cup O$ representing its input and output ports. In addition, the edges of the graph come with (fractional) weights representing traffic amounts between corresponding ports. The goal is to schedule transmission of traffic for all edges, using a minimum number of time slots. This amounts to colouring the edges, so that each colour class is a demand matching (where each $b_v = 1$). An introduction to this area and some of its open problems can be found in [20, 10]. They consider a model where each node of the bipartite graph has maximum fractional degree of Δ. Moreover, they assume that at each node v, of fractional degree k say, the edges incident to v can be partitioned into k "bins" each of whose fractional weight is 1. The goal is to find a smallest constant c such that the edges can be partitioned into $c\Delta$ demand matchings (asymptotically). The current best such bound on c is 2.548 [7]. In [7] a more general version is studied where the bin condition is weakened and one only requires that the fractional weight of each edge is at most 1. The best lower bound for the switch problem is presently given by $c = \frac{5}{4}$. At first glance our integrality gap of 2.5 for demand matching in bipartite graphs would seem to suggest a 2.5 lower bound for the above problems. However, our lower bound examples do not satisfy the extra bin conditions from the switch design problem. In fact, the demands (weights) on our edges are not even bounded by 1. It is an interesting question to understand the colouring question when one wishes to partition into general demand matchings.

1.1 The Demand Matching Problem

Take a graph $G = (V, E)$ and let each node $v \in V$ have an integral *capacity*, denoted by b_v. Let each edge $e = uv \in E$ have an integral *demand*, denoted by d_e. In addition, associated with each edge $e \in E$ is a *profit*, denoted by p_e. A *demand matching* is a subset $M \subseteq E$ such that $\sum_{e \in \delta(v) \cap M} d_e \leq b_v$ for each node v. Here $\delta(v)$ denotes the set of edges of G incident to v. We assume, throughout, that the demand of any edge is less than the capacities of both its end-nodes; otherwise such an edge is not contained in any demand matching and may be discarded. The demand matching problem is to find a demand matching of maximum profit. It can be formulated as the following integer program.

$$\begin{array}{lrcll} & \max \sum_{e \in E} p_e \, y_e & & & \\ \text{s. t.} & \sum_{e \in \delta(v)} y_e \, d_e & \leq & b_v & \forall\, v \in V \\ & y_e & \in & \{0,1\} & \forall\, e \in E \end{array} \tag{1}$$

We also associate with an edge e a value, $\pi_e = p_e/d_e$, which we call the *marginal profit* of that edge. Marginal profits play an important role in the understanding of demand matchings and, as a result, it will be useful to reformulate the integer program (1) in terms of marginal profits. Doing so we obtain the following integer program.

$$\begin{array}{lrcll} & \max \sum_{e \in E} \pi_e \, x_e & & & \\ \text{s. t.} & \sum_{e \in \delta(v)} x_e & \leq & b_v & \forall\, v \in V \\ & x_e & \in & \{0,d_e\} & \forall\, e \in E \end{array} \tag{2}$$

There are several special cases of the demand matching problem that are interesting in their own right. We begin by considering specific demand and profit functions.
(i) Unit Demand Function: the demand of each edge is one. Observe that this problem is just the familiar b-matching problem. Hence, the unit demand version can be solved in polynomial time.
(ii) Maximum Cardinality Demand Matching Problem: the profit associated with each edge is one, and hence the objective is to find a feasible demand matching containing as many edges as possible.
(iii) Constant Marginal Profit Function: the profit associated with each edge is proportional to its demand. As a result the marginal profit of each edge is the same.

Particular underlying graphs also give rise to interesting subproblems. For example, suppose that the underlying graph is a *star*, i.e., a tree in which every node is a leaf except for the root node r. In this case, the demand matching problem is equivalent to the *knapsack problem*, where the knapsack capacity is b_r and there is an item of weight d_e for each edge e. In addition, if we have a unit marginal profit function, then the demand matching problem on a star includes the familiar *subset-sum problem*. As a consequence, we note that the demand matching problem is NP-hard even where G is a star.

1.2 An Overview of the Paper

The paper is organized as follows. In Section 2 we show that the demand matching problem, even in the cardinality case, is MAXSNP complete. In Section 3 the natural linear programming relaxation is studied. Here it is seen that the integrality gap for the formulation is at least $2\frac{1}{2}$ for bipartite graphs (whereas, it is 1 for the unit-demand case) and at least 3 for general graphs. We also discuss an extension of Berge's Augmenting Path Theorem which gives an optimality certificate for fractional demand matchings. Most of the remainder of the paper seeks upper bounds on the integrality gap of the linear program by way of approximation algorithms that turn fractional solutions into integral solutions. The basic scheme uses augmenting paths and is introduced in Section 4. It yields a 3-approximation for bipartite graphs and a $3\frac{1}{2}$-approximation for general graphs. In Section 4.3 it is seen that understanding the integrality gap is related to determining the fractional chromatic number of bipartite graphs where some of the edges have been subdivided. A randomised algorithm is then devised for this problem which improves the general bound for bipartite graphs to 2.764 (and 3.264 for general graphs). In Section 5 algorithms for the cardinality problem are presented with approximation guarantees of $2\frac{1}{2}$ for bipartite graphs and 3 for general graphs. Finally, in Section 6 we present a fully polynomial time approximation scheme for the problem in which the underlying graph is a tree; this generalises the well known result for the knapsack problem [15].

2 Hardness Results

We have already seen that the demand matching problem is hard for instances with unit marginal profit functions. However, we have also seen that instances with a unit demand function are polynomially solvable. In this section we examine the hardness of the demand matching problem in more detail. In particular, we show that the maximum cardinality demand matching problem (and hence the general demand matching problem) is MAXSNP-hard. Thus, there exists a constant $\epsilon > 0$ such the problem admits no $\frac{1}{1-\epsilon}$-approximation algorithm unless $P = NP$. We say that $\frac{1}{1-\epsilon}$ is the *inapproximability constant* for the problem.

Theorem 1. *The cardinality demand matching problem is MAXSNP-complete (even if the demands are restricted to be* 1 *or* 3*).*

Proof. We first give a reduction from the the stable set problem to the cardinality demand matching problem. Let (G, k) be an instance of the (decision) stable set problem; that is, we want to know whether or not G contains a stable (pairwise non-adjacent) set of nodes of cardinality k. We construct an instance G' of the (decision) cardinality demand matching problem such that G has a stable set of size k if and only if G' has a demand matching of size $2m + k$, where m is the number of edges in G. For each node v in G we have two nodes v and $\bar{v}$ connected by an edge in G'. In addition, for each edge uv in G there is a node (u, v) in G'. Between nodes v and (u, v) there is a path consisting of two edges. Similarly there is a a path consisting of two edges between the nodes u and (u, v). Thus, for each node v in G we have an associated gadget in G'; such a gadget is shown in Figure 1. Here v is adjacent to $u_1, u_2, \ldots, u_r$ in G, where r equals the degree, d_v, of v in G. We say that node v is *selected* by a demand

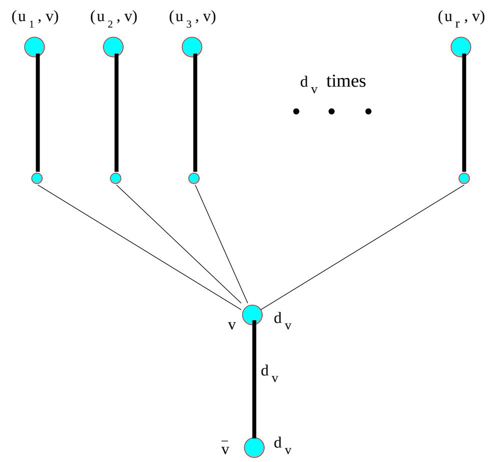

Fig. 1. A Gadget.

matching if the matching contains all the $d_v + 1$ "outer" edges from v's gadget (these outer edges are shown in bold in Figure 1). If the demand matching chooses the d_v "inner" edges from the gadget, then v is said to be *deselected*. If each node is either selected or deselected, then the size of the demand matching is $\sum_{v \notin S} d(v) + \sum_{v \in S}(d(v)+1) = \sum_v d(v) + |S|$, where S is the set of selected nodes.

To enforce these selection rules, we set the capacity of each node in a gadget to be 1 with the exception of v and $\bar{v}$ which have capacity d_v. In addition, every edge has demand 1, except for edges of the form $v\bar{v}$ which have demand d_v. This ensures that if $v\bar{v}$ is chosen, then none of the inner edges may be, and that only one of the two edges on the path from the node v to the node (u,v) may be chosen. One sees that any demand matching which contains $d(v)+1$ edges from a node v's gadget must use the edge $v\bar{v}$. Moreover, take a demand matching of size $2m+k$, for some k. Such a matching can easily be transformed into a demand matching, of the same or greater magnitude, in which each node is either selected or deselected. By our choice of capacities, we then have that the selected nodes form a stable set in G. Conversely, suppose S is a stable set of size k in G. Then the union of the outer edges from gadgets corresponding to nodes in S with the inner edges from gadgets corresponding to nodes not in S gives a matching of cardinality $2m+k$. Thus, we have the desired reduction.

We now complete the proof of MAXSNP completeness for the case of demands 1 and 4; similar arguments apply if we restrict to demands 1 and 3. Consider the maximum stable set problem in graphs with bounded maximum degree. Inapproximability results are given in [2] for such graphs; for graphs of maximum degree 4 it is hard to find a stable set within a factor of 1.0136 (roughly) of the optimum. So, suppose we have an $\frac{1}{1-\epsilon}$-approximation for the cardinality demand matching problem. Now take a graph G of maximum degree 4 with a maximum stable set of size k. Observe that $k \geq \frac{n}{5} \geq \frac{m}{10}$. In addition, our reduction produces a graph G' with maximum demand matching of cardinality $\text{OPT} = 2m + k$. Applying our approximation algorithm we obtain a demand matching of cardinality at least $(1-\epsilon)(2m+k)$. This corresponds to a stable set in G of size at least $(1-\epsilon)(2m+k) - 2m = (1-\epsilon)k - \epsilon \cdot 2m \geq (1-\epsilon)k - \epsilon \cdot 20k = k(1-21\epsilon)$. So $\frac{1}{1-21\epsilon} \geq 1.0136$ and hence the inapproximability constant for the demand matching problem is at least 1.00064.

It is straightforward to check that this approach also applies if we use cubic graphs, but a slightly weaker inapproximability constant results. □

We close by mentioning that it is now independently known [22, 25] that the maximum demand matching problem remains NP-hard in bipartite graphs, even when all demands are restricted to be either 1 or 2.

3 A Linear Programming Relaxation

We now consider the linear program relaxation of the formulation (2), i.e., for each edge we now have the linear constraints $0 \leq x_e \leq d_e$. We call the solution space of the resultant linear program the *fractional demand matching polytope.* We say that a point $\mathbf{x}$ in the polytope is a *fractional demand matching.* We also call x *basic* if it is an extreme point of the polytope. In this section we investigate the structure of this polytope and its extreme points; we apply the results later when we return to the integral problem.

3.1 Lower Bounds on the Integrality Gap

We first describe some lower bounds on the integrality gap of the linear programming relaxation. For trees the integrality gap is at least two (we will see in Section 4 that it is exactly two). This is well-known as there is a class of knapsack problems for which the fractional optimum is twice the integral optimum. An example is shown in Figure 2a, where the nodes are labelled by their capacities. For non-bipartite graphs, we now show that the integrality gap of the linear program is at least three asymptotically. Consider the simple example shown in Figure 2b. Note that no node can satisfy both of its incident edge demands. Thus, the optimal integral solution contains exactly one edge from the triangle. Since each edge has value $p_e = 1$ the optimal integral solution has value 1. However, consider setting each edge weight to be $x_e = k - 1$. This gives a feasible fractional demand matching with profit $3 - \frac{3}{k}$. Thus the integrality gap is at least three. Finally, for bipartite graphs, Figure 3 shows that the integrality gap is at least $2\frac{1}{2}$, even for cardinality instances. The optimal integral solutions contain exactly two edges. However, the fractional solution, shown in Figure 3, gives a profit of value $5 - \frac{4}{k} - \frac{k-1}{W}$, for some constants W and k. This is $5 - \frac{5}{k}$ if we set $W = k(k-1)$, giving the claimed gap as k becomes large.

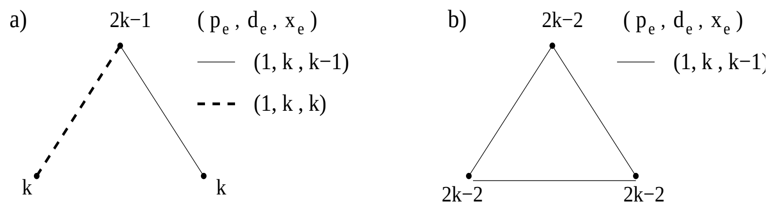

Fig. 2. Lower bound examples for trees and non-bipartite graphs.

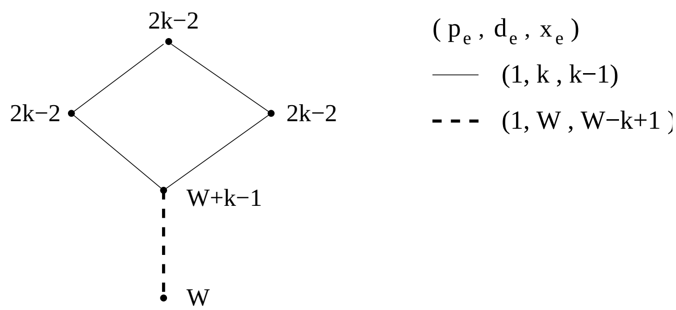

Fig. 3. A lower bound example for bipartite graphs.

3.2 Basic Solutions of the Fractional Demand Matching Polytope

For a fractional demand matching $\mathbf{x}$ a node v is termed *tight* if $\sum_{e\in\delta(v)} x_e = b_v$; otherwise the node is termed *slack*. We say that an edge e is *tight* if $x_e = d_e$; we say that e is *fractional* if $0 < x_e < d_e$. Let $F(\mathbf{x}) \subseteq E$ be the set of fractional edges induced by the fractional demand matching $\mathbf{x}$. Let $G(\mathbf{x})$ denote the graph induced by $F(\mathbf{x})$.

Lemma 1. *Let* $\mathbf{x}$ *be an extreme point of the demand matching polytope. Then each component of* $G(\mathbf{x})$ *consists of a tree plus (possibly) one edge. In addition, any cycle in* $G(\mathbf{x})$ *has odd length.*

Proof. First, suppose that $G(\mathbf{x})$ contains an even cycle $C = \{e_1, e_2, \ldots, e_{2k}\}$. For any ϵ, we define $\mathbf{y}$ by setting $y_{e_i} = x_{e_i} + (-1)^i\,\epsilon$ for each $i = 1, 2\ldots, 2k$ and $y_e = x_e$ for any other edge e. Similarly we define $\mathbf{z} := \mathbf{x} - (\mathbf{y} - \mathbf{x})$. Observe that, since the edges on the cycle are fractional, there exists a positive ϵ such that both $\mathbf{y}$ and $\mathbf{z}$ are feasible. We then obtain the contradiction that $\mathbf{x} = \frac{1}{2}(\mathbf{y} + \mathbf{z})$. Next suppose that some component contains two odd cycles C_1 and C_2. The cycles do not share an edge; otherwise they induce an even cycle. Thus C_1 and C_2 are edge disjoint and connected by a (possibly empty) path P. Observe that the edge set $C_1 \cup P \cup C_2 \cup P$ is an even length circuit. We then obtain a contradiction by similar reasoning to that given above except that increments along P are by $\pm 2\epsilon$. The result follows. $\square$

3.3 Berge Conditions for Fractional Demand Matchings

Berge's classic result on matchings asserts that a matching is of maximum cardinality if and only if it has no augmenting path. This result does not have a simple generalization for demand matchings. We can, however, extend this result to give optimality conditions for the fractional demand matching problem. We begin with some definitions. Suppose we partition some subset of the edges into two sets $\mathcal{P} \cup \mathcal{N}$. We call the resultant edge sets the *positive* edge set and the *negative* edge set, respectively. The marginal value $\pi(F)$ of a set of edges $F \subseteq \mathcal{P} \cup \mathcal{N}$ is just the sum of the marginal values of its positive edges minus the sum of the marginal values of its negative edges i.e., $\pi(F) = \sum_{e \in F \cap \mathcal{P}} \pi_e - \sum_{e \in F \cap \mathcal{N}} \pi_e$. A *balloon* is formed by an odd length cycle attached to a (possibly empty) path known as the *string*. A *dumbbell* is formed by two edge-disjoint odd length cycles connected by a (possibly empty) path known as the *rod*. A *structure* is either a path, an even length cycle, a balloon or a dumbbell.

A path P is called *augmenting* (with respect to $\mathbf{x}$) if there is a partition $\mathcal{P} \cup \mathcal{N}$ of E such that (i) Edges on the path are alternately positive and negative (or vice versa), (ii) $\pi(P) > 0$, (iii) If an endpoint v of the path is tight, then its incident edge is negative, (iv) None of the positive edges in P is tight. The motivation behind this definition is as follows. The first two conditions state that it is beneficial to *augment* around the structure i.e., add ϵ weight to the positive edges, remove ϵ weight from the negative edges. Such an augmentation maintains feasibility with respect to the node constraints at internal nodes in the path. The third condition ensures that, for small ϵ, the node constraints remain satisfied at the end nodes as well. Finally the fourth constraint ensures that, for small ϵ, the edge constraints also remain satisfied. Thus if we find an augmenting path we may improve our current demand matching.

Other structures may also be augmenting. An even length cycle is just a closed path of even length, and a dumbbell is an even length circuit in which the edges in the rod are traversed twice. Thus, augmenting cycles (and dumbbells) can be defined analogously to augmenting paths. Here, however, we may omit condition (iii). This is because in an even circuit all the nodes may be considered internal to the path. Observe that a balloon is just a closed odd length path in which edges on the string are traversed twice. Thus, augmenting balloons can be defined analogously to augmenting paths. Note that as the circuit is of odd length, some node, say v, is incident only to positive edges (or only to negative edges). We may think of v as the "endpoint" of the balloon and, thus, in defining augmenting balloons we do require condition (iii). It is also easy to see that it is beneficial to augment along any of these other augmenting structures.

Theorem 2. *A fractional demand matching is optimal if and only if it induces no augmenting structure.*

Proof. We have seen that in the presence of an augmenting structure the fractional demand matching can not be optimal. To prove the other direction take an optimal fractional demand matching $\mathbf{x}^*$. Given that $\mathbf{x}$ induces no augmenting structure we show that $\mathbf{x}$ gives the same profit as $\mathbf{x}^*$ and is therefore optimal.

Towards this end let $G' = \mathbf{x}^* \oplus \mathbf{x}$ denote the symmetric difference between $\mathbf{x}^*$ and $\mathbf{x}$. That is, the edge set of G' is $E' = \{e : x_e^* \neq x_e\}$. An edge e in E' is given a marginal value of π_e. In addition the edge set is divided into two classes M^* and M, where $M^* = \{e \in E' : x_e^* > x_e\}$ and $M = \{e \in E' : x_e^* < x_e\}$.

Choose an arbitrary node v in G'. Grow a maximal alternating path in the obvious way (where edges are alternately in M and M^*, or vice versa). If we discover an even length cycle or a dumbbell then stop. Similarly stop if we obtain a balloon with a maximal length string, or if both ends of the path can not be extended. Thus we obtain a structure S since some structure must be present. Note that $e \in M^*$ implies that $0 \leq x_e < x_e^* \leq 1$. In particular, this implies that x_e may be increased by ϵ and x_e^* may be decreased by ϵ, for a small positive value of ϵ. Now $e \in M$ implies that $0 \leq x_e^* < x_e \leq 1$ and similar comments apply. In addition, note that if an end-node v of the structure S is only incident to edges of M^* then v is slack with respect to the fractional demand matching $\mathbf{x}$, and vice versa. These observations imply that we may augment with respect to $\mathbf{x}^*$ in one direction along the structure and to augment with respect to $\mathbf{x}$ in the other direction (we abuse slightly the definition of augmentation as one of these augmentations has non-positive marginal value).

Consider the marginal value of this structure (where the edges in M^* are positive and the edges in M are negative). The structure is not augmenting with respect to $\mathbf{x}$ and so $\pi(S) \leq 0$. However, $\pi(S)$ can not be negative otherwise an augmentation with respect to $\mathbf{x}^*$ is beneficial, and hence $\mathbf{x}^*$ is not optimal. Thus $\pi(S) = 0$. So $\mathbf{x}^*$ and $\mathbf{x}$ give the same profit on the edges in S. We remove the edges in S from G' and search for another structure. It follows that $\mathbf{x}$ is optimal. □

4 Linear Programming based Algorithms

4.1 Transforming a Fractional Demand Matching: The Augmenting Paths Procedure

In this section, we present deterministic approximation algorithms for the demand matching problem. At the heart of these algorithms is a procedure which takes a fractional demand matching on a tree and transforms it so as to reveal a two colouring of the edges. Each coloring will induce a feasible demand matching thus giving a factor 2 approximation guarantee for the demand matching problem on a tree. This procedure is then used as a subroutine in an algorithm that gives a factor 3 guarantee if the underlying graph is bipartite, and a factor $3\frac{1}{2}$ approximation guarantee in general graphs.

As the algorithm for trees lies at the heart of the main algorithm, we tackle this specific case first. Take an optimal solution $\mathbf{x}$ to the linear programming relaxation of (2) on a tree. We show how to obtain an integral demand matching whose profit is at least half of that of the fractional demand matching $\mathbf{x}$. To do this we show that T contains two disjoint integral demand matchings whose combined profit is at least that of $\mathbf{x}$.

The algorithm is as follows. We may assume that any edge e with $x_e = 0$ has been discarded. Set $t = 0$ and let $\mathbf{x} = \mathbf{f^0} + \mathbf{h^0}$. Here $\mathbf{f^0}$ is the vector obtained by setting all edges not in $F(\mathbf{x})$ equal to 0, and $\mathbf{h}$ is the vector obtained by setting all the non-tight edges equal to 0. Whilst the forest $F(\mathbf{f^t})$ does not a form a standard 1-matching, take any tree T' in $F(\mathbf{f^t})$ that is not a single edge and choose two leafs i and j. Let $P = \{e_1, e_2, \ldots, e_k\}$ be the path in T' between i and j. For some small ϵ, we define $\mathbf{y}$ by setting $y_{e_i} = f^t_{e_i} + (-1)^i\,\epsilon$ for each $i = 1, 2 \ldots, k$ and $y_e = x_e$ for any other edge e. Similarly we define $\mathbf{z} := \mathbf{f^t} - (\mathbf{y} - \mathbf{f^t})$. The profit associated with either $\mathbf{y}$ or $\mathbf{z}$ is at least that of $\mathbf{f^t}$. Without loss of generality, we may assume that $\mathbf{z}$ is more profitable. We then set $\mathbf{f^{t+1}} = \mathbf{z}$. (Note that feasibility at one or both of the end-nodes of the path may be violated for the new vector $\mathbf{z}$. This will not be an issue for us, as we will ultimately partition the final result into 2 feasible demand matchings.) Clearly, we may choose ϵ so that either some edge becomes tight in $\mathbf{f^{t+1}}$ (in which case we remove the edge from $\mathbf{f^{t+1}}$ and add it to $\mathbf{h^t}$ to obtain a new set of tight edges $\mathbf{h^{t+1}}$), or some edge becomes zero and it is discarded.

We call the above process of modification of our linear programming solution the *augmenting paths procedure*. By construction, the value of the solution improved at each step. So the solution $\mathbf{f^{t^*}} + \mathbf{h^{t^*}}$ at the end of the augmenting paths procedure has profit at least that of the optimal fractional demand matching $\mathbf{x}$. Since an edge is removed from $F(\mathbf{f^t})$ at each step, we terminate in $|E|$ phases with a forest $F(\mathbf{f^{t^*}})$ that forms a 1-matching

As previously remarked, at some stage, $\mathbf{f^t} + \mathbf{h^t}$ may no longer be feasible. On any given path augmentation, however, the node constraints at internal nodes in the path continue to hold. It is only nodes at the ends of such a path which may become infeasible. For any such node, after the augmentation either it is still a leaf in the fractional subgraph, or the unique fractional edge incident to it became tight. In the latter case, this node will never participate in any more augmentations. Motivated by this observation, we say that an edge e is *bronze* with respect to v and some $\mathbf{f^t}$, if $e \in \delta(v)$ and either i) e is in $F(\mathbf{f^t})$ or ii) v is not incident to an edge in $F(\mathbf{f^t})$, and e was the final edge incident to v to become tight. Otherwise we say that e is a *copper* edge with respect to v. We call an edge *bronze* if it is bronze with respect to either of its endpoints, and call it *copper* otherwise. Note that a node v at termination of the procedure is incident to at most one edge that is bronze with respect to v. However it may be incident to several edges that are copper with respect to v and yet are bronze with respect to a neighbour of v.

We show how to use the copper-bronze edge colouring to find two feasible demand matchings on the supports of $\mathbf{f^{t^*}}$ and $\mathbf{h^{t^*}}$. This then immediately yields a factor 2 approximation algorithm. The following lemma will help us towards this goal.

Lemma 2. *The set of copper edges with respect to a node v are collectively feasible with regards to the associated node constraint.*

Proof. If there is no bronze edge with respect to v then all of the incident edges were tight in the initial linear program solution. Hence they are all collectively feasible at v. So let b be a bronze edge with respect to v. Let the copper edges with respect to v, ordered according to the order in which they became tight, be $c_1, c_2, \ldots, c_k$. Consider the edge c_i, and suppose it became tight whilst augmenting the path P. Note that the edge b must have been fractional at this time. Thus v was not a leaf node in some fractional tree. Therefore, v was an internal node on the path P and the associated node constraint remained satisfied after the augmentation. Thus, $c_1, c_2, \ldots, c_i$ were collectively feasible with respect to the node constraint at v. $\square$

The following theorem now shows that we have a factor 2 approximation algorithm for the case in which the underlying graph is a tree. (This is substantially improved in Section 6.)

Theorem 3. *A tree contains two disjoint demand matchings M_1 and M_2 whose combined profit is at least that of the optimal fractional demand matching.*

Proof. Apply the above procedure to a basic optimal fractional demand matching. The edges in the tree can now be partitioned into two feasible demand matchings as follows. By Lemma 2 a node v's capacity constraint is violated by a set of edges only if the set contains v's bronze edge plus at least one other edge of $\delta(v)$. In order to partition into two feasible demand matchings it is thus sufficient to two-colour the edges so that the bronze edge with respect to any node v receives a different colour from all other edges of $\delta(v)$. Such a colouring can be found greedily, starting from an arbitrary root node in the tree, and then working out towards the leaves. The combined profit of the resulting matchings is at least that of the optimal fractional demand matching and, therefore, at least that of the optimal demand matching. $\square$

Recall, by Lemma 1, that $F(x)$ is (almost) a forest for general graphs. This observation will allow us to use our algorithm for trees to obtain an approximation algorithm for general graphs. We first consider bipartite graphs.

Theorem 4. *There is a factor 3-approximation algorithm for the demand matching problem on bipartite graphs.*

Proof. Solve the linear programming relaxation to get a basic optimal solution. If the tight edges in the linear program provide at least one third of the profit of the optimal fractional solution (i.e., the value of the linear program) then we are done. Otherwise the fractional edges provide two-thirds of the total profit. Since the graph is bipartite, by Lemma 1, the fractional edges form a tree. Thus we can find an integral demand matching in the tree with at least half the value of the optimal fractional solution on that tree. This integral demand matching has, therefore, value at least one third that of the LP. $\square$

We may now consider non-bipartite graphs. We will need the following lemma.

Lemma 3. *Let $\mathbf{x}$ be a basic optimal demand matching and let C be an odd cycle in $F(\mathbf{x})$. Then C either contains an edge e with $x_e \leq \frac{1}{2}d_e$ or an edge e such that the set of tight edges in $\mathbf{x}$ plus e form a feasible demand matching.*

Proof. Take any odd cycle C induced by the fractional edges of $\mathbf{x}$. Let e be the edge in C for which $d_e - x_e$ is minimized. Let e be incident to e_1 and e_2 on this cycle. Observe that if $d_e - x_e \leq \min(x_{e_1}, x_{e_2})$, then the set of tight edges still form a feasible demand matching even with the addition of e as desired. Otherwise, suppose $d_e - x_e > x_{e_1}$, say. Then, since $d_e - x_e \leq d_{e_1} - x_{e_1}$, we have that $x_{e_1} \leq \frac{1}{2} d_{e_1}$. □

Theorem 5. *There is a factor $3\frac{1}{2}$-approximation algorithm for the demand matching problem on non-bipartite graphs.*

Proof. We again begin by partitioning the edge set induced by $\mathbf{x}$ into fractional edges and tight edges. We would like fractional edges to induce two demand matchings as before. However we may not be able to do this since the fractional edges may, by Lemma 1, induce components that contain a single odd cycle. Instead we obtain a total of four demand matchings by the following method.

Since the odd cycles in $F(\mathbf{x})$ are node disjoint we may apply Lemma 3 to remove one edge from each odd cycle. We either add each such edge to a new set S of edges with property $x_e \leq \frac{1}{2} d_e$, or to the set of tight edges. Again, since the odd cycles are node disjoint, the set S induces a matching and hence induces a demand matching. The set of tight edges (plus any edges added during this process) also form a feasible demand matching. Finally the remaining set of fractional edges induce a forest, from which we may obtain two demand matchings. Thus, we have partitioned the support of $\mathbf{x}$ into four demand matchings. Moreover, the edges in S have twice the profit that they contributed to the profit of $\mathbf{x}$. It follows that one of these demand matchings induces a profit of at least $\frac{2}{7}$ that of the linear programming solution. □

4.2 Augmenting Path Procedure in General Graphs: Extending the Edge-Colouring

The method of Section 4.1 produces a bronze-copper colouring of the fractional edges of the linear program solution (except for those edges that get discarded during the augmenting paths procedure). It will be useful in the following sections if we extend this edge colouring to include all the edges in the linear program solution. In particular, we will generate a 2-colouring of the edges for bipartite graphs, and a 3-colouring for non-bipartite graphs. Take a basic optimal fractional demand matching $\mathbf{x}$. So $\mathbf{x}$ induces a set of tight edges and a fractional subgraph. Each fractional component is a tree together (possibly) with an extra edge that induces an odd cycle. All of the tight edges induced by $\mathbf{x}$ are also termed copper with respect to both of their endpoints. The fractional edges are coloured bronze, copper and red by the following augmenting paths procedure. We again augment along leaf-to-leaf fractional paths, with the additional possibility that augments may be from a leaf to itself, via the odd cycle. Note that non-simple augmentations use increments of size $\pm 2\epsilon$ along the stem of the augmenting path, whereas along the odd cycle the increments are $\pm\epsilon$. As before, we choose the direction of the swaps so as to improve the profit of the fractional demand matching. In addition, we do not alter feasibility at a node unless it was the leaf of the augment.

Note that at some point in the process a fractional component may contain no leaves i.e., it is an odd cycle. If some edge e on the cycle has the property that $x_e \leq \frac{1}{2}d_e$ then we colour e *red* and remove it from the component. We then continue the augmentation procedure and colour the rest of the edges copper or bronze. If no edge has this property, then by Lemma 3, there is an edge e with the property that $d_e - x_e \leq \min(x_{e_1}, x_{e_2})$, where e is incident to e_1 and e_2 on the cycle. We colour e copper with respect to both its endpoints, remove it from the component and continue the augmentation procedure. Note that e_1, e_2 are now leaves in the fractional graph, and hence will eventually either be dropped from the solution, or be coloured bronze. The procedure terminates when the set of fractional edges forms a matching and all the edges are coloured. Note also that the red edges form a matching. Based on the remarks above, it is easy to check that Lemma 2 still holds, even with respect to the enlarged set of copper edges generated by this extended colouring. This observation will be needed in the subsequent sections.

4.3 To Stable Sets and a Randomised Algorithm

In this section we show that the case of bipartite graphs is qualitatively different from the case of non-bipartite graphs with respect to the linear programming relaxation. In particular, we show that the integrality gap of the linear program is strictly less than 3 for bipartite graphs. This we achieve via the use of a randomised algorithm. Before presenting the randomised algorithm, we first show how the edge colouring induced by the linear program allows us to recast our problem as a stable set problem. Consider first that we are in the bipartite case; hence, all the edges are coloured bronze or copper (for the non-bipartite case it will actually suffice to just remove the red edges).

Our stable set problem will be on a subgraph G' of the line graph induced by G. The nodes of G' are the bronze and copper edges of G; for simplicity, we also call the nodes in G' bronze and copper. The decision of whether to include an arc from the line graph in G' is based on the copper and bronze edge interactions; for clarity, we refer to arcs in G' and edges in G. Two nodes induce an arc in G' if their corresponding edges in G share an endpoint v and one of the edges is bronze with respect to v.

If we think of orienting bronze edges in G towards each node for which they are bronze, then we obtain a digraph of maximum in-degree one. Moreover, any cycle would be of length 2. It follows that the bronze nodes in G' induce a forest. The copper edges induce nodes of degree at most 2 in G', since for a copper edge uv it may be incident to at most one bronze edge incident to u and one to v. Note that it may not be possible to place uv in a demand matching with either of the bronze edges with respect to u and v. We avoid such conflicts if we restrict to edge sets of G corresponding to stable sets in G'. Indeed it follows from Lemma 2 that a stable set in G' corresponds to a demand matching in G.

We now present a randomised algorithm for the general demand matching problem in bipartite graphs. The algorithm gives an improved approximation guarantee of 2.764 for bipartite graphs. We work on the induced stable set problem in the associated graph G'. The randomised algorithm first selects a stable set from amongst the bronze nodes. It then greedily adds any copper node for which no bronze neighbour is in the stable set. The algorithm works as follows. Take the forest induced by the bronze nodes. For each tree in the forest pick an arbitrary root node r. We now give the nodes in the tree a $0-1$ labelling. First give r the label 1 with probability p, otherwise label it 0. We consider the other bronze nodes in the tree in increasing order of their distance to the root. Take a node v with parent u. If u has label 1, then give v the label 0. Otherwise, if u has label 0, then give v the label 1 with probability $\frac{p}{1-p}$ and the label 0 with probability $\frac{1-2p}{1-p}$. It is easy to check by induction on its distance from the root that, at the end of this labelling process, each bronze node has a probability p of being labelled 1. We associate the label 1 with throwing away the node. The nodes with label 0 form a collection of trees. Since trees are bipartite we obtain two stable sets from each tree and we choose at random one of these stable sets to be in our final stable set. Finally, each copper node is added into the stable set if neither of its two bronze neighbours is already chosen.

We will use the following property in analysing the above algorithm: given a path $P = \{v_0, v_1, \ldots, v_k\}$ in a tree T, the probability of any fixed $0-1$ labelling $\pi = \{\pi_0, \pi_1, \ldots, \pi_k\}$ of P is the same regardless of the initial choice of root node r in T (in fact, the probability of any given $0-1$ labelling of the whole tree is the same irrespective of the choice of root). First we claim that the probability of a given labelling is the same if either v_0 or v_k are chosen as the root. There are four possible pairs of labellings for v_0 and v_k. We consider the case $\pi_0 = 0$ and $\pi_k = 1$; the other cases are similar. Order the path $P = \{v_0, v_1, \ldots, v_k\}$ from left to right. Let $L_{0,1}$ be the number of times a node label changes from 0 to 1 (called a $0-1$ transition) if we traverse P from left to right. Let $R_{0,1}$ be the number of $0-1$ transitions if we traverse P from right to left. We define $L_{1,0}, R_{1,0}, L_{0,0}, R_{0,0}, L_{1,1}$ and $R_{1,1}$ similarly.

If v_0 is chosen as the root, then we have that v_{i+1} is a child of v_i. Letting l_i denote the label of node i, the probability of the labelling π on P is exactly

$$\Pr(l_i = \pi_0 = 0) \prod_{i=0}^{k-1} \Pr(l_{i+1} = \pi_{i+1} | l_i = \pi_i) = (1-p)(\frac{1-2p}{1-p})^{L_{0,0}}(\frac{p}{1-p})^{L_{0,1}} 1^{L_{1,0}}$$

since the probability of a $0-1$ transition is $\frac{p}{1-p}$, the probability of a $0-0$ transition is $\frac{1-2p}{1-p}$, and the probability of a $1-0$ transition is 1. Similarly, if v_k is chosen as the root then we have that v_i is a child of v_{i+1}, and the probability of the labelling π on P is exactly

$$\Pr(l_k = \pi_k = 1) \prod_{i=k-1}^{0} \Pr(l_i = \pi_i | l_{i+1} = \pi_{i+1}) = p(\frac{1-2p}{1-p})^{R_{0,0}}(\frac{p}{1-p})^{R_{0,1}} 1^{R_{1,0}}$$

Now as $\pi_0 = 0$ and $\pi_k = 1$, it is easy to see that $L_{0,1} = R_{0,1} + 1$, $L_{1,0} = R_{1,0} - 1$, $L_{0,0} = R_{0,0}$, and $L_{1,1} = R_{1,1} = 0$. The claim then follows from the fact that

$$\Pr(l_i = \pi_0 = 0)\frac{p}{1-p} = (1-p)\frac{p}{1-p} = p = \Pr(l_k = \pi_k = 1) \cdot 1$$

Moreover, given this claim it is easy to show that the probability of a fixed labelling π on P is also the same for any other choice of root node in T.

We now analyse the performance of this algorithm using the choice of $p = \frac{1}{10}(5 - \sqrt{5}) \approx 0.276$.

Theorem 6. *There is a polytime randomised algorithm that provides a factor* 2.764*-approximation guarantee for the demand matching problem in bipartite graphs.*

Proof. To show this we calculate the probability that a given node is chosen as part of the stable set (i.e., the probability that an edge is in the demand matching). First, note that each bronze node is not removed with probability $1 - p$. Thus, since its bipartition class is chosen with probability $\frac{1}{2}$, it is in the stable set with probability $\frac{1}{2}(1-p) \approx 0.3618$. We now consider the copper nodes. Suppose that a copper node c is adjacent to bronze nodes in different trees in the bronze forest. The discarding of bronze nodes is independent in different trees and therefore the probability that the copper node can be selected is $p^2 + p(1-p) + \frac{1}{4}(1-p)^2 \approx 0.4073$.

The remainder of the proof concerns the case where the copper node c is incident to two bronze nodes in the same tree of the bronze forest. The neighbours of the copper node, say u and v, induce a unique path in the bronze forest. Recall that we have an arc between two nodes in G' if the corresponding edges in G share an endpoint v and one of the edges is bronze with respect to v. It follows that it is not possible for the bronze neighbours of a copper node to be adjacent in a bronze tree. Hence this path in G' must contain at least three bronze nodes. To see this in more detail, let us examine what a path in G' from u to v corresponds to in G. Such a path is shown in bold in Figure 4b. This comes from the graph G shown in Figure 4a; here an edge $e = (i, j)$ is shown as pointing from i to j if edge e is bronze with respect to node j. So a path from u to v in G' corresponds to two directed paths in G ending with arcs u and v that share their first edge but are otherwise disjoint.

As we have seen we may assume without loss of generality that u is the root node. All the possible node $0-1$ labellings for paths of three nodes are shown in Figure 5, and for paths of four nodes in Figure 6. Associated with each labelling is a pair of probabilities (α, β), where α is the probability that the labelling occurs and β is the probability that (given this labelling) the copper node can be placed in the resultant stable set. For example, the probability α of the $0-0-0$ labelling is $(1-p) \cdot \frac{1-2p}{1-p} \cdot \frac{1-2p}{1-p} = 0.2764$. To calculate the probability β, note that we have the following possibilities. If both u and v are discarded (given label 1), then with probability 1 we can add c to the stable set. If exactly one of u and v is discarded, then with probability $\frac{1}{2}$ we can add c to the stable set, since any bipartition class is chosen with probability $\frac{1}{2}$. If

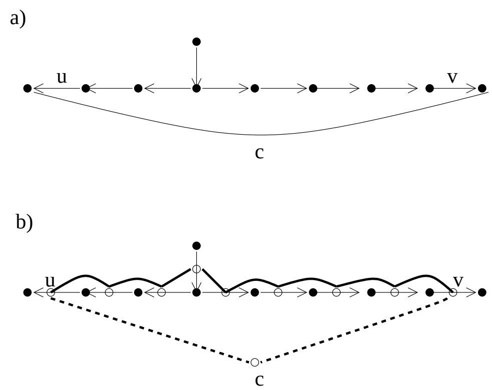

Fig. 4.

neither u nor v is discarded, then we have two possibilities. If some node on the path between u and v has been discarded, then they are in different trees and so with probability $\frac{1}{4}$ we can add c to the stable set. So consider the situation when no node on the path between u and v has been discarded. If u and v are in the same bipartition class then with probability $\frac{1}{2}$ we can add c to the stable set; if u and v are in different bipartition classes then we cannot add c to the stable set. For this reason, there is a lower probability of adding c if the path from u to v contains an odd number of edges. For example, for the paths with three nodes the probability that we can select edge c is at least 0.4839 and for the paths with four nodes the probability is at least 0.3618.

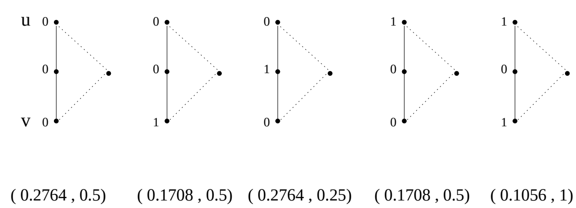

Fig. 5.

As the path length increases, the probability we can select the copper node converges to the probability $p^2 + p(1-p) + \frac{1}{4}(1-p)^2 \approx 0.4073$, i.e., the case in which u and v belong to different bronze trees. This is because, as the path length between u and v increases, the probability that some node on the path is discarded tends to one. We conjecture that for paths with an odd number of nodes we converge monotonically to 0.4073 from above, whereas for paths with an even number of nodes we converge monotonically to 0.4073 from below. For our purposes, however, it suffices to prove that the worst case probability arises

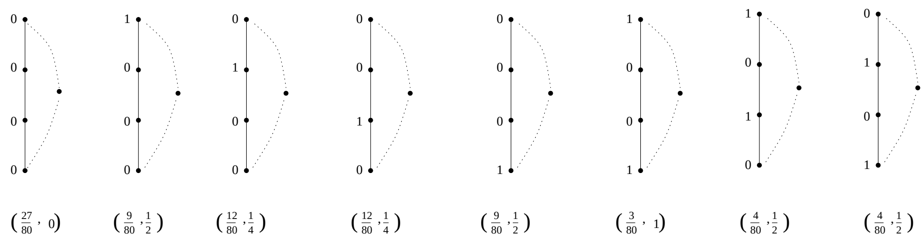

Fig. 6.

for paths with four nodes. It is easy to verify computationally that this is the case for path lengths of cardinality at most 16, say.

For longer paths, observe that the probability that all the nodes on the path between u and v receive label 0 is less than 0.001. If this does not occur then u and v are in different trees after discarding nodes on the path with label 1. We know that u and v themselves receive label 1 with probability p. The probability of being able to choose c is then minimised if u and v never receive label 1 at the same time; that is the label pairings $(0,1),(1,0)$ and $(0,0)$ occur with probability p, p and $1-2p$ respectively. In this case we may choose c with probability $\frac{1}{2}p + \frac{1}{2}p + \frac{1}{4}(1-2p) = \frac{1}{4} + \frac{1}{2}p \approx 0.3882$. It follows that that the worst case probability does arise for paths with four nodes. Hence the overall probability that the copper node c is placed in the stable set is at least 0.3618. Thus, each node is in the stable set with probability at least 0.3618. Therefore, we obtain an approximation guarantee of $\frac{1}{0.3618}$ which is at most 2.764. □

This result shows that the integrality gap of the integer program is strictly less for bipartite graphs than non-bipartite graphs. Moreover, the analysis above may be applied directly to non-bipartite graphs.

Theorem 7. *A randomised algorithm provides a factor* 3.264*-approximation guarantee for the demand matching problem in non-bipartite graphs.*

Proof. Recall that, after the generalised augmentation procedure is completed, the red edges have the property that $x_e \leq \frac{1}{2}d_e$. Let the value of the fractional solution after augmentation be v^*. If the demand matching induced by the red edges has value less than $0.3064v^*$, then the value of the copper and bronze edges is at least $0.8468v^*$. Therefore, applying the randomised algorithm to the stable set problem induced by the copper and bronze edges produces a solution whose expected value is at least $0.3618 \times 0.8468v^* = 0.3064v^*$. □

5 The Cardinality Problem

Recall that in the case of unit profits per edge our goal is to find a maximum cardinality demand matching. For the maximum cardinality problem we show that approximation guarantees of $2\frac{1}{2}$ and 3 are obtainable for bipartite and non-bipartite graphs, respectively. Hence, for the cardinality problem these factors coincide with the general integrality gaps of the corresponding linear program relaxations for bipartite and general graphs respectively. We begin with the bipartite case. From the exposition of the previous section, it is sufficient to find, in polynomial time, a stable set in G' that contains at least two-fifths of the nodes of G'. We show this can be done in a greedy fashion. In what follows we divide the arcs of G' into two classes: *tree* arcs are those arcs in the forest formed by the bronze nodes; the arcs incident to copper nodes are termed *non-tree* arcs.

Theorem 8. *In bipartite graphs, a greedy algorithm finds a demand matching whose size is at least $\frac{2}{5}$ of the size of the optimal fractional demand matching.*

Proof. Our aim here is to obtain a large stable in G'. To do this we show there is a stable set S in G' with the following properties. Firstly, the neighbour set $\Gamma(S)$ of S has size at most $\frac{3}{2}|S|$. Secondly, the removal of $S \cup \Gamma(S)$ from G' induces a graph of the same structure as G', i.e., the composition of a forest of bronze nodes with copper nodes adjacent to at most two bronze nodes. This second property allows us to repeatedly find such subsets, and the first property ensures that the resultant stable set is large. The theorem then follows. Naturally, we may assume that there are no isolated nodes at any stage, as we could add them directly to our stable set. Moreover, if any copper node c has degree exactly one, then we may set $S = \{c\}$.

We will use the notation $\Gamma^B(v)$ and $\Gamma^C(v)$ to refer to the set of bronze and copper neighbours of a node v respectively. Observe that $\Gamma^C(v) = \emptyset$ if v is a copper node. To begin, suppose there is a bronze node b incident to two (or more) copper nodes, that is $\Gamma^C(b) = \{c_1, c_2, \ldots, c_k\}$. We will grow a "breadth first search" tree from b to find a suitable stable set. Call $B_0 = \{b\}$ and $C_1 = \Gamma^C(B_0)$. Then let $B_1 = \Gamma^B(C_1) - B_0$ and $C_2 = \Gamma^C(B_1) - C_1$. More generally, we let $B_{i+1} = \Gamma^B(C_{i+1}) - B_i$ and $C_{i+1} = \Gamma^C(B_i) - C_i$. Since, by this breadth first search construction, any copper node is adjacent to bronze nodes either at the same level or at consecutive "bronze" levels, the sets $B_0, C_1, B_1, C_2, B_2, \ldots$ are disjoint. Moreover, at some stage this process must terminate with either $C_t = \emptyset$ or $B_t = \emptyset$ for some t. Now no two copper nodes are adjacent, and so $S = \cup_{1 \le i \le t} C_i$ is a stable set. In addition, observe that $|B_i| \le |C_i|$ as any copper node has at most one bronze neighbour in B_i. Since b is adjacent to at least two copper nodes, it follows immediately that $\Gamma(S) = \cup_{0 \le i \le t} B_i \le 1 + |S| \le \frac{3}{2}|S|$, as required.

So we may assume that no bronze node has two or more copper neighbours. Now take a leaf node v, with neighbour u in a bronze tree. We have four cases to deal with:

i) Neither v nor u is incident to a copper node. Select v to be in the stable set. Recurse on the graph $G' - \{u, v\}$. See Figure 7i).

ii) v is not incident to a copper node but u is. Call this copper node y. Then select v and y to be in the stable set. Recurse on the graph $G' - \{u, v, y, z\}$, where z is the other neighbour of y. See Figure 7ii).

iii) Both v and u are incident to copper nodes. Call these copper nodes x and y, respectively. Select v and y to be in the stable set. Recurse on the graph $G' - \{u, v, x, y, z\}$, where z is the other neighbour of y. Thus we select two nodes from five as required. See Figure 7iii).

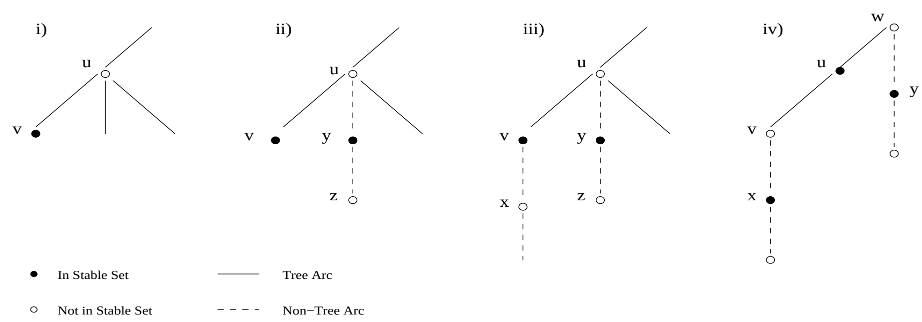

Fig. 7.

iv) v is incident to a copper node x but u is not incident to a copper node. If none of i) to iii) occur for any leaf node, then let v be of maximum depth (with respect to an arbitrarily chosen root node) in the bronze forest. We may then assume that u has degree two in the tree (with neighbour w). Otherwise, the children of u form a suitable set S. So, select u and x (and the copper neighbour y of w, if it has one) to be in the stable set and recurse on the graph obtained by removing u and x (and y) as well as their neighbours. See Figure 7iv). □

We now consider non-bipartite graphs.

Theorem 9. *A greedy algorithm provides a factor* 3*-approximation guarantee for the cardinality problem in non-bipartite graphs.*

Proof. Consider the set of red edges after the generalised augmentation procedure is completed. Each red edge e has the property that $x_e \leq \frac{1}{2}d_e$. Let the value of the fractional solution after augmentation be v^*. Suppose the contribution of the red edges to the value of the solution is at least $\frac{1}{6}v^*$. Then they form a demand matching of cardinality at least $\frac{1}{3}$ of optimum, since such edges satisfy $x_e \leq \frac{1}{2}d_e$. Thus we assume that the non-red edges contribute a value of at least $\frac{5}{6}v^*$. Observe that our cardinality algorithm for bipartite graphs did not rely directly on bipartiteness but only on the structure of copper and bronze edges. So we may apply this algorithm on the non-red edges to produce a demand matching of cardinality at least $\frac{1}{3}$ of optimum. □

6 An FPTAS for the Demand Matching Problem on a Tree

Recall that the knapsack problem can be formulated as a demand matching problem on a star. In addition, there exists a fully polynomial time approximation scheme (FPTAS) for the knapsack problem. In this section we describe an FPTAS for the demand matching problem when the underlying graph is a simple tree. As noted in [5], such an FPTAS is not possible even for a 2-node instance with multiple edges between the nodes. Our algorithm, based on a dynamic programming approach, turns out to be an exact algorithm in the unit-profits case.

Theorem 10. *There is a dynamic-programming-based FPTAS for the the demand matching problem on a tree.*

Proof. Let $\mathcal{I} = (T = (V, E), d, b, p)$ be such an instance. First, choose an arbitrary root node $v_n \in V$ and create an arborescence T by orienting all edges away from v_n. Let $v_1, v_2, \ldots, v_n$ be an inverse topological ordering of the nodes and, for each i, let T_i denote the subtree of T rooted at node v_i. We let h_i denote the *height* of the tree T_i.

For each $i < n$, let e_i denote the unique arc of T in $\delta^-(v_i)$. Also, for clarity let $b_i = b_{v_i}$. For each i, let $\mathcal{I}_i$ denote the instance obtained by restricting to T_i. Similarly, let $\mathcal{I}_i^-$ denote the instance obtained by restricting to T_i and reducing the capacity b_i by d_{e_i}. Let α_i and β_i denote the optima for the instances $\mathcal{I}_i$ and $\mathcal{I}_i^-$, respectively. The idea is to build up a solution for the whole problem by making choices between the optimal solutions to $\mathcal{I}_i$ and $\mathcal{I}_i^-$ at each node. For motivation, notice that if M is a feasible demand matching for $\mathcal{I}_i^-$, then $M \cup e_i$ is feasible for $\mathcal{I}$. Thus, the choice between the solutions β_i and α_i corresponds to the choice of whether or not we include the edge e_i.

Consider a non-leaf node v_i with out-neighbours v_{i_j}, $j = 1, 2, \ldots, r$. Let

$$S_i = \{i_j : \beta_{i_j} + p_{e_{i_j}} - \alpha_{i_j} > 0\}.$$

These are the indices where it is possibly beneficial to choose the edge e_{i_j}. We call these *candidates* for v_i. We now define a "local" knapsack problem at v_i. The items consist of the indices $i_j \in S_i$ and each has demand $d_{e_{i_j}}$ and profit value $\beta_{i_j} + p_{e_{i_j}} - \alpha_{i_j}$. The profit represents the "extra" value we can accrue by taking e_{i_j} into the demand matching. The knapsack itself has a variable capacity t which is later used to control whether or not we include e_i. Let $\text{OPT}(t)$ denote the optimum value for this knapsack problem, and let $\Delta(t)$ denote the set of indices chosen in some optimal solution. We claim that

$$\alpha_i = \text{OPT}(b_i) \; + \; \sum_{j=1}^{r} \alpha_{i_j} \tag{3}$$

$$\beta_i = \text{OPT}(b_i - d_{e_i}) \; + \; \sum_{j=1}^{r} \alpha_{i_j}.$$

Suppose that for some $j \notin S_i$, e_{i_j} is in a feasible demand matching M for $\mathcal{I}_i$. Then, since $\alpha_{i_j} \geq \beta_{i_j} + p_{e_{i_j}}$, we may discard the edge e_{i_j} and obtain a feasible demand matching of greater or equal value. It follows that $\alpha_i \leq \text{OPT}(b_i) + \sum_j \alpha_{i_j}$. A similar argument, with respect to $\mathcal{I}_i^-$, shows that $\beta_i \leq \text{OPT}(b_i - d_{e_i}) + \sum_j \alpha_{i_j}$. It is clear that the reverse inequalities also hold.

We now describe an algorithm that is built on repeatedly taking a non-leaf node in T and for some $\epsilon > 0$, using an FPTAS for the knapsack problem [15], to compute a feasible solution $\Delta'(t)$, of value $\text{OPT}'(t)$, to the "local" knapsack problem, with $\text{OPT}(t) \leq (1+\epsilon)\,\text{OPT}'(t)$. In general, we let $\text{FPTAS}(I)$ denote the result of applying a knapsack FPTAS to an instance I.

For each v_i we recursively define two subsets

$$\mathcal{A}_i = \bigcup_{j \in \Delta'(b_i)} (\mathcal{B}_j \cup e_j) \;\cup\; \bigcup_{j \notin \Delta'(b_i)} \mathcal{A}_j$$
$$\mathcal{B}_i = \bigcup_{j \in \Delta'(b_i - d_{e_i})} (\mathcal{B}_j \cup e_j) \;\cup\; \bigcup_{j \notin \Delta'(b_i - d_{e_i})} \mathcal{A}_j.$$

In particular, if v_i is a leaf, then $\mathcal{A}_i = \mathcal{B}_i = \emptyset$. Here $\mathcal{A}_i$ and $\mathcal{B}_i$ are the demand matching solutions that we have built up in approximating the solutions to α_i and β_i. Since $\mathcal{A}_i$ and $\mathcal{B}_i$ give only approximate solutions, α'_i and β'_i, to the true values α_i and β_i, the local knapsack candidate sets S_i may become distorted as we move up the tree. In particular, we are concerned when we "lose" candidate indices j, i.e., those j which become invisible because $\alpha_j < \beta_j + p_{e_j}$ but $\alpha'_j \geq \beta'_j + p_{e_j}$. We call such an index *lost*, and denote by L_i the set of lost indices for the local problem induced by v_i. We can no longer potentially gain from such lost indices when we solve the local knapsack problems. Thus there are two ways in which our solutions become suboptimal due to the repeated use of the FPTAS. One is due to the fact that α'_{i_j}'s and β'_{i_j}'s are only approximating the α_i's and β_i's for the candidate indices, and the second is due the subproblem distortions or lost indices themselves. We show, nevertheless, that this loss cannot be too great.

In the following, we use $\text{OPT}(t)$ (and α_i, β_i) to denote true optimal subproblem values on the true instances. We denote by $\overline{I}(t)$ the instance obtained by dropping all the lost candidates and for which the remaining candidates have their approximate α'_i, β'_i values.

We now show by induction on h_i, that

(I) $\alpha_i \leq (1+\epsilon)^{h_i}\,\alpha'_i$
(II) $\beta_i \leq (1+\epsilon)^{h_i}\,\beta'_i$
(III) $\text{OPT}(t) \leq (1+\epsilon)^{h_i-1}\text{OPT}(\overline{I}(t)) + \sum_{j \in L_i}((1+\epsilon)^{h_i-1}\,\alpha'_j - \alpha_j)$ for $t = b_i, b_i - d_{e_i}$.

If $h_i = 0$ the result is trivial. Suppose that claims (I), (II) and (III) hold for all $i < k$. We first show (III) and towards this we claim:

$$\text{OPT}(t) \leq (1+\epsilon)^{h_k-1}\,\text{OPT}(\overline{I}(t)) + \sum_{j \in L_k} \beta_j + p_{e_j} - \alpha_j. \tag{4}$$

An optimal solution in the original instance will be comprised of profit from the lost indices in L_k and profit from a set X from the other remaining indices. The former are accounted for in the second sum. We show that the profit from X in the new instance is not too much smaller. In particular, we see that for each $j \in X$, $\beta_j + p_{e_j} - \alpha_j \leq \beta_j + p_{e_j} - \alpha'_j \leq (1+\epsilon)^{h_k-1}\beta'_j + p_{e_j} - \alpha'_j$. The first inequality follows since $\alpha'_j \leq \alpha_j$ as we always maintain a feasible solution, and the second inequality follows from (II).

Next, for any lost item $j \in L_k$, we have by induction that

$$\begin{aligned}
\beta_j + p_{e_j} - \alpha_j &= (\beta'_j + p_{e_j} - \alpha'_j) + (\beta_j - \beta'_j + \alpha'_j - \alpha_j) \\
&\leq (\beta_j - \beta'_j + \alpha'_j - \alpha_j) \\
&\leq ((1+\epsilon)^{h_k - 1} - 1)\, \beta'_j + \alpha'_j - \alpha_j \\
&\leq ((1+\epsilon)^{h_k - 1} - 1)\, \alpha'_j + \alpha'_j - \alpha_j \\
&\leq (1+\epsilon)^{h_k - 1}\, \alpha'_j - \alpha_j.
\end{aligned}$$

This together with (4) now imply that (III) holds for $i = k$. Note next that by applying (3), (4), (III) and (I) we obtain:

$$\begin{aligned}
\alpha_k &= \textsc{opt}(b_k) + \sum_j \alpha_j \\
&\leq (1+\epsilon)^{h_k-1}\, \textsc{opt}(\overline{I}(b_k)) + \sum_{j \in L_k} ((1+\epsilon)^{h_k-1}\, \alpha'_j - \alpha_j) + \sum_j \alpha_j \\
&= (1+\epsilon)^{h_k-1}\, \textsc{opt}(\overline{I}(b_k)) + \sum_{j \in L_k} (1+\epsilon)^{h_k-1}\, \alpha'_j + \sum_{j \notin L_k} \alpha_j \\
&\leq (1+\epsilon)^{h_k-1}\, \textsc{opt}(\overline{I}(b_k)) + (1+\epsilon)^{h_k-1} \sum_j \alpha'_j \\
&\leq (1+\epsilon)^{h_k} \textsc{fptas}(\overline{I}(b_k)) + (1+\epsilon)^{h_k-1} \sum_j \alpha'_j \\
&\leq (1+\epsilon)^{h_k} (\textsc{fptas}(\overline{I}(b_k)) + \sum_j \alpha'_j)
\end{aligned}$$

Since $\alpha'_k = \textsc{fptas}(\overline{I}(b_k)) + \sum_j \alpha'_j$ we thus obtain (I). The argument for (II) is similar. Thus, $\mathcal{A}_n$ is a feasible demand matching with profit at least $\frac{1}{(1+\epsilon)^n}$ times that of the optimum demand matching for $\mathcal{I}$.

We can then generate an FPTAS as follows. Apply the dynamic programming procedure outlined above with $\epsilon = \frac{\log(1+\epsilon^*)}{n}$. We obtain a solution of value at least $\frac{1}{(1+\epsilon)^n}\textsc{opt} \geq \frac{1}{1+\epsilon^*}\textsc{opt}$. The total running time is bounded by a polynomial in n multiplied by the maximum time required to solve a knapsack problem at a node. These knapsack problems take time polynomial in the input size and $\frac{1}{\epsilon}$ since we invoke an FPTAS for each knapsack. For small ϵ^*, we have $\epsilon \geq \frac{\log(1+\epsilon^*)}{n} \geq \frac{\epsilon^*}{2n}$ and hence the overall running time of the algorithm is polynomial in n and $\frac{1}{\epsilon^*}$. □

Corollary 1. *There is an exact polytime algorithm for the unit-profits demand matching problem on a tree.*

Proof. Consider the node v_i with out-neighbours u_j, $j = 1, \ldots, r$. Note that in the unit-profits case, we may determine $\textsc{opt}(t)$ exactly. Simply consider the edges $e_1, \ldots, e_j$ by increasing magnitude of demand, and take the largest feasible subset of the form $\{e_j\}_{j=1}^r$. Then dynamic programming gives an optimal solution for the whole instance. □

Recently C. Chekuri mentioned that one may alter the demands of a tree instance ahead of time so that they lie in a small range (polynomially bounded in n and ϵ) and so that an optimal solution to the new instance is within a $1+\epsilon$ factor of the original. In this case, Theorem 10 could be proved along the lines of the above corollary by computing the α, β values exactly for the new instance. We leave the details to the reader.

Acknowledgments. This work was performed while the second author was visiting Bell Labs in the Summer 2001. The work grew, in a convoluted way, out of discussions in 1998 with Vincenzo Liberatore on maximum multicommodity flow in a path. We are thankful to Chandra Chekuri for very helpful comments including pointing out connections to [23]. The authors also thank Juli Atherton, Anupam Gupta, Santosh Vempala and Gerhard Woeginger for their constructive remarks. The authors are especially grateful to Joseph Cheriyan for generously sharing his time and insights on the contents of the paper. Finally, this paper benefitted greatly from three very thoughtful referee reports; the authors thank the anonymous referees for their efforts.

References

1. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor and B. Schieber, "A unified approach to approximating resource allocation and scheduling", *Journal of the ACM*, **48(5)**, pp1069-90, 2001.
2. P. Berman and M. Karpinski, "On some tighter inapproximability results", *ICALP*, pp200-209, 1999.
3. G. Calinescu, A. Chakrabarti, H. Karloff and Y. Rabani, "Improved approximation algorithms for resource allocation", *IPCO*, pp401-414, 2002.
4. A. Chakrabarti, C. Chekuri, A. Gupta and A. Kumar, "Approximation algorithms for the unsplittable flow problem", *APPROX*, pp51-66, 2002.
5. C. Chekuri and S. Khanna, "A PTAS for the multiple knapsack problem", *SODA*, pp213-222, 2000; to appear in *SIAM J. Computing*.
6. C. Chekuri, M. Mydlarz and F. B. Shepherd, "Multicommodity demand flow on a tree", *ICALP*, pp410-425, 2003.
7. J. Correa and M. Goemans,"An approximate König's theorem for edge coloring weighted bipartite graphs", *STOC*, pp398-406, 2004.
8. S. Cosares and I. Saniee, "An Optimization Problem Related to Balancing Loads on SONET Rings", *Telecommunications Systems* **3**, pp165-181, 1994.
9. Y. Dinitz, N. Garg and M. Goemans, "On the single-source unsplittable flow problem", *Combinatorica*, **19**, pp17-41, 1999.
10. D.Z. Du, B. Gao, F.K. Hwang, J.H. Kim, "On multirate rearrangeable Clos networks", *SIAM J. Computing*, **28**, pp463-470, 1998.
11. J. Edmonds, "Maximum matching and a polyhedron with $0,1$-vertices", *Journal of Research of the National Bureau of Standards (B)*, **69**, pp125-30, 1965.
12. M. Farach and V. Liberatore, "On local register allocation", *SODA*, pp564–573, 1998.

13. N. Garg, V. Vazirani and M. Yannakakis, "Primal-dual approximation algorithms for integral flow and multicut in trees with applications to matching and set cover", *Algorithmica* **18**, pp3-20, 1997.
14. A. Hoffman and J. Kruskal, "Integral boundary points of convex polyhedra", in H. Kuhn and A. Tucker eds., *Linear Inequalities and Related Systems*, Princeton University Press, pp223-246, 1956.
15. O. Ibarra and C. Kim, "Fast approximation for the knapsack and sum of subset problems", *Journal of the ACM*, **22**, pp463-468, 1975.
16. J. Kleinberg, "Approximation Algorithms for Disjoint Paths Problems", *Ph.D Thesis, Dept. of EECS, MIT*, 1996.
17. J. Kleinberg, "Single-source unsplittable flow", *FOCS*, pp68-77, 1996.
18. S.G. Kolliopoulos and C. Stein, "Approximation algorithms for single-source unsplittable flow", *SIAM J. Computing*, **31**, pp919-946, 2002.
19. S.G. Kolliopoulos and C. Stein, "Approximation disjoint-path problems using packing integer programs", *Math. Programming A*, **99**, pp63-87, 2004.
20. H.Q. Ngo and V.H. Vu, "On multirate rearrangeable Clos networks and a generalized edge coloring problem on bipartite graphs, *SIAM Journal on Computing*, **32(4)**, pp1040-1049, 2003.
21. C. Papadimitriou, *Computational Complexity*, Addison Wesley, 1994.
22. F.B. Shepherd and G. T. Wilfong, *Manuscript*, June 2003.
23. D. Shmoys and É. Tardos, "An approximation algorithm for the generalized assignment problem", *Mathematical Programming A*, **62**, pp461-474, 1993.
24. Martin Skutella, "Approximating the single source unsplittable min-cost flow problem", *Mathematical Programming B* 91(3), pp493-514, 2002.
25. G. Woeginger, *Private Communication*, Jan. 2005.