Approximating Transitivity in Directed Networks

Piotr Berman*
Department of Computer Science & Engineering
Pennsylvania State University
University Park, PA 16802
Email: berman@cse.psu.edu

Bhaskar DasGupta†
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607-7053
Email: dasgupta@cs.uic.edu

Marek Karpinski‡
Department of Computer Science
University of Bonn
53117 Bonn, Germany
Email: marek@cs.uni-bonn.de

August 7, 2008

Abstract

We consider the minimum equivalent digraph (directed network) problem (also known as the strong transitive reduction) and its maximum objective function variant, with two types of extensions. First, we allow to declare set $\mathsf{D} \subset \mathsf{E}$ and require that a valid solution A satisfies $\mathsf{D} \subset \mathsf{A}$ (it is sometimes called transitive reduction problem). In the second extension (called p-ary transitive reduction), we have integer edge labeling and we view two paths as equivalent if they have the same beginning, ending and the sum of labels modulo p. A solution $\mathsf{A} \subseteq \mathsf{E}$ is valid if it gives an equivalent path for every original path. For all problems we establish the following: polynomial time minimization of $|\mathsf{A}|$ within ratio 1.5, polynomial time maximization of $|\mathsf{E}-\mathsf{A}|$ within ratio 2 and MAX-SNP hardness even if the length of simple cycles is limited to 5. Furthermore, we believe that the combinatorial technique behind the approximation algorithm for the minimization version might be of interest to other graph connectivity problems as well.

*Research partially done while visiting Dept. of Computer Science, University of Bonn and supported by DFG grant Bo 56/174-1.

†Supported by NSF grants DBI-0543365, IIS-0612044 and IIS-0346973.

‡Supported in part by DFG grants, Procope grant 31022, and Hausdorff Center grant EXC59-1.

1 Introduction

1.1 Definitions and motivation

Minimum equivalent digraph is a classic computational problem (cf. [13]) with several recent extensions motivated by applications in social sciences, systems biology etc.

The statement of the equivalent digraph problem is simple. For a digraph (V, E) the *transitive closure* of E is relation "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.

The assumption that the valid solutions are the equivalent digraphs of E yields two different optimization problems when we define two objective functions: Min-ED, in which we minimize $|A|$, and Max-ED, in which we maximize $|E - A|$. where A is an equivalent digraph for E.

Skipping condition (a) yields *transitive reduction* problem which is optimally solved by Aho *et al.* [1]. This could motivate renaming the equivalent digraph as a *strong transitive reduction* [14].

In the study of biological systems networks of interactions are considered, *e.g.* nodes can be genes and an edge (u, v) means that gene u *regulates* gene v. Without going into biological details, regulations may mean at least 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 used the term *de-repression*). Interactions of such two types can appear in other contexts as well, including social networks. This motivates an extension of the notion of digraph and its transitive closure described in points ①-③ below.

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 $\ell(P) = \sum_{i=1}^{k} \ell(u_{i-1}, u_i)$ **mod** p;

③ $\mathtt{Closure}_\ell(E) = \{(u, v, q) : \exists P$ 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 $\mathtt{Closure}_\ell(A) = \mathtt{Closure}_\ell(E)$.

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

1.2 Earlier results

The initial work on Min-ED by Moyles and Thomson [13] 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.* [10], with approximation ratio $1.617 + \varepsilon$ and by Vetta [15] with approximation ratio 1.5. The latter result did not have a full peer review, however.

If edges have costs, we can minimize $c(A)$ within factor 2 using an algorithm for minimum cost rooted arborescence [6, 8] of Edmonds (who described it) and Karp (who simplified it). We find minimum cost in- and out- arborescence in respect to an arbitrary root $r \in V$.

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 $2 + o(1)$-approximation for Min-TR$_p$*when* p *is a prime*. Other heuristics for these problems were investigated in [3, 7].

On the hardness side, Papadimitriou [14] formulated an exercise to show that strong transitive reduction is NP-hard, Khuller *et al.* have proven it formally and they also showed MAX-SNP hardness. Motivated by their *cycle contraction* method in [10], they were interested in the complexity of

the problem when there is an upper bound γ on the cycle length; in [9] 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 [15], 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. One surely 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 show 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 ratio 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 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 approximation algorithms for MIN-TR_p with ratio 1.5 and for MAX-TR_p with ratio 2 (we can compensate for the insertion of a single edge, so the ratio does not change) [2].

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

1.4 Some Motivations and Applications

Application of MIN-ED: Connectivity Requirements in Computer Networks. Khuller *et al.* [9] mentioned applications of MIN-ED to design of computer networks that satisfy given connectivity requirements.

If a set of connections exists already, then this application motivates MIN-TR_1 (cf. [11]).

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 and Cécile [5] applies MIN-TR_1 to the social network built upon interaction data (email boxes) of Enron corporation to study general properties (such as scale-freeness) of such networks and to help in the visualization process. They use a straightforward greedy approach which, as we have discussed, has inferior performance, both for MIN-TR_1 and MAX-TR_1.

Application of MIN-TR_2: Inferring Biological Signal Transduction Networks. In subsection 1.1 we motivated MIN-TR_2 with the study of gene regulatory networks. The same issues apply to other cellular interactions, like signal transduction networks and they were addressed by Albert *et al.* [3], with the use of an approximation algorithm MIN-TR_2.

[1] It appears that the approach of [15] may be correct, but the proofs and the description of the algorithm seem to have some gaps. It is somewhat difficult to point out these gaps without going into technical details; we will point out some problems as an illustration in the proof section of the paper.

[2] Albert *et al.* [2] did not use this approach and tried to approximate MIN-TR_p for $p > 1$ directly thus obtaining a $2 + o(1)$-approximation for prime p.

1.5 Our Techniques

Approximation algorithm for the MIN objective. Vetta used a primal/dual LP formulation for MIN-ED, more precisely, a solution that satisfies a subset of linear constraints, and the optimum solution for that subset is integer and it can be found using a maximum matching. We observed that a larger set of constraints also has this property, and that the extra edges for justified by this extension make it much easier to analyze the algorithm.

To tackle MIN-TR_1 problem we had to justify yet more edges, as the algorithm is not allowed to delete the required edges. We showed that we can use a yet larger set of constraints, with a "good enough" solution that can be found using a maximum weight matching.

We also used depth first search in a manner inspired by Tarjan's algorithm for finding strongly connected components.

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 for prime p. 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 chooses single edge. In conjunction 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 adapt the reduction used by Khuller *et al.* [10], but we apply it to a very restricted (and yet, MAX-SNP hard) version of MAX-SAT (cf. [4]).

1.6 Notation

We use the following additional notations.

- $G = (V, E)$ is the input digraph;
- $\iota(U) = \{(u, v) \in E : u \notin U \ \& \ v \in U\}$, $\iota(u_1, \ldots, u_k) = \iota(\{u_1, \ldots, u_k\})$; ;
- $o(U) = \{(u, v) \in E : u \in U \ \& \ v \notin U\}$, $o(u_1, \ldots, u_k) = o(\{u_1, \ldots, u_k\})$;
- $scc_A(u)$ 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 [13] 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 3.2 we will show the same for MIN-TR_p.

The minimum cost rooted arborescence problem on G is defined as follows. We are given a weighted digraph (V, E), 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 $c(A)$. An 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)

minimize $c \cdot x$ subject to

$$x \geq 0$$

$$\iota(U) \cdot x \geq 1 \text{ for all } U \text{ s.t. } \emptyset \subset U \subset V \text{ and } r \notin U \qquad (2.1)$$

Edmonds [6] and Karp [8] 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)
minimize $\mathbf{1} \cdot x$ 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. $\emptyset \subset U \subset V$	$\sum_{e \in \iota(U)} y_U \leq 1$ 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 a one-edge requirement $\{e\}$ (*i.e.* inequality $x_e \geq 1$) for each required edge e.

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

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

3.1.1 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 $T[v] \subset \mathrm{scc}_{T \cup B}(u)$.

(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 $T[u]$.

```
DFS(u)
{   COUNTER ←COUNTER+1
    NUMBER[u] ←LOWDONE[u] ←LOWCANDO[u] ←COUNTER
    for each edge (u, v)     // scan the adjacency list of u
        ifNUMBER[v] = 0
            INSERT(T, (u, v))      // (u, v) is a tree edge
            DFS(v)
            ifLOWDONE[u] > LOWDONE[v]
                LOWDONE[u] ←LOWDONE[v]
            ifLOWCANDO[u] > LOWCANDO[v]
                LOWCANDO[u] ←LOWCANDO[v]
                LOWEDGE[u] ←LOWEDGE[v]
        elseifLOWCANDO[u] > NUMBER[v]
            LOWCANDO[u] ←NUMBER[v]
            LOWEDGE[u] ← (u, v)
    // the final check: do we need another back edge?
    ifLOWDONE[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

Application of (A) to each child of v shows that $T[v] \subset \mathsf{scc}_{T \cup B}(v)$ 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.*

3.1.2 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. For each L-edge we give a credit of $1.5€$ and we allocate these credits to objects.

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 connecting it to an ancestor. Each solution edge costs $1€$.

Some object have enough money to pay for all L-edges inside, which make them 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.

When we discuss a small object A, we call it a *path node*, a *digon* or a *triangles* when $|A| = 1, 2, 3$ respectively.

These are the rules of allocating credits for L–edges to objects:

- L-edge inside object A: allocate to A;

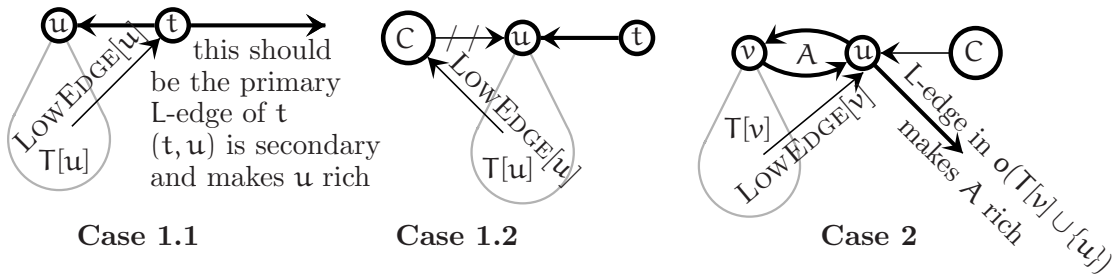

Figure 2: Illustrations for the cases of path nodes and digons.

- 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 $A \to B$: $0.5^{€}$ to B ($1^{€}$ to be allocated in the analysis of Min-TR_1).

Later we will formulate Rule ✯ to assure a desired property of primary edges.
The following lemma follows immediately from the definitions:

Lemma 1 *Object A is rich in each of the following cases:*

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 $|D| > 1$.*

3.1.3 Guiding Dfs

For a rich object A, use L-edges inside A in our solution, and 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.

3.1.4 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.

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 Dfs(u) is performed, and thus T[u] is already strongly connected. Dfs(u) inserts LowEdge[u]. Would this edge go to a proper ancestor of t, A would share (with $\{t\}$). Thus

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.

However, we can eliminate this situation by a rule of selecting the primary edges.

Rule ✯: *When* Dfs *visits a path node object* $\{u\}$*, it selects a primary edge, an* L*-edge* (u, v) *such that* Dfs(v) *is the first recursive call of* Dfs(u)*, in such a way that in* $V - \{u\}$ *a proper ancestor* $\{u\}$ *is reachable from* v*.*

To see that there exists (u, v) that satisfies Rule ✯ suppose that L-edges (u, v_i), $i = 1, \ldots, k$ fail this rule and S_i is the set of nodes reachable from v_i in $V - \{u\}$. Then $R = o(\{u\} \cup S_1 \cup \ldots S_k) \subset o(u)$ and R must contain a suitable L-edge.

Case 1.2: $C \neq \{t\}$. Initially we pay for $C \to u$ and 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 Dfs(t) introduces edge (t, u), paid by A using the money for the deleted edge, and the cost of LowEdge[u] is shared by A and $\{t\}$.

Note that our actual algorithm differs from Dfs in two ways: ① 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: $|A| = 2$**,** $A = \{u, v\}$**.** Dfs(u) starts with making the call Dfs(v).

We consider what happens when we execute the final check of 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 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 Dfs(v), and thus $T[v]$ is already strongly connected. Thus Dfs(v) inserts LowEdge[v] to B. If this edge goes to an ancestor of C, again, A pays for three edges only.

If LowEdge[v] goes to u, then $o(T[v]) \subset \iota(u)$, hence $R = o(T[v] \cup \{u\}) \subset o(u)$. Then R contains an L-edge hat exits u and does not go to v, and this means that A is rich.

Case 3: $|A| = 3$**,** $A = \{u, v, w\}$**.**

We use the following preprocessing. If a triple of nodes can be connected with a cycle (triangle) A, and it must contain at least two solution edges, we contract it to a single node (decreasing the optimum cost by at least 2), find a solution and then we insert triangle a to the solution (increasing the cost by 3). Clearly, this preserves approximation ratio 1.5.

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*. We will show that A can have a surplus of $0.5€$ by traversing A as follows: from $V - 1$, two edges inside A, back to $V - A$, so the balance is $4.5 - 4 = 0.5€$.

Suppose that all edges $(V - A) \to A$ enter through the same node, then an L-edge has to enter 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 $\textsc{Dfs}(u)$ 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, we can 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 that exits from t_1 are restricted to t_2, as we would have $o(\{t_1, t_2\}) \subset o(t_2)$, necessitating an L-edge exiting the digon and the digon would be rich.

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 subcases 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, and neither is Case 3.3.5.2.

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 $D_2 = \{t_2\}$. Our argument was that we can proceed to t_1, and if we cannot exit from t_1 to a node different than t_2 then the digon $\{t_1, t_2\}$ is rich as $o(t_1, t_2) \subset o(t_2)$. This argument does not work if we try to apply it to $T[t_1]$ and $T[t_2]$ rather than to t_1 and t_2.

However, if as $o(T[t_1] \cup \{t_2\}) \subset o(t_2)$ the argument is still valid, so it remains to address the case when we have an edge from $T[t_1]$ to $T[t_2] - \{t_2\}$. Then rather then using the edge from outside of $T[u]$ we change the depth first search as follows: when we reach t_1, we give the edge (t_1, t_2) the last priority. and the resulting $T[t_1]$ will contain some part of $T[t_2]$ and t_2 itself. Thus we get a path from A to A that includes at least 3 objects: t_1, t_2 and an object that belonged to $T[t_2]$, and this path delivers a surplus to A.

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. When we form "lower bound" edge set L we clearly have $D \subset L$, but the algorithm in some cases fails to include L-edges in the solution. 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.

If an L–edge is not "noticed" by the algorithm, then it was not considered in the lower bound used to justify the edges of the solution, so when we insert this edge, we can also "notice" its contribution to the lower bound.

As an obvious addition to pre-processing, cycles of D-edges can be collapsed to single nodes, so we can assume that such cycles do not exist.

Another clarification that makes case analysis more clear is the choice of a root object for DFS. We can always make sure that it is a cycle with no entering L-edges.

3.2.1 Digons with D-edges

A *problematic digon* $\{u, v\}$ contains a non-D-edge (u, v) and a D-edge (v, u). A problematic digon can be adopted as a digon of L and subsequently it can cause the algorithm for MIN-ED to "malfunction" *i.e.* to remove its D-edge. Hence we need to alter the algorithm and the analysis.

We start with changing the rules for allocating credits for L-edges that were formulated in Subsection 3.1.2; consider an L edge between two objects, $P \to Q$:

if this is a secondary edge, or $|P| > 2$, allocate $0.5€$ to P, $1€$ to Q.

This change merely takes care of credits that were not used in the analysis of MIN-TR$_1$ algorithm.

A problematic digon exited by a D-edge

If such a digon becomes a cycle object in L, it is exited by an L edge, thus it is rich, and the algorithm does not remove its edges.

A problematic digon entered by a D-edge

When such a problematic digon becomes a digon of L, it has to be the ending object of an L-path, say $Q_1 \to \ldots \to Q_k$, where all edges but the first one are primary edges exiting path nodes, and the first edge is either secondary or it exits a cycle. Initially, assume the latter, moreover, assume that Q_1 is also a problematic digon; the remaining cases are easier because either we have extra $0.5€$ to balance the credits with edges forming the solution, or we have an option to save by removing an edge from Q_1.

Together, we have k objects with $k+3$ edges and thus, $1.5k+4.5€$ credits, of which 4 are used for the edges in the two cycles that start and end the path. This leaves $1.5k+0.5€$ credits for the T- and B-edges for these objects.

The edges of the path are always selected by the algorithm, either as T-edges, or, if Q_{i+1} is visited before Q_i, $Q_i \to Q_{i+1}$ is adopted as a B-edge.

In the analysis, an object pays for the entering T-edge, while the cost of connecting it to the ancestors, or introducing a B-edge, can be shared—or not. Intuitively, we have a problem because we have the balance if we share in all k instances with one exception, but the arguments made in the previous section allow for two exceptions.

Let us recall those arguments (Case 1 in Subsection 3.1.4). Consider edges introduced on behalf of Q_i, T-edge $R \to Q_i$ from its parent object and a back edge that is found or introduced when we perform the final check in $\textsc{Dfs}(Q_i)$. If this back edge is introduced before that check, by a child call, its cost is shared with that child. Otherwise, $T[Q_i]$ is already strongly connected and we select the "best possible" back edge. If this edge goes beyond the parent object R, the cost is shared with R. Otherwise we have two cases.

Case A1: $R = Q_{i-1}$. We use the fact that $o(T[Q_i]) \subset \iota(Q_{i-1})$. If $|Q_{i-1}| = 1$ then we know about a "nice requirement" contained in $o(Q_{i-1})$ that does not include $Q_{i-1} \to Q_i$, and we argued that in that case $Q_{i-1} \to Q_i$ would not be a primary edge (Case 1.1). This argument fails for $i \le 2$; if we have a balance better by $0.5€$, as it happens when $Q_1 \to Q_2$ is a secondary edge or $|Q_1| > 2$, it is still fine. If Q_1 is a non-problematic digon $\{u, v\}$ and $Q_1 \to Q_2 = u \to Q_2$, we can select v as the target node of the B-edge from $T[Q_2]$ to Q_1 so we can delete edge (v, u) and improve the balance. If we select such a back edge, $o(T[Q_2]) \subset \iota(u)$, thus, an L-edge enters Q_1, and we would analyze Q_1 like a path node that cannot use the secondary edge argument but with an extra $1€$)

Now we can assume that $R = Q_1$ is a problematic digon as described above and $(u, v) \in D$. If Q_2 fails to share but Q_1 does, we still have satisfactory balance (only one exception from sharing among $Q_1,]ldots, Q_k$. Otherwise, we can change the connections of $S = R \cup T[Q_2]$ with $V - S$. First, we choose an edge from $T[Q_2]$ to R so it enters u. Second, we choose an edge from S to $V - S$ so it exits v (it has to exit u or v, if the latter is not possible, we have an L-edge from u to $V - S$ and we can set $Q_1 \to Q_2$ to be a secondary L-edge of Q_1). Then we select an edge from $V - S$ to S so it enters $S - \{v\}$.

Case A2: $R \neq Q_{i-1}$.

We "prepare" $Q_i \to R$ as the back edge but we delete T-edge $R \to Q_i$ and wait until Q_{i-1} is visited later; then we introduce $Q_{i-1} \to Q_i$ as the T-edge and the back edge $Q_i \to R$ is shared with Q_{i-1}.

This argument fails if Q_{i-1} is visited during $\textsc{Dfs}(Q_i)$, but we can "repair" it.

In the first attempt we check if after the completion of $\textsc{Dfs}(Q_i)$ the search visits an object P with an edge to some Q_j, $j < i$. Then we introduce $P \to Q_j$ and delete edge $R \to Q_i$, and more generally, to each object on on the list $Q_j, Q_{j+1}, \ldots, Q_i$ if that object does not share. If nothing else, we decreased the number of the last object on our list $Q_1, \ldots, Q_k$ that does not share. The same applies if we encounter P with edge $P \to P'$ where P' is on a T path from Q_i to Q_j.

The second attempt we discuss in two cases.

Case A2.1: either $i = k$ (so Q_j is the final object on our Q-path) or Q_{i+1} was visited before Q_i, hence $Q_{i+1}, \ldots, Q_k$ are parts of the parent object R at the time $\textsc{Dfs}(Q_i)$ starts.

If $i = 1$, we got only non-sharing object on Q-path, so it is fine.

If $i = 2$ and Q_1 does not share, we can access Q_1 only by paths that start at Q_2. This produces evidence of extra L-edges unless $k = 2$, so $|Q_2| = 2$ Suppose we have only direct edges from Q_2 to Q_1, then $Q_1 \cup Q_2$ must contain at least 4 solution edges: 3 D edges, inside Q_1, inside Q_2 and $Q_1 \to Q_2$, plus an edge $Q_2 \to Q_1$, while we can connect $Q_1 \cup Q_2$ with 6 edges. Because $6/4 \le 1.5$ we can collapse $Q_1 \cup Q_2$ into a single node during pre-processing.

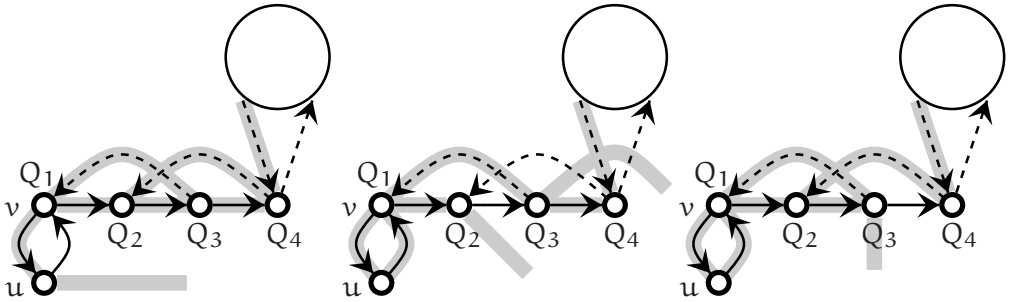

Figure 3: Case A2.1, $i = 4$. Left panel: if we can avoid the use of an edge that starts at u and is contained in $S = Q_1 \cup Q_2 \cup Q_3 \cup Q_4$, we can exit S at u and save edges $v \to Q_2$ and $Q_4 \to R$. Middle panel: if we can avoid the use of an edge that starts at Q_2 and is contained in S we save similarly, because we can enter S at Q_3 (if we can enter at Q_1, Q_2 we save even simpler, and we have extra lower bound if we can enter only at Q_4). Right panel: if we can avoid an edge that starts at Q_3 and is contained in S, we can save. If we cannot avoid either one, together with D-edges we must have 7 edges inside S, which allows to collapse it in the preprocessing.

If we also have paths from Q_2 to Q_1 through some intermediate objects forming set S, one can easily show that either we can save an edge in the balance or for every node in S we have an evidence that $Q_1 \cup Q_2 \cup S$ must contained an additional edge, to that node, and the sufficient number of edges that would make $Q_1 \cup Q_2 \cup S$ increases by at most $1.5|S|$. To simplify the proof, we will continue as if all connections between Q-objects were direct edges.

If $i = 3$, then we will have an evidence for extra L-edges entering or exiting Q_2 unless we have an edge from Q_3 to Q_1, in which case Q_1 and Q_2 share a back edge and we have a good balace. This argument works for every odd j.

If $i = 4$, we have a good case when there is an edge $Q_4 \to Q_1$, because Q_1, Q_2, Q_3 share. Otherwise, without more L-edges incident to these objects we have edges $Q_4 \to Q_2$ and $Q_3 \to Q_1$ and we can apply the reasoning from the caption of Fig. 3.

This argument can be generalized for every odd value of k.

Case A2.2: $Q_{i+1} \subset T[Q_i]$. Then the only way it may happen that Q_i does not share is if $T[Q_i]$ contains the entire Q-path $Q_1 \to \dots Q_k$, and we apply the reasoning of Case 2.2 as in Case 2.1, as if we had $i = k$.

A problematic digon with two in-requirements

When we have a problematic digon Q, we could have more than one sink strongly connected components in $V - Q$, say, S_1 and S_2. In that case $V - Q$ also has a source component, say S_3, that is different from S_1 and S_2. We collapse Q to a single node, and the "hidden balance" is $-0.5€$, because we credit of $1.5€$ for the D-edge of Q and two edges in the solution. However, Q can "collect" $0.5€$ from the slack in the analysis. One can see that node Q will be either a path node with two incoming L-edges or a node on a cycle of length at least 3 with an incoming L-edge. The former was an easy case for L-cycles that are entered by L-edges. The latter can be analyzed as follows. the edge satisfying requirement $o(S_i)$ comes from object Q_i. If one of these two edges is secondary, we are getting the required surplus. Otherwise, assume that Q_2 is visited after Q_1, so the first step of $\textsc{Dfs}(Q_2)$ is making a B-edge to Q. Then Q_2 is on a path from a rich cycle object that either was visited after Q_1 and Q, or there is a secondary edge on this path. A secondary edge "sends surplus", and a cycle object visited after Q saves an edge because it uses an L-edge it "owns" as its B-edge.

A problematic digon with two out-requirements

This is a similar case, except that $V - Q$ has two source strongly connected components. After collapse, Q becomes a path node with a secondary edge going out, or a node on a rich cycle with at least 3 edges. Either way, we can get the extra $0.5^{\in}$ to balance the deficit of Q.

Worrisome digons

After modifications described so far, we guaranteed that many problematic digons will not have D-edges removed by the algorithm, either by collapsing sets that contain them and ensuring a desired solution inside those sets, or by explicit rules that forbid such removal and yet achieve the balance.

Remaining problematic digons are disjoint because they do not have D-edges that would enter or exit. Moreover, they do not have entering or exiting L-edges. This implies, for example, that if $Q = \{u, v\}$ is a remaining problematic digon, there is only one requirement contained in $o(u)$, otherwise one such requirement would not contain edge (u, v) and it would have to be satisfied by an L-edge exiting Q. Finally, $V - Q$ has only one source and one sink strongly connected components. We will refer to the remaining problematic digons as *worrisome*.

For the sake of worrisome digons we will altering the linear program P3 – which keeps those requirements $Rx \geq 1$ of P2 that for some node u satisfy $R \subseteq o(u)$ or $R \subseteq \iota(u)$, let us call them nice requirements. The dual D3 is to find a maximum size collection of nice requirements such that no two can be satisfied by a single edge (i such requirements $\Longrightarrow$ we need i edges). In principle, such packing can be "fractiona", but in this case it is related to maximum matching, so the fractional optimum is integer/combinatorial.

We will form a new dual program, D4. For a node w that does not belong to a worrisome digon, as before we introduce requirements contained in $o(w)$ and in $\iota(w)$. For a worrisome digon with D-edge (v, u), we have the one-edge requirement $\{(v, u)\}$, and thus no other requirements contained in $o(v)$ and $\iota(u)$. Also, rather than having requirements $o(u)$ and $\iota(v)$ we form requirement as follows:

- if there is more then one requirement contained in $o(u, v) = o(\{u, v\})$ we will have these requirements;
- if there is only one minimal requirement R contained in $o(u, v)$, we will form a *sibling pair* of requirement: R, for simplicity referred to as as $o(u, v)$, as well as $o(u)$, both with coefficient 0.5.
- if there is more then one requirement contained in $\iota(u, v)$) we will have these requirements;
- if there is only one minimal requirement R contained in $\iota(u, v)$, we will have a sibling pair of requirements with coefficients 0.5, R, referred to it as $o(u, v)$, as well as $\iota(v)$.

In P4 an edge can satisfy more than two requirements, but the sum of coefficients satisfied by an edge is at most two.

Assume that the sum of coefficients of requirements in P4 is N and consider a set of edges L^* that satisfies all of them. For every requirement we give credit to one of the L^*-edges that satisfy them. Say that a net credit of an edge is the sum of credits it received minus one; then the size of L^* is N minus the sum of net credits of edges of L^*.

Now, consider a set of edges M of L^* with positive net credits; the lower bound for the size of $L*$ is N minus the sum of net credits in M. We can assign credits in such a way that M can be viewed as a matching.

Consider a sibling pair of requirements, $o(u)$ and $o(u, v)$ (the same observation will hold for $\iota(v)$ and $\iota(u, v)$). The only way to satisfy the former without satisfying the latter is with edge (u, v); this edge satisfies only two requirements, both with coefficient 0.5, so it cannot belong to M. Thus

if an edge in M gets credit for satisfying $o(u)$ we can also give it the credit for satisfying $o(u, v)$; in this manner only one edge can get credit from a sibling pair of requirements.

This defines a bipartite graph: "nodes" are requirements with coefficient 1 and sibling pairs, edges are pairs of "nodes" in which requirements can be satisfied simultaneously, edge weights are net credits. We can find M as a matching in this graph with the maximum weight.

Once we find M, we can complete the lower bound by taking account of the credits not distributed to M:

(a) If we have a requirement with coefficient 1 that did not give credit to M, we can add an edge that satisfies it, the same can be done for a sibling pair of requirements that did not give any credits to M (note that we can satisfy a sibling pair with a single edge).

(b) In the case of a sibling pair that gave 0.5 credit to M, its digon retains 0.5 credit, and we use it to collapse this digon to a single node: collapsed digon uses two edges, and it has $1.5^{\in}$ for its D-edge and $0.5^{\in}$ for its remaining credit.

After applying (a-b), all credits are used up and all requirements of P4 are satisfied, either within the computed lower bound, or using the "extra credit" from collapsed D-edges. Moreover, for each new node x_{uv} (a collapsed worrisome digon) we have edges that satisfy all the minimal requirements contained in $o(x)uv$ and in $\iota(x_{uv})$.

However, we may have the following pathology: in the resulting set of edges L we may have nodes with no edge exiting (or no edge entering). For no edge entering, it may happen like that: for a worrisome digon $\{u, v\}$ with D-edge (v, u) we have multiple requirements contained in $\iota(u, v)$, and we satisfied all of them with edges that enter u. We will try to correct this as follows: if one of these edges exits a cycle, we replace it with an edge that also satisfies this requirement, but which enters v. Thus the pathology remains if none of at least two L-edges entering $\{u, v\}$ exits a cycle. One of these edges can be on a cycle that contains u, but at least one enters $\{u, v\}$ from a path node (or a worrisome digon that can be considered like a path node). We can collapse a worrisome digon that exhibits such a pathology, and because it becomes either a path node with two edges entering from other path nodes, or a part of a cycle that is entered with an edge from a path node, we can collect $0.5^{\in}$ for the use within $\{u, v\}$.

Of course, a similar pathology and its resolution may happen when we have multiple L-edges exiting $\{u, v\}$.

3.2.2 Triangles

Our algorithm for Min-ED 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 can 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 $0.5^{\in}$.

One can see that a free triangle without required edges and which is not collapsed in preprocessing also saves an edge and arrives at a surplus.

We can summarize this section with the following theorem:

Theorem 2 *There is a polynomial time algorithms 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$_1$

Theorem 3 *There is a polynomial time algorithms 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 [8], 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$_p$ for prime 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. For simplicy. we discuss the case of MIN-TR$_p$, but every statement applies to MAX-TR$_p$ as well.

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$_1$ can be transformed into a solution to MIN-TR$_p$ by adding a single edge, and in polynomial time (proportional to p) we can find that edge.

In turn, when we run an approximation algorithms within strongly connected components, we obtain its approximation ratio even if we add one extra edge (it is actually a property of our algorithms, but in any case one can try to guess correctly several solution edges and save an additive constant from the approximation).

Consider an instance (V, E, ℓ, D) of MAX-TR$_p$. The following proposition says that it suffices to restrict our attention to strongly connected components of (V, E)[3].

[3]The authors in [2] prove their result only for MIN-TR$_p$, but the proof applies to MAX-TR$_p$ as well.

Proposition 4 [2] *Let $\rho > 1$ be a constant. If we are given ρ-approximation of* Max-TR$_p$ *for each strongly connected component of (V, E), we can compute in polynomial time a ρ-approximation for (V, E).*

The following characterization of strongly connected graphs appears in [2].

Lemma 5 [2] *Let $(C, E[C], \ell, D)$ is an instance of* Max-TR$_p$*. Every strongly connected component C of (V, E) is one of the following two types:*

(Multiple Parity Component) $|\{a \in \mathbb{Z}_p : (u, v, a) \in \mathsf{Closure}_\ell(E)\}| = p$ *for any $u, v \in C$;*

(Single Parity Components) $|\{a \in \mathbb{Z}_p : (u, v, a) \in \mathsf{Closure}_\ell(E)\}| = 1$ *for any $u, v \in C$.*

Moreover, C is a multiple parity component if and only if it contains a simple cycle of non-zero parity.

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 $C \subset V$ we consider an induced instance of Min-TR$_1$, $(C, E(C), D \cap C)$. We find an approximate solution A_C that contains an out-arborescence T_C with root r. We label each node $u \in C$ with $\ell(u) = \ell(P_u)$ where P_u is the unique path in T_C from r to u.

Now for every $(u, v) \in E(C)$ we check if $\ell(v) = \ell(u) + \ell(u, v)$ **mod** p.

If this is true for every $e \in E(C)$ then C is a single parity component. Otherwise, we pick a single edge (u, v) violating the test and we insert it to A_C. This is sufficient because A_U contains a path Q from v to r, and the cycles $(P_u, (u, v), Q)$ and (P_v, Q) have different parities, hence one of them is non-zero.

6 Integrality Gap of the LP Formulation for Min-ED and Max-ED

Lemma 6 *The primal LP formulation 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 $0 \le i < 2$ and $0 \le i \le n$. 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 $U \neq \emptyset$ implies $|\iota(U)| \ge 2$. 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 MAX-SNP-hardness Results

Theorem 7 *Let* k-MIN-ED *and* k-MAX-ED *be the* MIN-ED *and* MAX-ED *problems, respectively, restricted to graphs in which the longest cycle has* k *edges. Then,* 5-MIN-ED *and* 5-MAX-ED *are both MAX-SNP-hard.*

Khuller *et al.* used a reduction from 3-SAT to MIN-ED. We are basically using their reduction, but we restrict its application to sets of clauses in which each literal occurs exactly twice (and each variable, four times). It was shown in [4] that this restriction yields a MAX-SNP hard problem.

More precisely, 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. 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.

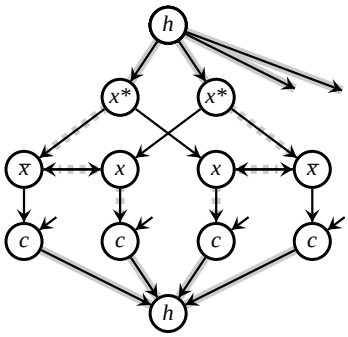

Figure 4: Illustration of our reduction. Marked edges are necessary. Dash-marked edges show set A_x that we can interpret it as x =true. If some i clause nodes are not reached (*i.e.*, the corresponding clause is not satisfied) then we need to add k extra edges. Thus, k unsatisfied clauses correspond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

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*, Algorithmica, 51 (2), pp. 129-159, 2008.

[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, 14 (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] S. Kachalo, R. Zhang, E. Sontag, R. Albert and B. DasGupta. *NET-SYNTHESIS: A software for synthesis, inference and simplification of signal transduction networks*, Bioinformatics, 24 (2), pp. 293-295, 2008.

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

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

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

[11] 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.

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

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

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

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