Approximating Transitive Reductions for Directed Networks

Piotr Berman* Bhaskar DasGupta† Marek Karpinski‡

Abstract

We consider minimum equivalent *digraph* (*directed network*) problem (also known as the *strong transitive reduction*), its maximum optimization variant, and some extensions of those two types of problems. We prove the existence of polynomial time approximation algorithms with ratios 1.5 for all the minimization problems and 2 for all the maximization problems. We establish also approximation hardness result for those problems to within a constant even if the length of cycles is bounded by 5.

1 Introduction

1.1 Definitions and Motivation

Finding a minimum equivalent digraph is a classical computational problem (cf. [14]). Recently, several extensions of this problem were motivated by applications e.g. in networks, visualisation processes, social sciences and systems molecular biology.

The statement of the equivalent digraph problem is simple. For a digraph $G = (V, E)$ the *transitive closure* of E is relation $u \overset{E}{\to} v$ ("E contains a path from u to v"). In turn, A is an equivalent digraph for E if (a) $A \subseteq E$, (b) transitive closures of A and E are the same.

To formulate an optimization problem, besides the definition of a valid solution we need an objective function. We consider two versions: MIN-ED, in which we minimize $|A|$, and MAX-ED, in which we maximize $|E - A|$. where A is an equivalent digraph for E.

*Department of Computer Science & Engineering, Pennsylvania State University, University Park, PA 16802. Research partially done while visiting Dept. of Computer Science, University of Bonn and supported by DFG grant Bo 56/174-1. Email: `berman@cse.psu.edu`

†Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053. Supported by NSF grants DBI-0543365, IIS-0612044 and IIS-0346973. Email: `dasgupta@cs.uic.edu`

‡Department of Computer Science, University of Bonn, 53117 Bonn. Supported in part by DFG grants, Procope grant 31022, and Hausdorff Center research grant EXC59-1. Email: `marek@cs.uni-bonn.de`

If we skip condition (a) we obtain *transitive reduction* problem which is optimally solved by Aho *et al.* [1]. These names are a bit confusing because one would expect a *reduction* to be a subset and an *equivalent set* to be unrestricted, but transitive reduction was first discussed when the name *minimum equivalent digraph* was already introduced [14]. This could motivate renaming the equivalent digraph as a *strong transitive reduction* [15].

In the study of biological networks two types of interactions are considered. For example, nodes can represent genes and an edge (u, v) means that gene u *regulates* gene v. Without going into biological details, *regulates* may two different things: when u is *expressed*, *i.e.* molecules of the protein coded by u are created, the expression of v can be *repressed* or *promoted.* A path in this network is an indirect interaction, and promoting a repressor represses, while repressing a repressor *promotes* (biologists also use the term *de-repression*). Interactions of such two types can appear in other contexts as well, including e.g. social networks. This motivates an extension of the notion of a digraph and its transitive closure.

Moreover, for certain interactions we have direct evidence, so an instance description includes set $D \subset E$ of edges which have to be present in every valid solution. Formally, we define A to be a valid solution to instance (V, E, ℓ, D) as follows:

① $\ell : E \to \mathbb{Z}_p$;

② a path $P = (u_0, u_1, \ldots, u_k)$ has characteristic $\ell(P) = \sum_{i=1}^{k} \ell(u_{i-1}, u_i)$;

③ $\mathsf{Closure}_\ell(E) = \{(u, v, q) : \exists P$ in E from u to v and $\ell(P) = q\}$;

④ A is a p-ary transitive reduction of E with a required subset D if $D \subseteq A \subseteq E$ and $\mathsf{Closure}_\ell(A) = \mathsf{Closure}_\ell(E)$.

Our two objective functions define optimization problems Min-TR$_p$ and Max-TR$_p$.

1.2 Earlier Results

The initial work on the minimum equivalent digraph by Moyles and Thomson [14] described an efficient reduction to the case of strongly connected graphs and an exact exponential time algorithm for the latter.

Several approximation algorithms for Min-ED were described, by Khuller *et al.* [11], with approximation ratio $1.617+\varepsilon$ and claimed by Vetta [16] with approximation ratio 1.5. The latter result seems to have some gaps in a correctness proof.

If edges have costs, we can minimize $c(A)$ within factor 2 using an algorithm for minimum cost rooted arborescence [6, 9] of Edmonds (who described it) and Karp (who simplified it). We can obtain an equivalent digraph by fixing an arbitrary root $r \in V$ and taking the union of minimum costs in- and out- arborescence.

Albert *et al.* [2] showed how to convert an algorithm for Min-ED with approximation ratio r to an algorithm for Min-TR$_1$ with approximation ratio $3-2/r$. They have also shown a ratio 2 approximation for Min-TR$_p$. Other heuristics for these problems were investigated in [3, 8].

On the hardness side, Papadimitriou [15] indicates that the strong transitive reduction is NP-hard, Khuller *et al.* have proven it formally and they also showed its MAX-SNP hardness. Motivated by their *cycle contraction* method in [11], they were interested in the complexity of the problem when there is an upper bound γ on the cycle length; in [10] they showed that MIN-ED is polynomial with $\gamma = 3$, NP-hard with $\gamma = 5$ and MAX-SNP-hard with $\gamma = 17$.

1.3 Results in this Paper

We show an approximation algorithm for MIN-ED with ratio 1.5, We use a method somewhat similar to that of Vetta [16], but our combinatorial lower bound makes a more explicit use of the primal-dual formulation of Edmonds and Karp, and this makes it much easier to justify edge selections within the promised approximation ratio.[1]

Next, we show how to modify that algorithm to approximate MIN-TR$_1$ within ratio 1.5. Notice that one cannot use a method for MIN-ED as a "black box" because we need to control which edges we keep and which we delete.

We design an approximation algorithm with ratio 2 for MAX-TR$_1$. While it was shown by Albert *et al.* [3] that a simple greedy algorithm, delete an unnecessary edge as long as one exists, yields ration 3 approximation, it is easy to provide an example of MAX-ED instance with n nodes and $2n-2$ edges in which greedy removes only one edge, and the optimum solution removes $n-2$ edges. Other known algorithms for MIN-ED are not much better in the worst case when applied to MAX-ED.

We show that for a prime p we can transform an equivalent digraph that contains the required edges into a p-ary transitive reduction by a single edge insertion per strongly connected component. Because every p-ary transitive reduction is also an equivalent digraph, this transformation implies an approximation algorithms for MIN-TR$_p$ with ratio 1.5 and for MAX-TR$_p$ with ratio 2 (we can compensate for an insertion of a single edge, so the ratio does not change).

We simplify an approximation hardness proof for MIN-ED (the proof applies to MAX-ED as well) and it is applicable even when γ, the maximum cycle length, is 5. This leaves unresolved only the case of $\gamma = 4$.

1.4 Some Motivations and Applications

Application of MIN-ED: Connectivity Requirements in Computer Networks. Khuller *et al.* [10] indicated applications of MIN-ED to design of computer networks that satisfy given connectivity requirements.

With preexisting sets of connections, this application motivates MIN-TR$_1$ (cf. [12]).

Application of MIN-TR$_1$: Social Network Analysis and Visualization. MIN-TR$_1$ can be applied to social network analysis and visualization. For example, Dubois

[1] It appears that the approach of [16] contains some gaps. We will comment on the approach of [16] again in Section 3.1.

and Cécile [5] applies Min-TR_1 to the publically available (and famous) social network built upon interaction data from email boxes of Enron corporation to study useful properties (such as scale-freeness) of such networks as well as help in the visualization process. The approach employed in [5] is the straightforward greedy approach which, as we have discussed, ha inferior performance, both for Min-TR_1 and Max-TR_1.

Application of Min-TR_2: Inferring Biological Signal Transduction Networks. Most biological characteristics of a cell arise from complex interactions between its numerous constituents such as DNA, RNA, proteins and small molecules. Cells use signaling pathways and regulatory mechanisms to coordinate multiple functions, allowing them to respond to and adapt to an ever-changing environment. Genome-wide experimental methods now identify interactions among thousands of cellular components. Experimental information about the interactions in a given signal transduction network can be *direct*, *e.g.* by observing a particular molecular interaction in a special experiment, or *indirect*, *e.g.* when we influence a component of the network and observe the impact on other components. To synthesize all this information into a consistent network, we need to determine how the different interactions suggested by experiments fit together. The Min-TR_2 problem allows to determine the sparsest graph consistent with experimental observations and it is the key part of the network synthesis procedure described in Figure 1 of Albert *et al.* [3].

1.5 Our Techniques

Approximation Algorithm for the Min Objective. We consider a natural primal/dual IP formulation for Min-TR_1 that is a straightforward extension of a primal/dual formulation of minimum cost rooted arborescence problem on directed graphs (e.g., see [6, 9] or [13, Corollary 6.12]). This formulation introduces an inequality for every node set, but they exist for the analysis only. We keep only some of the constraints and this gives a system whith a feasible integral solution L of the dual which is not necessarily a feasible integral solution of the primal but nonetheless provides a lower bound on the optimal cost and can be computed in polynomial time. This L is subsequently converted into a correct (primal) solution using a careful DFS and amortized accounting—we are allowed to use 3 edges for every 2 edges of L—to satisfy the remaining constraints. This yields an 1.5-approximation for Min-TR_1 that improved the 1.78-approximation in [2] and an alternate 1.5-approximation proof for Min-ED.

We also show an inherent limitation of our approach by showing an integrality gap of the LP relaxation of the above IP to be of at least $\frac{4}{3}$.

Approximation Algorithm for the Max Objective. For the Max-TR_1, we utilize the integrality of the polytope of the rooted arborescence problem to provide a 2-approximation. We also observe that the integrality gap of the LP relaxation of the IP formulation is at least 3/2.

The p-ary Case. We show that we can solve an instance of Min-TR_p/Max-

TR_p by solving a related instance of MIN-TR_1/MAX-TR_1 and inserting a appropriately choses single edge. In conjuction with our above results, this leads to a 1.5-approximation for MIN-TR_p and 2-approximation for MAX-TR_p. This method works only if p is prime.

Inapproximability. We show that both MIN-ED and MAX-ED are MAX-SNP-hard even if all cycles are of length at most 5, rather then 17 that was assumed in [11], Our improvement is obtained by reducing a very restricted version of the MAX-SAT (cf. [4]).

1.6 Notation

We use the following additional notations.

- $G = (V, E)$ is the input digraph;
- $\iota(U) = \{(u, v) \in E : u \notin U \;\&\; v \in U\}$;
- $o(U) = \{(u, v) \in E : u \in U \;\&\; v \notin U\}$;
- $scc_A(u)$ is the strongly connected component containing vertex u in the digraph (V, A);
- $T[u]$ is a the node set of the subtree with root u (of a rooted tree T).

2 A Primal-Dual Linear Programming Relaxation of TR_1

Moyles and Thompson [14] showed that MIN-ED can be reduced in linear time to the case when the input graph (V, E) is strongly connected, therefore we will assume that (V, E) is already strongly connected. In Section 5 we will will use a similar result obtained for MIN-TR_p in [2].

The minimum cost rooted arborescence problem on G is defined as follows. We are given a weighted digraph $G = (V, E)$ with a cost function $c : E \to \mathbb{R}_+$ and root node $r \in V$. A valid solution is $A \subseteq E$ such that in (V, A) there is a path from r to every other node and we need to minimize $c(A)$. A (possibly exponential-size) LP formulation for this was provided by Edmonds and Karp and goes as follows. As in any other edge/arc selection problems, we use the linear space with a coordinate for each arc, so edge sets can be identified with 0-1 vectors, and for an arc e variable x_e describes whether we select e ($x_e = 1$) or not ($x_e = 0$). Then, the LP formulation is:

(primal P1)
mininize $c \cdot x$ subject to
$x \geq 0$
$\iota(U) \cdot x \geq 1$ for all U s.t. $\varnothing \subset U \subset V$ and $r \notin U$ (2.1)

Edmonds [6] and Karp [9] showed that the above LP always has an integral optimal solution and that we can find it in polynomial-time.

We can modify the above LP formulation to a LP formulation for MIN-ED provided we set $c(e) \equiv 1$ and in (2.1) we remove "and $r \notin U$" from the condition. The dual program of this LP can be constructed by having a vector y that has a coordinate y_U for every $\emptyset \subset U \subset V$; both the primal and the dual is written down below for clarity:

(primal P2)	(dual D2)
mininize $\mathbf{1} \cdot x$ subject to	**maximize** $\mathbf{1} \cdot y$ subject to
$x \geq 0$	$y \geq 0$
$\iota(U) \cdot x \geq 1$ for all U s.t. $\emptyset \subset U \subset V$	$\sum_{e \in \iota(U)} y_U \iota(U) \leq 1$ for every $e \in E$

We can change P2 into the LP formulation for MAX-ED by replacing the objective to "maximize $\mathbf{1} \cdot (\mathbf{1} - x)$". and the dual is changed accordingly to reflect this change.

From now on, by a *requirement* we mean a set of edges R such that a valid solution must intersect it; in LP formulation it means that we have a constraint $Rx \geq 1$.

We can extend P2 to an LP formulation for TR_1 by adding one-edge requirements $\{e\}$ (and thus inequality $x_e \geq 1$) for each $e \in D$ where D is the set of edges that have to be present in a valid solution.

We can obtain a lower bound for solutions of P2 by solving P3, an IP obtained from P2 by allowing only those requirements $Rx \geq 1$ that for some node u satisfy $R \subseteq \iota(u)$ or $R \subseteq o(u)$. To find requirements of P3 efficiently, we first find strongly connected components of $V - \{u\}$. Then,

(a) for every source component C have requirement $\iota(C) \subset o(u)$;
(b) for every sink component C have requirement $o(C) \subset \iota(u)$;
(c) if we have requirements $R \subset R'$ we remove R'.

It is easy to see that after (c) one-edge requirements are disjoint with other requirements, hence multi-edge requirements form a bipartite graph (in which connections have the form of shared edges).

3 Minimization Algorithms

3.1 1.5-Approximation for MIN-ED

Using DFS

One can find an equivalent digraph using *depth first search* starting at any root node r. Because we operate in a strongly connected graph, only one root call of the depth first search is required. This algorithm mimics Tarjan's algorithm for finding strongly connected components and biconnected components. As usual for depth first search, the algorithm forms a spanning tree T in which we have an edge (u, v) if and only if DFS(u) made a call DFS(v). The invariant is

(A) if DFS(u) made a call DFS(v) and DFS(v) terminated then $T[v] \subset \mathrm{scc}_{T \cup B}(u)$.

```
Dfs(u)
{   Counter ←Counter+1
    Number[u] ← LowDone[u] ← LowCanDo[u] ←Counter
    for each edge (u, v)     // scan the adjacency list of u
        if Number[v] = 0
            Insert(T, (u, v))      // (u, v) is a tree edge
            Dfs(v)
            if LowDone[u] > LowDone[v]
                LowDone[u] ← LowDone[v]
            if LowCanDo[u] > LowCanDo[v]
                LowCanDo[u] ← LowCanDo[v]
                LowEdge[u] ← LowEdge[v]
        else if LowCanDo[u] > Number[v]
            LowCanDo[u] ← Number[v]
            LowEdge[u] ← (u, v)
    // the final check: do we need another back edge?
    if LowDone[u] = Number[u] and u ≠ r
        Insert(B,LowEdge[u])    // LowEdge[u] is a back edge
        LowDone[u] ← LowCanDo[u]
}

T ← B ← ∅
for every node u
    Number[u] ← 0
Counter ← 0
Dfs(r)
```

Figure 1: Dfs for finding an equivalent digraph of a strongly connected graph

(A) implies that $(V, T \cup B)$ is strongly connected when Dfs(r) terminates. Moreover, in any depth first search the arguments of calls that already have started and have not terminated yet form a simple path starting at the root. By (A), every node already visited is, in $(V, T \cup B)$, strongly connected to an ancestor who has not terminated. Thus, (A) implies that the strongly connected components of $(V, T \cup B)$ form a simple path. This justifies our convention of using the term *back edge* for all non-tree edges.

To prove the invariant, we first observe that when Dfs(u) terminates then LowCanDo$[u]$ is the lowest number of an end of an edge that starts in $T[u]$.

Application of (A) to each child of v shows that $T[v] \subset \mathrm{scc}_{T\cup B}(v)$ when we perform the final check of Dfs(v).

If the condition of the final check is false, we already have a B edge from $T[v]$ to an ancestor of u, and thus we have a path from v to u in $T \cup B$. Otherwise, we attempt to insert such an edge. If LowCanDo$[v]$ is "not good enough" then there is no path from $T[v]$ to u, a contradiction with the assumption that the graph is strongly connected.

The actual algorithm is based on the above Dfs, *but we also need to alter the set of selected edges in some cases.* We explain that below.

Objects, Credits, Debits

The initial solution L to the system P3 is divided into *objects*, strongly connected components of (V, L). L-edges are either inside objects, or between objects. We allocate L-edges to objects, and give 1.5 for each. In turn, an object has to pay for solution edges that connect it, for a T-edge that enters this object and for a B-edge that connects it to an ancestor. Each solution edge costs 1. Some object have enough money to pay for all L-edges inside, so they become strongly connected, and two more edges of the solution, to enter and to exit. We call them *rich*. Other objects are *poor* and we have to handle them somehow.

Allocation of L-Edges to Objects

- L-edge inside object A: allocate to A;
- from object A: call the first L-edge primary, and the rest secondary;
  - primary L-edge $A \to B$, $|A| = 1$: 1.5 to A;
  - primary L-edge $A \to B$, $|A| > 1$: 1 to A, and 0.5 to B;
  - secondary L-edge $A \to B$ (while there is a primary L-edge $A \to C$): if $|A| > 1$, 1.5 to B, otherwise 0.5 to each of A, B and C.

When is an Object A rich?

1. A is the root object, no payment for incoming and returning edges;
2. $|A| \geq 4$: it needs at most L-edges inside, plus two edges, and it has at least $0.5|A|$ for these two edges;
3. if $|A| > 1$ and an L-edge exits A: it needs at most L-edges inside, plus two edges, and it has at least $(1 + 0.5|A|)$ for these two edges;
4. if $|A| = 1, 3$ and a secondary L-edge enters A;
5. if $|A| = 1, 3$ and a primary L-edge enters A from some D where $|D| > 1$.

To discuss a poor object A, we call it a *path node, digons* or a *triangles* when $|A| = 1, 2$, or 3 respectively.

Guiding Dfs

For a rich object A, we decide at once to use L-edges inside A in our solution, and we consider it in Dfs as a single node, with combined adjacency list. This makes point **(1)** below moot. Otherwise, the preferences are in the order: **(1)** L-edges inside the same object; **(2)** primary L-edges; **(3)** other edges.

Analyzing the Balance of Poor Objects

A poor object A has parent object C; Dfs enters A from C to node $u \in A$.

We say that A shares (the cost of a B-edge) if either a B-edge to an ancestor of C is introduced by Dfs within a proper descendant D of A (A and D share the cost) or Dfs from an element of A introduces a B-edge to a proper ancestor of C (A and C share the cost). Path nodes and triangles that share have needs reduced to 1.5 and 4.5 respectively, hence they achieve balance.

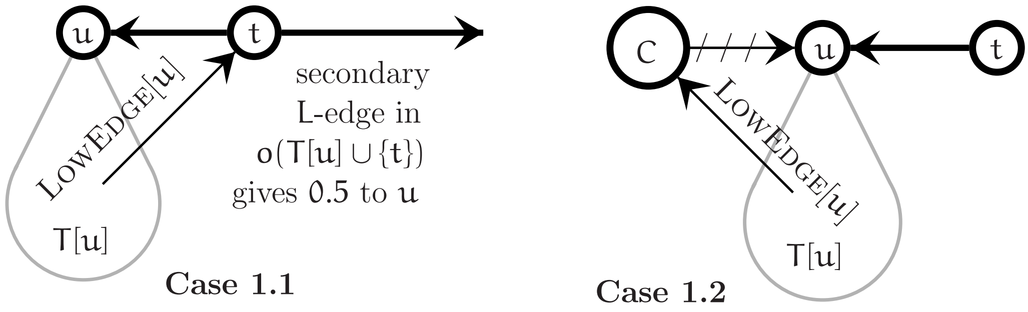

Figure 2: Illustrations for the cases of path nodes

Case 1: $|A| = 1$, $A = \{u\}$, A does not share.

Because we have requirements contained in $\iota(u)$ and in $o(u)$, there exists L-edges that enter and exit u. If an L-edge entering u is secondary or exits a multi-node object, A is rich. Hence we assume a primary edge (t, u) from object $\{t\}$.

Case 1.1: $C = \{t\}$. Because A does not share, no edge to an ancestor of t is present in B when the final check of $\text{DFS}(u)$ is performed, and thus $T[u]$ is already strongly connected. $\text{DFS}(u)$ inserts $\text{LowEdge}[u]$. Would this edge goes to a proper ancestor of t, A would share (with $\{t\}$). Thus $\text{LowEdge}[u]$ goes to $\{t\}$ (see Fig. 2). Then $o(T[u]) \subset \iota(u)$, hence $o(T[u] \cup \{t\}) \subset o(u)$, hence there must be an L-edge from t to a node different than u, a secondary edge from t, which makes A rich.

Case 1.2: $C \neq \{t\}$. Initially we pay for $C \to u$ and $\text{LowEdge}[u]$.

This means that t will be visited later. Because A does not share, it neither helps connecting $T[u]$ (which is strongly connected), nor it helps connecting C with its ancestor.

Thus it is OK when we delete T-edge $C \to u$ and we wait until t is visited in the future. Then $\text{DFS}(t)$ introduces edge (t, u), paid by A using the money for the deleted edge, and the cost of $\text{LowEdge}[u]$ is shared by A and $\{t\}$.

Note that our actual algorithm differs from DFS in two ways: ① $\text{DFS}(t)$ inserts to B the primary edge exiting t without waiting for the results of its recursive calls, and ② we insert this L-edge and delete a T edge. We will describe similar deviations in the subsequent cases.

Case 2: $|A| = 2$, $A = \{u, v\}$. $\text{DFS}(u)$ starts with making the call $\text{DFS}(v)$.

We consider what happens when we execute the final check of $\text{DFS}(v)$. If an edge to an ancestor of C is already introduced, A has to pay for T-edge $C \to u$, for edge (u, v) and it can "afford" to pay 1 for that B-edge (more than 0.5 for sharing the cost).

If an edge to u is already introduced, A does not share its cost, while $T[v] \cup \{u\}$ is already strongly connected. At the end of $\text{DFS}(u)$, A can afford to pay for introducing a B-edge.

Now we assume that no edge to a proper ancestor of v was introduced before the final check of $\text{DFS}(v)$, and thus $T[v]$ is already strongly connected. Thus DFSv inserts $\text{LowEdge}[v]$ to B. If this edge goes to an ancestor of C, again, A pays for three edges only.

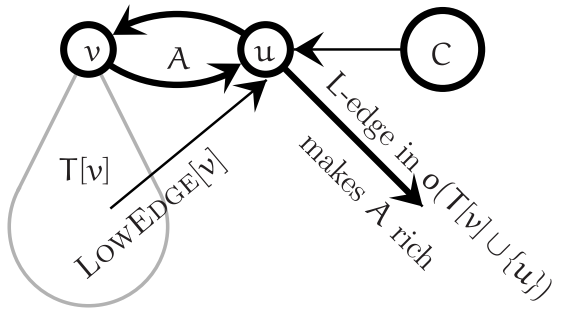

Figure 3: Illustrations for the cases of digons (Case 2)

If LowEdge$[v]$ goes to u, then $o(T[v]) \subset \iota(u)$, hence $o(T[v] \cup \{u\}) \subset o(u)$. This means that the has to exists an L edge that exits u and does not go to v, and this means that A is rich.

Case 3: $|A| = 3$, $A = \{u, v, w\}$. Triangles are subject of the following pre-processing. If $|V| > 3$ and a triangle must contain two solution edges, we contract it to a single node (this decreases the optimum cost by at least 2), find a solution and then we replace it with the cycle of this triangle (increasing the solution cost by 3). Clearly, this preserves approximation ratio 1.5. After the pre-processing a cycle of 3 nodes contains at most one necessary edge (or V is a triangle).

We assume that (u, v, w) is an oriented cycle, so Dfs(u) starts with a call to Dfs(v), which starts with a call to Dfs(w).

Case 3.1: A contains an endpoint of a primary L-edge. We can repeat the reasoning of Case 1, assume that this is edge from a path node t etc.

Case 3.2: Assume that the solution remains strongly connected when we remove A. We say that A is *free*. A has a surplus of 0.5 if we can traverse it as follows: from $V - A$, two edges inside A, back to $V - A$, so the balance is $4.5 - 4 = 0.5$. Below we show that such traversal is possible unless A is rich or Case 3.1 applies.

If all edges $(V - A) \to A$ enter through the same node, then an L-edge enters A, and either A is rich or we have Case 3.1. Similarly, if all edges $A \to (V - A)$ exit from the same node, A is rich.

If we can exit from node w, use the path $(V - A) \to u \to v \to w \to (V - A)$. Otherwise we can assume exits from both u and v.

If we can enter A at node v, use the path $(V - A) \to v \to w \to u \to (V - A)$ and if we can enter A at node w, use $(V - A) \to w \to u \to v \to (V - A)$.

Case 3.3: In the remaining case, no L-edge enters or exits A, edges enter A at two nodes, and exit at two nodes, A does not share and A is not free.

One conclusion is that Dfs(u) must visit other objects besides A. Therefore $T[u]$ has *branches* that extend beyond A. We will consider the structure of those branches. A branch is completed when a B-edge to a proper ancestor of its first object is inserted; this merges some scc's, say $D_1, ..., D_k$ into the ancestral scc, so it uses $k + 1$ edges outside D_i's. If such a branch has a surplus we transfer 0.5 to A which completes its accounting. Our goal is to show that such a surplus exists, or we can make A share, or we can make A free.

Case 3.3.1: $k > 2$. Each D_i has a "local surplus" of at least 0.5 after paying for the incoming edge, and this includes the case of nodes of a digon; a digon has 3 and two nodes. Thus we can pay for the last edge and the branch still has positive surplus.

Case 3.3.2: $k = 2$ and either D_1 or D_2 is rich. The accounting is the same as in Case 3.3.1.

Case 3.3.3: D_1 is a path node, say t. Then there exists an L-edge (s, t) and because A is not rich, $s \notin A$. Delete $A \to t$ and backtrack from t using L-edges until you leave $T[u]$ or you encounter a cycle object, say F.

If you leave $T[u]$ while backtracking, replace $C \to u$ with the edge that "left $T[u]$ backwards". Because we removed edge $A \to t$, this improves the balance by 1.

If you reach a cycle object inside $T[u]$ via a primary edge, we obtain a branch that goes through rich F and t, so we have Case 3.3.1 or 3.3.2. If we reach F via a secondary edge $F \to s$, s is "super rich" with 3 and it can transfer 0.5 to A.

Case 3.3.4: D_2 is a path node, say t, while D_1 is not. Because D_1 is not rich and not a digon, it is a triangle. We repeat the reasoning of the previous case, while D_1 becomes a free triangle.

Case 3.3.5: Remaining case: each branch either has a single scc D_1, which is rich (otherwise it is a free triangle), or two triangle scc's, or two single-node scc's that together form a digon.

Case 3.3.5.1: Open case: there exists an edge between $V - T[u]$ and $T[u] - A$. We will analyze the case of an edge $V - T[u] \to T[u] - A$, the other case is symmetric.

It this edge enters some D_1, then we insert it and delete two edges, $C \to A$ and $A \to D_1$.

If this edge enters some D_2 where D_1, D_2 is a pair of triangles, we insert it, remove $C \to A$, $A \to D_1$ and $D_1 \to D_2$ and thus we make D_1 a free triangle, so restoring the connections of $A \to D_1 \to D_2$ will save an edge.

If this edge enters some $D_2 = \{t_2\}$ where $D_1 = \{t_1\}$ and $\{t_1, t_2\}$ is a digon, we insert this edge, remove $A \to t_1$ and $t_2 \to A$ and insert an edge (t_1, s) for some $s \neq t_2$. It is not possible, exits from t_1 are restricted to t_2, hence $o(\{t_1, t_2\}) \subset o(t_2)$, so an L-edge must exit the digon and the digon is rich and we can treat it like "entering D_1".

If $s \in T[u]$, we can delete $C \to A$ and save. Otherwise the progress is in making the set $T[u] - A$ smaller.

Case 3.3.5.2: Closed case. The edges between $V - T[u]$ and $T[u]$ must include nodes in A. We can free A: there must be at least two nodes in A that are entered by such edges, otherwise the edge $C \to A$ is an L-edge, Case 3.1. Similarly, there must be two nodes in A from which such edges can exit. So we can repeat the reasoning of Case 3.2.

Remark 1. When we discuss sub-cases of Case 3.3, we assumed that the scc's that are coalesced when a branch is completed are of one of the "basic types". Actually, they can have a nested structure, following the recursive nature of DFS. If this is the case, we can identify an scc D_i with its root object. If the root is rich, then D_i inherits the initial balance if the root, so it is rich. If the root is a triangle, D_i inherits

its balance as well, additionally, making D_i free has the same effect as having free $T[u]$, a subtree rooted by A that is discussed in those cases. Because we want to gain by making a subtree rooted by a triangle free, we reduce the problem to that of a smaller subtree with the same property.

The cases of path nodes, 3.3.3 and 3.3.4 are not altered if these path nodes are roots of larger scc's.

Finally, the open case with digon (3.3.5.1) has to be elaborated when we have an edge from "outside" to the subtree rooted at $D_2 = \{t_2\}$. We can follow the reasoning of Case 2: after reaching t_t, we proceed instantly to t_1 and re-launch $\text{DFS}(t_1)$. If it will produce a B-edge to $T[t_1]$, then the digon $\{t_1, t_2\}$ gets a "free connection" and we can balance it in the same way as a rich object 4 nodes.

If it will produce an edge somewhere else, the situation is equivalent to the one we discussed, the digon together with attached subtrees either provides a new entry to A or it can be detached from $T[u]$, and we have progress in either case.

Remark 2. Vetta [16] uses a similar combinatorial lower bound but without a formulation that we have got an L-edge inside every requirement contained in some $o(u)$ or $\iota(u)$. That results in a large number of cases to consider for the path nodes and digons, making it hard to verify how resolution of some cases impacts consistency of other cases. The second consequence is that in many situations, [16] indicates how to make an edge replacement rather than to show existence of a lower bound edge. However in Min-TR and Max-TR problems we have a class of edges that we are not allowed to replace, so we have to use edge replacement only as "the last resort".

3.2 Extending the Algorithm for Min-ED to Min-TR$_1$

When the set of required edges is not empty, $D \neq \emptyset$, the approach in the previous section has to be somewhat modified. First, we can do the following initial preprocessing. If there is a cycle of D-edges we simplify the problem by replacing the cycle with a single node. We do the same if there is a triangle that contains two D-edges, as it was described for the necessary edges. When we construct combinatorial lower bound L, we automatically get $D \subset L$. The difficulty is that in some cases the previous algorithm can replace L edges.

It never happens with L-edges in paths and rich objects, so it suffices to consider poor digons and triangles, and make necessary modification to our algorithm.

Digons with Required Edges

Because our algorithm can remove edges from digons, we want to avoid the possibility of digons that contain required (or necessary) edges. Therefore we alter the method of obtaining the approximate (lower bound) solution L.

Suppose that we have a digon with required edge (u, v) and the other edge (v, u). We collapse it to a single node t_{uv}. Now we can have P3 constraints that are contained in $\iota(t_{uv})$ (and the symmetric case of $o(t_{u,v})$). If there is only one such constraint, we increase the cost of satisfying it by 1.5 if the edge that is used enters node v.

The idea is that a solution in which edge (v, u) is not used corresponds to a solution of the reduced problem in which constraint $\iota(t_{uv})x \geq 1$ is satisfied with an edge to node u, hence without increased cost, while the use of an "expensive" edge corresponds to sharing the cost of (v, u) with constraint $o(t_{uv})$.

If we have multiple constraints contained in $\iota(t_{uv})$ we could extract 0.5 with a more careful analysis, so we can pay for (v, u). Otherwise, if we obtain a solution with an expensive way of satisfying $\iota(t_{uv})x \geq 1$ we transfer this 0.5 from this edge to the digon to pay for (v, u).

The same happens for $o(t_{uv})$. Thus either we proceed with both edges of the digon selected and paid for, or with a path $(wt_{uv}w')$ that can be replaced with path $(wuvw')$. We never proceed with a digon object that has a required edge.

Triangles

Our method can be applied to triangles with small modifications. It is still the case that when a triangle is free we can connect its nodes with the rest of the solution using 4 edges, but the argument has to be a bit different in the presence of required edges.

Thus suppose that we obtained a solution for the complement of a triangle $A = (u, v, w)$ in which edge (w, u) is required. If we cannot enter A through node v, v has to be entered from inside A, hence every solution must have two edges inside A, hence we would collapse A in the preprocessing. The case when there is no exit of A from node u is symmetric. Thus we can enter A through v, traverse (v, w, u) and exit through u. We say that A is *free*, and a free triangle has a surplus of 0.5.

We can summarize this section with the following theorem:

Theorem 1 *There is a polynomial time algorithm that given an input graph* (V, E) *and a set of required edges* $D \subset E$ *produces a transitive reduction* H *such that* $D \subseteq H \subseteq E$ *and* $|H| \leq 1.5k - 1$*, where* k *is the size of an optimum solution.*

The reason for -1 in the statement is that no edges are added for the root object in L, and this object has at least 2 edges.

4 2-Approximation for Max-TR$_1$

We formulate now a result on approximability of Max-TR$_1$ problem.

Theorem 2 *There is a polynomial time algorithm that given an input graph* (V, E) *and a set of required edges* $D \subset E$ *produces a transitive reduction* H *such that* $D \subseteq H \subseteq E$ *and* $|E - H| \geq 0.5k + 1$*, where* k *is the size of* $|E - H|$ *for an optimum solution.*

(In the proof, we add in parenthesis the parts needed to prove $0.5k + 1$ bound rather than $0.5k$.) First, we determine the *necessary* edges: e is necessary if $e \in D$ or $\{e\} = \iota(S)$ for some node set S. (If there are any cycles of necessary edges, we replace them with single nodes.)

We give a cost of 0 to the necessary edges and a cost of 1 for the remaining ones. Remember the primal/dual formulations (in particular (P2) and (D2)) of Section 2. We set $x_e = 1$ if e is a necessary edge and $x_e = 0.5$ otherwise. This is a valid solution for the fractional relaxation of the problem as defined in (P1).

Now, pick any node r. (Make sure that no necessary edges enter e.) The out-arborescence problem, as defined in Section 2, is to find a set of edges of minimum cost that provides a path from r to every other node; edges of cost 0 can be used in every solution. An optimum (integral) out-arborescence T can be computed in polynomial time by the greedy heuristic in [9], this algorithm also provides a set of cuts that forms a dual solution.

Suppose that m edges of cost 1 are *not* included in T, then no solution can delete more than m edges (indeed, more than $m-1$, to the cuts collected by the greedy algorithm we can add $\iota(r)$). Let us reduce the cost of edges in T to 0. Our fractional solution is still valid for the in-arborescence, so we can find the in-arborescence with at most $m/2$ edges that still have cost 1. Thus we delete at least $m/2$ edges, while the upper bound is m $(m-1)$.

(To assure deletion of at least $\ell/2+1$ edges, where ℓ is the optimum number, we can try in every possible way one initial deletion. If there optimum number of deletions is k, we are left with approximating among $k-1$, we get an upper bound of at least $k-1$ with k edges left for possible deletions, so we delete at least $k/2$, plus the initial 1.)

5 Approximating Min-TR$_p$ and Max-TR$_p$

We will show how to transform our approximation algorithms for Min-TR$_1$ and Min-TR$_1$ into approximation algorithms for Min-TR$_p$ and Max-TR$_p$ with ratios 1.5 and 2 respectively. In a nutshell, we can reduce the approximation in the general case the case of a strongly connected graph, and in a strongly connected graph we will show that a solution to Min-TR$_1$ (Max-TR$_1$) can be transformed into a solution to Min-TR$_p$ (Max-TR$_p$) by adding a single edge, and in polynomial time we can find that edge.

In turn, when we run approximation algorithms within strongly connected components, we obtain the respective ratio even if we add one extra edge.

Let G be the input graph. The following proposition says that it suffices to restrict our attention to strongly connected components of G[2]. (One should note that the algorithm implicit in this Preposition runs in time proportional to p.)

Proposition 3 [2] *Suppose that we can compute a ρ-approximation of TR$_p$ on each strongly connected component of G for some $\rho > 1$. Then, we can also compute a ρ-approximation of TR$_p$ on G.*

The following characterization of scc's of G appear in [2].

Lemma 4 [2] *Every strongly connected component $U \subset V$ is one of the following two types:*

(Multiple Parity Component) $\{q : (u,v,q) \in \mathsf{Closure}_\ell(E(U))\} = \mathbb{Z}_p$ *for any two vertices* $u, v \in U$*;*

(Single Parity Component) $|\{q : (u,v,q) \in \mathsf{Closure}_\ell(E(U))\}| = 1$ *for any two vertices* $u, v \in U$.

[2]The authors in [2] prove their result only for Min-TR$_p$, but the proofs work for Max-TR$_p$ as well.

Based on the above lemma, we can use the following approach. Consider an instance (V, E, ℓ, D) of MIN-TR$_p$. For every strongly connected component $U \subset V$ we consider an induced instance of MIN-TR$_1$, $(U, E(U), D \cap U)$. We find an approximate solution A_U that contains an out-arborescence T_U with root r. We label each node $u \in U$ with $\ell(u) = \ell(P_u)$ where P_u is the unique path in T_U from r to u.

Now for every $(u, v) \in E(U)$ we check if $\ell(v) = \ell(u) + \ell(u, v) \ \mathbf{mod}\ p$.

If this is true for every $e \in E(U)$ then U is a single parity component. Otherwise, we pick a single edge (u, v) violating the test and we insert it to A_U, thus assuring that (U, A_U) becomes a multiple parity component.

6 Integrality Gap of the LP Formulation for TR$_1$

We formulate now

Lemma 5 *The LP formulation P2 for* MIN-ED *and* MAX-ED *has an integrality gap of at least* $4/3$ *and* $3/2$*, respectively.*

We use the same construction for MIN-ED and MAX-ED. Our graph will consist of $2n$ nodes. We first define $2n + 2$ nodes of the form (i, j) where $0 \leq i < 2$ and $0 \leq i \leq n$. Later we will collapse together nodes $(0, 0)$ and $(1, 0)$ into node 0, as well as nodes $(0, n)$ and $(1, n)$ into node n. We have two types of edges: $((i, j), (i, j+1))$ and $((i, j), (i \pm 1, j - 1))$.

We get a fractional solution by giving coefficient 0.5 to every edge. We need to show that $U \neq V$ and $U \neq \varnothing$ implies $|\iota(U)| \geq 2$. Suppose $\{0, (0, 1), \ldots, (0, n-1), n\} \subset U$; let j be the least number such that $(1, j) \notin U$; then $\iota(U)$ contains $((1, j-1), (1, j))$ and $((0, j+1), (1, j))$. A symmetric argument holds if $\{0, (0, 1), \ldots, (0, n-1), n\} \subset U$. In the remaining case, edge disjoint paths $(0, (i, 1), \ldots, (i, n-1), n)$, $i = 0, 1$, contain edges from $\iota(U)$.

The cost of this fractional solution is $2n$ (two edges from every of $2n$ nodes, times 0.5). We will show that the minimum integer solution costs $\lceil (8n-4)/3 \rceil$. For this, it suffices to show that no simple cycle in this graph has the length exceeding 4 nodes.

If no two consecutive edges in a cycle increase the value of the second coordinate, no pair of edges increases the this value, so we have at most two such values, hence at most 4 nodes. Alternatively, if a cycle has two such edges, say the path $((0, i-1), (0, i), (0, i+1))$, it has to return without using edges that are incident to $(0, i)$ so it has to use $((0, i+1), (1, i))$ and $((1, i), (0, i-1))$, so it is a cycle of length 4, $((0, i-1), (0, i), (0, i+1), (1, i))$.

Every edge of a minimum solution belongs to a simple cycle contained in that solution; we can start with set $\{0\}$ and extend it using an edge going from the current set, and a simple cycle that contains that edge; if we add k edges to the solution we add $k - 1$ nodes, and $k \leq 4$; thus the average cost of adding a node must be at least $4/3$ edges. This completes the proof for MIN-ED.

When we analyze this example for MAX-ED, fractional relaxation allows $2n$ deletions while actually we can perform only $\frac{4}{3}n$ of them, so the ratio is $2\frac{3}{4} = \frac{3}{2}$.

7 Approximation Hardness Results

We turn now to the approximation hardness of the above problems. The proof method uses similar ideas to [11] and [7].

Theorem 6 *Let* k-problem *be the* problem *restricted to graphs in which the longest cycle has* k *edges.* 5-Min-ED *and* 5-Max-ED *are both MAX-SNP-hard.*

Proof. We will use a single approximation reduction that reduces 2Reg-Max-SAT to 5-Min-ED and 5-Max-ED.

In Max-SAT problem the input is a set S of disjunctions of literals, a valid solution is an assignment of truth values (a mapping from variables to $\{0, 1\}$), and the objective function is the number of clauses in S that are satisfied. 2Reg-Max-SAT is Max-SAT restricted to sets of clauses in which every variable x occurs exactly four times (of course, if it occurs at all), twice as literal x, twice as literal $\overline{x}$. This problem is MAX-SNP hard even if we impose another constraint, namely that each clause has exactly three literals [4].

Consider an instance S of 2Reg-Max-SAT with n variables and m clauses. We construct a graph with $1 + 6n + m$ nodes and $14n + m$ edges. One node is h, *the hub*. For each clause c we have node c. For each variable x we have a gadget G_x with 6 nodes, two switch nodes labeled x, two nodes that are occurrences of literal x and two nodes that are occurrences of literal $\overline{x}$.

We have the following edges: (h, x^*) for every switch node, (c, h) for every clause node, (l, c) for every occurrence l of a literal in clause c, while each node gadget is connected with 8 edges as shown in Fig. 4.

We will show that

① if we can satisfy k clauses, then we have a solution of Min-ED with $8n+2m-k$ nodes, which is also a solution of Max-ED that deletes $6n - m + k$ edges;

② if we have a solution of Min-ED with $8n+2m-k$ edges, we can show a solution of 2Reg-Max-SAT that satisfies k clauses.

To show ①, we take a truth assignment and form an edge set as follows: include all edges from h to switch nodes ($2n$ edges) and from clauses to h (m edges). For a variable x assigned as true pick set A_x of 6 edges forming two paths of the form $(x^*, \overline{x}, x, c)$, where c is the clause where literal x occurs, and if x is assigned false, we pick set $A_{\overline{x}}$ of edges from the paths of the form $(x^*, x, \overline{x}, c)$ ($6n$ edges). At this point, the only nodes that are not on cycles including h are nodes of unsatisfied clauses, so for each unsatisfied clause c we pick one of its literal occurrences, l and add edge (l, c) ($m - k$ edges).

To show ②, we take a solution D of Min-ED. D must contains all $2n + m$ edges of the form (h, x^*) and (c, h). Let D_x be the subset of D consisting of edges that are incident to the literals of variable x and let C be the set of clause nodes.

Simple inspection of cases show that if $|D_x| = 6$ then $D_x = A_x$ or $D_x = A_{\overline{x}}$.

If $|D_x| \geq 8$ we replace D_x with A_x and two edges $\overline{x} \to C$.

If D_x contains i edges to C, then $|D_x| \geq 4 + i$, because besides these edges D_x contains 4 edges to the literals of x. If $i = 4$ we are in the case already discussed.

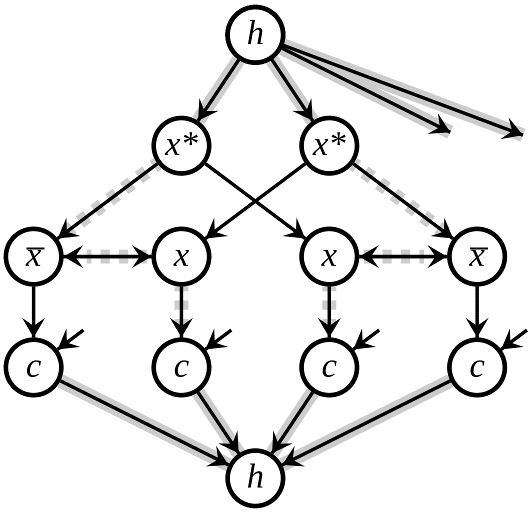

Figure 4: Illustration of our reduction. Marked edges are necessary. Dash-marked edges show set A_x that we can interpret it as $x =$true. If some i clause nodes are not reached (*i.e.*, the corresponding clause is not satisfied) then we need to add k extra edges. Thus, k unsatisfied clauses correspond to $8n + m + k$ edges being used ($6n - k$ deleted) and k satisfied clauses correspond to $8n + 2m - k$ edges being used ($6n + m - k$ deleted).

If $i = 3$, suppose that a clause where $\overline{x}$ occurs has no incoming edge in D_x; we can replace D_x with A_x plus one edge to a clause in which $\overline{x}$ occurs. If a clause where x occurs has no incoming edge in D_x, we perform a symmetric replacement of D_x.

In the remaining case $i_x \leq 2$ and $|D_x| = 7$, and we can perform a replacement as in the case of $i = 3$.

After all these replacements, the size of A did not increase while each D_x has the form of A_x or $A_{\overline{x}}$ plus some edges to C. If $A_x \subset D_x$ we assign x to true, otherwise to false. Clearly, if the union of D_x's has $6n + m - k$ edges, at most $m - k$ clauses are not satisfied by this truth assignment (those entered by "some other edges to C"), so if $|A| = 8n + 2m - k$, at least k clauses are satisfied.

Berman *et al.* [4] have a randomized construction of 2Reg-Max-SAT instances with $90n$ variables and $176n$ clauses for which it is NP-hard to tell if we can leave at most εn clauses unsatisfied or at least $(1 - \varepsilon)n$. The above construction converts it to graphs with $(14 \times 90 + 176)$ edges in which it is NP-hard to tell if we need at least $(8 \times 90 + 176 + 1 - \varepsilon)n$ edges or at most $(8 \times 90 + 176 + \varepsilon)n$, which gives a bound on approximability of Min-ED of $1 + 1/896$, and $1 + 1/539$ for Max-ED.

❑

Acknowledgments.

The authors thank Samir Khuller for useful discussions.

References

[1] A. Aho, M. R. Garey and J. D. Ullman. *The transitive reduction of a directed graph*, SIAM Journal of Computing, 1 (2), pp. 131-137, 1972.

[2] R. Albert, B. DasGupta, R. Dondi and E. Sontag. *Inferring (Biological) Signal Transduction Networks via Transitive Reductions of Directed Graphs*, to appear in Algorithmica.

[3] R. Albert, B. DasGupta, R. Dondi, S. Kachalo, E. Sontag, A. Zelikovsky and K. Westbrooks. *A Novel Method for Signal Transduction Network Inference from Indirect Experimental Evidence*, Journal of Computational Biology, Volume 14, Number 7, pp. 927-949, 2007.

[4] P. Berman, M. Karpinski and A. D. Scott. *Approximation Hardness of Short Symmetric Instances of MAX-3SAT*, Electronic Colloquium on Computational Complexity, Report TR03-049, 2003, available at `http://eccc.hpi-web.de/eccc-reports/2003/TR03-049/index.html`.

[5] V. Dubois and C. Bothorel. *Transitive reduction for social network analysis and visualization*, IEEE/WIC/ACM International Conference on Web Intelligence, pp. 128 - 131, 2005.

[6] J. Edmonds. *Optimum Branchings*, Mathematics and the Decision Sciences, Part 1, G. B. Dantzig and A. F. Veinott Jr. (eds.), Amer. Math. Soc. Lectures Appl. Math., 11, pp. 335-345, 1968.

[7] L. Engebretsen and M. Karpinski, *TSP with bounded metrics*, J. Comp. System Scie. 72 (2006), pp 509-546.

[8] S. Kachalo, R. Zhang, E. Sontag, R. Albert and B. DasGupta, *NET-SYNTHESIS: A software for synthesis, inference and simplification of signal transduction networks*, to appear in Bioinformatics.

[9] R. M. Karp. *A simple derivation of Edmonds' algorithm for optimum branching*, Networks, 1, pp. 265-272, 1972.

[10] S. Khuller, B. Raghavachari and N. Young. *Approximating the minimum equivalent digraph*, SIAM Journal of Computing, 24(4), pp. 859-872, 1995.

[11] S. Khuller, B. Raghavachari and N. Young. *On strongly connected digraphs with bounded cycle length*, Discrete Applied Mathematics, 69 (3), pp. 281-289, 1996.

[12] S. Khuller, B. Raghavachari and A. Zhu. *A uniform framework for approximating weighted connectivity problems*, 19th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 937-938, 1999.

[13] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*, Springer-Verlag, 2000.

[14] D.M. Moyles and G.L. Thompson, *Finding a minimum equivalent of a digraph*, J. ACM **16**(3), pp. 455-460, 1969.

[15] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, New York, 1994, page 212.

[16] A. Vetta. *Approximating the minimum strongly connected subgraph via a matching lower bound*, 12th ACM-SIAM Symposium on Discrete Algorithms, pp. 417-426, 2001.