algorithms

MDPI

Article

Adding Edges for Maximizing Weighted Reachability †

Federico Corò [1,*], Gianlorenzo D'Angelo [2] and Cristina M. Pinotti [3]

[1] Department of Computer Science, Sapienza University of Rome, 00161 Rome, Italy
[2] Gran Sasso Science Institute (GSSI), 67100 L'Aquila, Italy; gianlorenzo.dangelo@gssi.it
[3] Department of Computer Science and Mathematics, University of Perugia, 06123 Perugia PG, Italy; cristina.pinotti@unipg.it
* Correspondence: federico.coro@uniroma1.it
† This paper is an extended version of our paper published in the 14th International Symposium on Algorithms and Experiments for Wireless Networks (ALGOSENSOR 2018).

Received: 23 December 2019; Accepted: 15 March 2020; Published: 18 March 2020

Abstract: In this paper, we consider the problem of improving the reachability of a graph. We approach the problem from a graph augmentation perspective, in which a limited set size of edges is added to the graph to increase the overall number of reachable nodes. We call this new problem the *Maximum Connectivity Improvement* (MCI) problem. We first show that, for the purpose of solve solving MCI, we can focus on Directed Acyclic Graphs (DAG) only. We show that approximating the MCI problem on DAG to within any constant factor greater than $1 - 1/e$ is *NP*-hard even if we restrict to graphs with a single source or a single sink, and the problem remains *NP*-complete if we further restrict to unitary weights. Finally, this paper presents a dynamic programming algorithm for the MCI problem on trees with a single source that produces optimal solutions in polynomial time. Then, we propose two polynomial-time greedy algorithms that guarantee $(1 - 1/e)$-approximation ratio on DAGs with a single source, a single sink or two sources.

Keywords: graph augmentation; approximation algorithms; greedy algorithms; submodularity; DAG; trees; dynamic programming

1. Introduction

The problem of improving the reachability of a graph is an important graph-theoretical question, which finds applications in several areas. There are several recent possible application scenarios for this problem, for example suggesting friends in a social network in order to increase the spreading of information [1–3], reducing the convergence time of random walk processes to perform faster network simulations [4,5], improving wireless sensor networks resilience [6] and even control elections in social networks [7,8].

In this paper, we approach the problem from a graph augmentation perspective, which is the problem of adding a set of non-existing edges to a graph to increase the overall number of reachable nodes. In traditional graph theory, this problem was first studied by Eswaran and Tarjan [9]. They considered the problem of adding a minimum-cost set of edges to a graph such that the resulting graph satisfies a given connectivity requirement, e.g., to make a directed graph strongly connected or to make an undirected graph bridge-connected or biconnected. In their seminal paper, Tarjan et al. presented linear time algorithms for many graph augmentation problems and proved that some variants are instead *NP*-complete.

More recently, several optimisation problems related to graph augmentation have been addressed. Several papers in the literature deal with the problem of minimising the eccentricity of a graph by adding a limited number of new edges [10–13].

The problem of minimising the average all-pairs shortest path distance—characteristic path length—of the whole graph was studied by Papagelis [5]. The author considered the problem of adding a small set of edges to minimise the characteristic path length and proves that the problem is *NP*-hard. He proposed a path screening technique to select the edges to be added. It is worth noting that the objective function in [5] does not satisfy the submodularity property, and so good approximations cannot be guaranteed via a greedy strategy. The problem of adding a small set of links to maximise the centrality of a given node in a network has been addressed for different centrality measures: page-rank [4,14], average distance [15], harmonic and betweenness centrality [16,17] and some measures related to the number of paths passing through a given node [18]. It is worth noting that many traditional graph problem can also be applied to dynamic graphs. Graph problems close to ours studied in the dynamic setting include connectivity [19], minimum spanning tree [20] and reachability [21,22].

In this work, which is based on and extends previous preliminary research in this direction [23], we study the problem of adding at most B edges to a directed graph in order to maximise the overall weighted number of reachable nodes, which we call the *Maximum Connectivity Improvement* (MCI) problem. The rest of the paper is organised as follows. We first show that we can focus on Directed Acyclic Graphs (DAG) without loss of generality (Section 2). Then, we focus on the complexity of the problem (Section 3) and we prove that the MCI problem is *NP*-hard to approximate to within a factor greater than $1 - \frac{1}{e}$. This result holds even if the DAG has a single source or a single sink. Moreover, the problem remains *NP*-complete if we further restrict to the unweighted case. In Section 4, we give a dynamic programming algorithm for the case in which the graph is a rooted tree, where the root is the only source node. In Section 5, we present a greedy algorithm which guarantees a $(1 - 1/e)$-approximation factor for the case in which the DAG has a single source or a single sink. As a first step in the direction of extending our approach to general DAGs, i.e., multiple sources and multiple sinks, in Section 6, we tackle the special case of DAGs with two sources providing a constant factor approximation algorithm to solve the problem. We end with some concluding remarks in Section 7. Compared to the conference version, we add the results in Section 6 on DAGs with two sources and we revise the paper in order to improve readability, by adding examples and detailed proofs.

2. Preliminaries

Let $G = (V, E)$ be a directed graph. Each node $v \in V$ is associated with a non-negative weight $w_v \in \mathbb{N}_{\geq 0}$ and a profit $p_v \in \mathbb{N}_{\geq 0}$. Given a node $v \in V$, we denote by $R(v, G)$ the set of nodes that are reachable from v in G, that is $R(v, G) = \{u \in V : \exists \text{ path from } v \text{ to } u \text{ in } G\}$. Moreover, we denote by $p(R(v, G)) = \sum_{u \in R(v,G)} p_u$ the sum of the profits of the nodes reachable from v in G. In the rest of the paper, we also use the form $p(R(v, G) \setminus R(u, G)) = \sum_{u \in R(v,G) \setminus R(u,G)} p_u$ to denote the sum of the profits of the nodes in G that are reachable from v, but not from u. Note that, in the case $R(u, G) \subseteq R(v, G)$, it holds $p(R(v, G) \setminus R(u, G)) = p(R(v, G)) - p(R(u, G))$. Given a set S of edges not in E, we denote by $G(S)$ the graph augmented by adding the edges in S to G, i.e., $G(S) = (V, E \cup S)$. Let $R(v, G(S))$ and $p(R(v, G(S)))$ be, respectively, the set of nodes that are reachable from v in $G(S)$ and the sum of the profits of the nodes in $R(v, G(S))$. Note that, augmenting G, the connectivity cannot be worse, and thus: $R(u, G) \subseteq R(u, G(S))$. Let $f(G) = \sum_{v \in V} w_v p(R(v, G))$ be a weighted measure of the connectivity of G. When weights and profits are unitary, $f(G)$ represents the overall number of connected pairs in G.

In this paper, we aim to augment G by adding a set S of edges of at most size B, i.e., $|S| \leq B$ and $B \in \mathbb{N}_{\geq 0}$, that maximises the weighted connectivity of $f(G(S))$. We call this problem the *Maximum Connectivity Improvement* (MCI) problem because maximising $f(G(S))$ is the same as maximising $f(G(S)) - f(G)$. Formally,

Definition 1 (Maximum Connectivity Improvement). *Given a graph $G = (V, E)$, given for each node $v \in V$ a weight $w_v \in \mathbb{N}_{\geq 0}$ and a profit $p_v \in \mathbb{N}_{\geq 0}$ and given a budget $B \in \mathbb{N}_{\geq 0}$, we want to find a set of edges S^* such that*

$$S^* = \underset{S \subseteq V \times V : |S| \leq B}{\arg\max} f(G(S)).$$

From now on, for simplicity, we omit from the notations the original graph G. Thus, we simply use $R(v)$ and $R(v, S)$ to denote $R(v, G)$ and $R(v, G(S))$, respectively. Similarly, we simply denote with f and $f(S)$ the value of the weighted connectivity in G and in $G(S)$, respectively.

At first, we show how to transform any directed graph G with cycles into a *Directed Acyclic Graph* (DAG) $G' = (V', E')$ and how to transform any solution for G' into a feasible solution for G.

Graph $G' = (V', E')$ has as many nodes as the number of strongly connected components of G, i.e., there is one vertex in V' for each strongly connected component of G. Specifically, G' selects one representative node for each strongly connected component of G and G' adds one directed edge between two nodes u' and v' of G' if there is a directed edge in G connecting any vertex of the strongly connected component represented by u' with any vertex of the strongly connected component represented by v'. Graph G' is called *condensation* of G and can be computed in $\mathcal{O}(|V| + |E|)$ time by using Tarjan's algorithm which consists in performing a DFS visit [24].

The weight and the profit of a node v' in G' is given by the sum of the weights and profits of the nodes of G that belong to the strongly connected component $C_{v'}$ that is represented by v', i.e., $w_{v'} = \sum_{v \in C_{v'}} w_v$ and $p_{v'} = \sum_{v \in C_{v'}} p_v$. Note that, when the profits are unitary in G, $p_v = 1 \ \forall v \in V$, then $p_{v'}$ is equal to the size of the strongly connected component $C_{v'}$ associated to v'.

Since the condensation preserves the connectivity of G, the following lemma can be proved:

Lemma 1. *Given a graph G and its condensation G', it yields: $f(G') = f(G)$.*

Proof. First, consider two nodes u and v that belong to the same strongly connected component $C_{v'}$ in G'. Clearly, $R(u, G) = R(v, G)$.

Moreover, it holds that $p(R(v, G)) = p(R(v', G'))$ because $R(v', G')$ contains one node for each different strongly connected component in $R(u, G)$ and thus:

$$p(R(v', G')) = \sum_{u' \in R(v', G')} p_{u'} = \sum_{u' \in R(v', G')} \sum_{u \in C_{u'}} p(u) = \sum_{u \in R(v, G)} p(u) = p(R(v, G))$$

Denoting $C_{v'}$ the strongly connected component represented by v', we have:

$$\begin{aligned}
f(G') &= \sum_{\mathbf{v}' \in V'} w_{\mathbf{v}'} p(R(\mathbf{v}', G')) \\
&= \sum_{\mathbf{v}' \in V'} w_{\mathbf{v}'} \left(\sum_{u' \in R(\mathbf{v}', G')} \sum_{u \in C_{u'}} p(u) \right) = \sum_{\mathbf{v}' : C_{\mathbf{v}'} \in \mathcal{C}} w_{\mathbf{v}'} \left(\sum_{u' \in R(\mathbf{v}', G')} \sum_{u \in C_{u'}} p(u) \right) \\
&= \sum_{\mathbf{v}' : C_{\mathbf{v}'} \in \mathcal{C}} \left(\sum_{v \in C_{\mathbf{v}'}} w_v \right) \left(\sum_{u' \in R(\mathbf{v}', G')} \sum_{u \in C_{u'}} p(u) \right) = \sum_{\mathbf{v}' : C_{v'} \in \mathcal{C}} \sum_{v \in C_{\mathbf{v}'}} \left(w_v \left(\sum_{u' \in R(\mathbf{v}', G')} \sum_{u \in C_{u'}} p(u) \right) \right) \\
&= \sum_{\mathbf{v}' : C_{\mathbf{v}'} \in \mathcal{C}} \sum_{v \in C_{\mathbf{v}'}} \left(w_v \sum_{u \in R(v, G)} p(u) \right) = \sum_{\mathbf{v}' : C_{\mathbf{v}'} \in \mathcal{C}} \sum_{v \in C_{\mathbf{v}'}} w_v p(R(v, G)) \\
&= \sum_{v \in V} w_v p(R(v, G)) = f(G)
\end{aligned}$$

□

Given a solution S' for the MCI problem in G', we can build a solution S with the same value for the MCI problem in G as follows: For each edge (u', v') in S', we add an edge (u, v) in S, where u and v are two arbitrary nodes in the connected component corresponding to u' and v', respectively.

This derives from the fact that applying the condensation algorithm to $G' \cup S'$ or to $G \cup S$ we obtain the same condensed graph, say G''. From Lemma 1, we can conclude that $f(G' \cup S') = f(G'') = f(G \cup S)$.

Observe that, if we add an edge e within the same strongly connected component in G, we do not add any edge to G'. Since the condensation G'' of $(G \cup \{e\})$ is the same as G', we have $f(G \cup \{e\}) = f(G') = f(G)$. Hence, we can assume, without loss of generality, that any solution to MCI in G does not contain any edge within a unique strongly connected component since such edge does not improve the objective function. As a consequence, in the remainder of the paper, we assume that the graph is a DAG.

Given a DAG, we can distinguish between three kinds of nodes: *sources*, nodes with no incoming edges; *sinks*, nodes with no outgoing edges; and the rest of the nodes. The next lemma allows us to focus on solutions that contain only edges connecting sink nodes to source nodes. In the remainder of the paper, we use this property to derive algorithms to solve both the simple case with binary trees and the more general case with DAGs (with single or double source and single sink).

Lemma 2. *Let S be a solution to the MCI problem, then there exists a solution S' such that $|S| = |S'|$, $f(S) \leq f(S')$, and all edges in S' connect sink nodes to source nodes.*

Proof. We show how to modify any solution S in order to find a solution S' that contains only nodes that connect sink nodes to source nodes with the same cardinality and such that $f(S) \leq f(S')$. To obtain S', we start from S and we repeatedly apply the following modifications to each edge (u, v) of S such that u is not a sink or v is not a source:

1. If u is not a sink, then there exists a path from u to some sink u' and we swap edge (u, v) with edge (u', v). The objective increases at least by the sum of the weights on a path from u to u'. Namely, after adding the edge (u', v), any node z on the path from u to u' now reaches v passing through u'. Note that the objective function does not decrease and, instead, may increase due to the fact that the nodes z now are able to reach the node v.
2. If v is not a source, then there exists a path from a source v' to v and we swap edge (u, v) with edge (u, v'). The objective function does not decrease and increases at least by the number of nodes in a path from v' to v multiplied by w_u.

Note that, in both cases the gain of a node on the path we are extending can be zero if it was already able to reach the source/sink from another edge in the solution. Since we have neither added nor removed any edge, the cardinality of the new solution S' is equal to the previous solution S. Furthermore, since the objective function can only increase by modifying the edges according to the rules described above, we have that $f(S) \leq f(S')$. □

3. Hardness Results

In this section, we first show that the MCI problem is *NP*-complete, even in the case in which all the weights and profits are unitary and the graph contains a single sink node or a single source node. Then, we show that it is *NP*-hard to approximate MCI to within a factor greater than $1 - \frac{1}{e}$. This last result holds also in the case of graphs with a single sink node or a single source node, but not in the case of unitary weights.

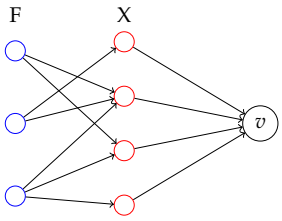
Theorem 1. *MCI is NP-complete, even in the case in which all the weights and profits are unitary and the graph contains a single sink node or a single source node.*

Proof. We consider the decision version of MCI in which all the weights and profits are unitary (i.e., $w_v = p_v = 1$): Given a directed graph $G = (V, E)$ and two integers $M, B \in \mathbb{N}_{\geq 0}$, the goal is to find

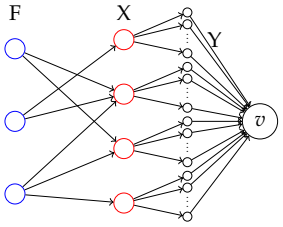
a set of additional edges $S \subseteq (V \times V) \setminus E$ such that $f(S) \geq M$ and $|S| = B$. The problem is in *NP* since it can be checked in polynomial time if a set of nodes S is such that $f(S) \geq M$ and $|S| = B$. We reduce from the Set Cover (SC) problem which is known to be *NP*-complete [25]. Consider an instance of the SC problem $I_{SC} = (X, F, k)$ defined by a collection of subsets $F = \{S_1, \ldots, S_m\}$ for a ground set of items $X = \{x_1, \ldots, x_n\}$. The problem is to decide whether there exist k subsets whose union is equal to X. We define a corresponding instance $I_{MCI} = (G, M, B)$ of MCI as follows:

- $B = k$;
- $G = (V, E)$, where $V = \{v_{x_j} | x_j \in X\} \cup \{v_{S_i} | S_i \in F\} \cup \{v\}$ and $E = \{(v_{S_i}, v_{x_j}) | x_j \in S_i\} \cup \{(v_{x_j}, v) | x_j \in X\}$; and
- $M = (n + 1 + B)^2 + (m - B)(n + B + 2)$.

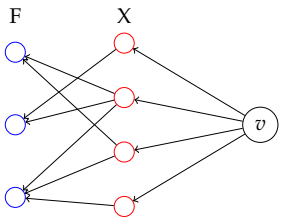
See Figure 1 (left, top) for an example. Note that G is a DAG. By Lemma 2, we can assume that any solution S of MCI contains only edges (v, v_{S_i}) for some $S_i \in F$. In fact, v is the only sink node and v_{S_i} are the only source nodes. Assume that there exists a set cover F'; then, we define a solution S to the MCI instance as $S = \{(v, v_{S_i}) | S_i \in F'\}$. It is easy to show that $f(S) = M$ and $|S| = k = B$. Indeed, all the nodes in G can reach: node v, all the nodes v_{x_j} (since F' is a set cover), and all the nodes v_{S_i} such that $S_i \in F'$. Moreover, each node v_{S_i} such that $S_i \notin F'$ can reach itself. Therefore, there are $n + B + 1$ nodes that reach $n + B + 1$ nodes and $m - B$ that reach $n + B + 2$ nodes, that is $f(S) = M$. On the other hand, assume that there exists a solution for MCI; then, S is in the form $\{(v, v_{S_i}) | S_i \in F\}$ and we define a solution for the set cover as $F' = \{S_i | (v, v_{S_i}) \in S\}$. We show that F' is a set cover. By contradiction, if we assume that F' is not a set cover and it cover only $n' < n$ elements of X, then $f(S) = (n' + B + 1)^2 + (n - n' + m - B)(n' + B + 2) < M$. Note that, in the above reduction, the graph G has a single sink node. We can prove the *NP*-hardness of the case of graphs with a single source node by using the same arguments on an instance of MCI made of the inverse graph of G, $M = (B + n + 1)(n + m + 1) + m - B$, and $B = k$ (see Figure 1 (left, bottom) for an example). □



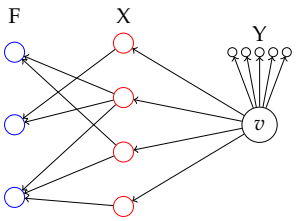
(**a**) Example of reduction from Set Cover to MCI (single sink) used in Theorem 1



(**b**) Example of reduction from Maximum Coverage to MCI (single sink) used in Theorem 2



(**c**) Example of reduction from Set Cover to MCI (single source) used in Theorem 1



(**d**) Example of reduction from Maximum Coverage to MCI (single source) used in Theorem 2

Figure 1. Examples of reductions used in Theorems 1 and 2.

Theorem 2. *MCI is NP-hard to approximate to within a factor* $1 - \frac{1}{e} + \epsilon$*, for any* $\epsilon > 0$*, even if graph contains a single sink node or a single source node.*

Proof. We give two approximation factor preserving reductions from the Maximum Coverage problem (MC), which is known to be *NP*-hard to approximate to within a factor greater than $1 - \frac{1}{e}$ [26].

The MC problem is defined as follows: given a ground set of items $X = \{x_1, \ldots, x_n\}$, a collection of subsets $F = \{S_1, \ldots, S_m\}$ of subsets of X, and an integer k, find k sets in $\mathcal{F}$ that maximise the cardinality of their union.

We first focus on the single sink problem. Given an instance of the MC problem $I_{MC} = (X, F, k)$, we define an instance of the (maximisation) MCI problem $I_{MCI} = (G, k)$ similar to the one used in Theorem 1, but where we modify the weights and add Y paths of one node between each v_{x_j} and v, where Y is an arbitrarily high number (polynomial in $n + m$).

In detail, I_{MCI} is defined as follows:

- $B = k$;
- $G = (V, E)$, where $V = \{v_{x_j} | x_j \in X\} \cup \{v_{S_i} | S_i \in F\} \cup \{v^l_{x_j} | x_j \in X \text{ and } l = 1, \ldots, Y\} \cup \{v\}$ and $E = \{(v_{S_i}, v_{x_j}) | x_j \in S_i\} \cup \{(v_{x_j}, v^l_{x_j}) | x_j \in X, l = 1, \ldots, Y\} \cup \{(v^l_{x_j}, v) | x_j \in X, l = 1, \ldots, Y\}$;
- $w(v) = 1$ and $w(u) = 0$, for each $u \in V \setminus \{v\}$; and
- $p_v = 1$ for any node $v \in V$.

See Figure 1 (right, top) for an example. We first show that there exists a solution $F' \subseteq F$ to I_{MC} that covers n' elements of X if and only if there exists a solution S to I_{MCI} such that $f(S) = n'(Y + 1) + B + 1$. Moreover, we can compute F' from S and vice versa in polynomial time. Indeed, given F', we define S as $S = \{(v, v_{S_i}) | S_i \in F'\}$. We can verify that $f(S) = n'(Y + 1) + B + 1$ and $|S| = k = B$. Indeed, only node v as a weight different from 0, and then $f(S) = R(v, S) = n'(Y + 1) + B + 1$, since v can reach the $n'(Y + 1)$ nodes v_{x_j} corresponding to the items x_j covered by F', the B nodes v_{S_i} it is connected to, and itself. On the other hand, given a solution S to I_{MCI}, by Lemma 2, we can assume that it has only edges from v to nodes v_{S_i}. Let n' be the number of nodes v_{x_j} in $R(v, S)$, then $f(S) = n'(Y + 1) + B + 1$ and $F' = \{S_i | (v, v_{S_i}) \in S\}$ covers n' elements in X.

If $OPT(I_{MC})$ and $OPT(I_{MCI})$ denote the optimum value for I_{MC} and I_{MCI}, respectively, then $OPT(I_{MCI}) \geq OPT(I_{MC})(Y + 1) + B + 1 \geq Y \cdot OPT(I_{MC})$. Moreover, given the above definition of S and F', then for any $\epsilon' > 0$ there exists a value of $Y = \mathcal{O}(poly(n + m))$ such that $f(S) \leq (n' + \epsilon')Y$.

Let us assume that there exists a polynomial-time algorithm that guarantees an α approximation for I_{MCI}, then we can compute a solution S such that $f(S) \geq \alpha OPT(I_{MCI})$. It follows that:

$$\alpha Y \cdot OPT(I_{MC}) \leq \alpha OPT(I_{MCI}) \leq f(S) \leq (n' + \epsilon')Y,$$

where n' is the number of nodes covered by the solution F' to MC obtained from S. Therefore, we obtain an algorithm that approximates the MC problem with a factor α (up to lower order terms). Since it is *NP*-hard to approximate to within a factor greater than $1 - \frac{1}{e}$ [26], then the statement follows.

Let us now focus on the single source case. Given I_{MC}, we define $I_{MCI} = (G, B)$ as follows:

- $B = k$;
- $G = (V, E)$, where $V = \{v_{x_j} | x_j \in X\} \cup \{v_{S_i} | S_i \in F\} \cup \{v\} \cup \{v^l | l = 1, \ldots Y\}$ and $E = \{(v_{x_j}, v_{S_i}) | x_j \in S_i\} \cup \{(v, v_{x_j}) | x_j \in X\} \cup \{(v, v^l) | l = 1, \ldots Y\}$;
- $w(v_{x_j}) = 1$, for each $x_j \in X$ and $w(u) = 0$, for each $u \in V \setminus \{v_{x_j} | x_j \in X\}$; and
- $p_v = 1$ for any node $v \in V$.

Y is an arbitrarily high polynomial value in $m + n$. See Figure 1 (right, bottom) for an example. We use similar arguments as above. In detail, there exists a solution $F' \subseteq F$ to I_{MC} that covers n' elements of X if and only if there exists a solution S to I_{MCI} such that $f(S) = n'(n + m - m' + Y) + n(m' + 1)$, where m' is the number of sets in F that do not cover any of the n' elements covered by F'. Moreover, we can compute F' from S and vice versa in polynomial time. Given F', we define S as $S = \{(v, v_{S_i}) | S_i \in F'\}$ and we can verify that $f(S) = \sum_{x_j \in X} p(R(v_{x_j}, S)) = \sum_{x_j \in X} |R(v_{x_j}, S)| = n'(n + m + Y + 1) + (n - n')(m' + 1) = n'(n + m - m' + Y) + n(m' + 1)$ and $|S| = B$. Given S, if n'

is the number of nodes v_{x_j} such that $v \in R(v_{x_j}, S)$, then $f(S) = n'(n + m - m' + Y) + n(m' + 1)$ and $F' = \{S_i | (v_{S_i,v}) \in S\}$ covers n' elements in X.

As above, we can show that $OPT(I_{MCI}) \geq Y \cdot OPT(I_{MC})$ and that there exists a value of $Y = \mathcal{O}(poly(n + m))$ such that $f(S) \leq (n' + \epsilon')Y$, for any $\epsilon' > 0$. Then, the statement follows by using the same arguments as above. □

4. Polynomial-Time Algorithm for Trees

In this section, we focus on the case of directed weighted rooted trees in which the root of the tree is the only source node and all the edges are directed towards the leaves. We give a polynomial-time algorithm based on dynamic programming that focuses on the special case of binary trees and requires $\mathcal{O}(|V|B^2)$ time and $\mathcal{O}(|V|B)$ space. Moreover, exploiting Lemma 2, the algorithm focuses only on edges that connect leaves to the root. We then extend our result and give an algorithm to transform any tree into a binary tree that requires $\mathcal{O}(|V|)$ time and $\mathcal{O}(|V|)$ space. Using this transformation, each solution for the transformed instance has the same value as the corresponding solution in the original instance.

4.1. Binary Trees

In the following, we introduce our dynamic-programming algorithm to solve the MCI problem in binary trees. Let us consider an instance of the MCI problem with a directed weighted binary tree $T = (V, E)$, where all the edges are directed towards the leaves, the root $r \in V$ is the only source node, and $w : V \to \mathbb{N}_{\geq 0}$, $p : V \to \mathbb{N}_{\geq 0}$. Let us denote by $\psi(v)$ (left child) and $\delta(v)$ (right child) the children of node $v \in T$; moreover, we denote as $T(v)$ the sub-tree rooted at v.

Let us first note that given a node v, and given a solution S_v that connects some leaves of $T(v)$ to r. The *gain* of solution S_v in $T(v)$ is simply the increase in weighted reachability of some of the nodes in $T(v)$ that can be written as $\sum_{u \in T(v)} w_u(p(T(u, S_v)) - p(T(u)))$. Note that, given a node $u \in T(v)$, if in the subtree of u is not present one of the edges of S_v, then the value $p(T(u, S_v)) - p(T(u))$ is equal to zero since $T(u, S_v) = T(u)$. On the other hand, if at least one edge is present, then, $T(u, S_v) = T(r)$

The algorithm that we propose computes a solution that connects b leaves of $T(v)$ to r and maximises the gain in $T(v)$ for each node $v \in V$ and for each budget $b = 0, 1, \ldots, B$. Formally, we define $g(v, b)$ as the maximum gain in $T(v)$ achievable by adding at most b edges from b leaves of $T(v)$ to node r. Note that $g(v, 1) \leq g(v, 2) \leq \ldots \leq g(v, b)$. We then now show how to compute $g(v, b)$ for each node v and for each budget $b = 0, 1, \ldots, B$ by using a dynamic programming approach. For each leaf $v \in T$ and for each $b = 1, 2, \ldots, B$, we have that $g(v, b) = w_v \cdot (p(T(r)) - p(T(v)))$, that is, the sum of profits p of the new nodes that v can reach thanks to the new edge (v, r). Moreover $g(v, 0) = 0$ for each $v \in V$. Then, the algorithm visits the nodes in T in post order. For each internal node v we compute $g(v, b)$ by using the solutions of its sub-trees, i.e., $T(\psi(v))$ and $T(\delta(v))$.

Let us assume that we have computed $g(\psi(v), b)$ and $g(\delta(v), b)$, for each $b = 0, 1, \ldots, B$. Recall that, if a solution adds an edge between any leaf of $T(v)$ and r, then the gain of node v is $w_v(p(T(r)) - p(T(v)))$ since v now reach all the nodes in T. This gain is independent of the number of edges that are added from the leaves of $T(v)$ to r. In fact, given $g(\psi(v), b_l)$ as the maximum gain for $T(\psi(v))$ and budget $b_l \in \{1, 2, \ldots, B\}$, the gain in $T(v)$ of a solution that connects b_l leaves of $T(\psi(v))$ to r is equal to $g(\psi(v), b_l) + w_v \cdot (p(T(r)) - p(T(v)))$. Similarly, the gain in $T(v)$ of the solution that connects, for some $b_r \in \{1, 2, \ldots, B\}$, b_r leaves of $T(\delta(v))$ to r is equal to $g(\delta(v), b_r) + w_v \cdot p(T(r)) - p(T(v))$.

Then, once we have decided how many edges to add in $\psi(v)$ and in $\delta(v)$ for $g(v, b)$, we increase the reachability function of the same quantity, i.e., $w_v \cdot (p(T(r)) - p(T(v)))$.

Hence, $g(v, b)$ is given by the combination of b_l and b_r such that the sum is equal to the considered budget, i.e., $b_l + b_r = b$, that maximises the sum $g(\psi(v), b_l) + g(\delta(v), b_r) + w_v \cdot (p(T(r)) - p(T(v)))$. Precisely:

$$g(v, b) = \max_{\substack{b_l, b_r \in \{0,\dots,b\} \\ b_l + b_r = b}} \{g(\psi(v), b_l) + g(\delta(v), b_r)\} + w_v \cdot (p(T(r)) - p(T(v))). \quad (1)$$

The optimal value of the problem $f(S) = g(r, B) + f(T)$, where $f(T)$ is the value of the objective function on T (i.e., when no edges have been added). The pseudo-code of the algorithm is reported in Algorithm 1.

Algorithm 1: Dynamic programming algorithm for MCI.

Input : $T = (V, E)$, B, w_v, p_v $\forall v \in V$
Output: Set S of edges

1 **for** *each node* v **do**
2 $g(v, 0) := 0$
3 $S_{(v,b)} := \emptyset$
4 **for** *each leaf* v *and budget* b, *with* $\{1, \dots, B\}$ **do**
5 $g(v, b) := w_v \cdot (p(T(r)) - p(T(v)))$
6 $S_{(v,b)} := \{(v, r)\}$
7 **for** *each node* v *in post-ordering* **do**
8 **for** $b \in 1, \dots, B$ **do**
9 $g(v, b) := \max_{\substack{b_l, b_r \in \{0,\dots,b\}, \\ b_l + b_r = b}} \{g(\psi(v), b_l) + g(\delta(v), b_r)\} + w_v \cdot (p(T(r)) - p(T(v)))$
   `/* Let` b_l `and` b_r `the budgets that maximise Line 9 */`
10 $S_{(v,b)} := S_{(\psi(v), b_l)} \cup S_{(\delta(v), b_r)}$
11 $S := S_{(r,B)}$

As a running example, let us consider a binary tree $T = (V, E)$ composed by eight nodes, as depicted in Figure 2, with $p_v = 1$ for any node $v \in V$ and $w_v = 1$ for any node $v \in V \setminus \{c, e\}$, finally $w_c = w_e = 2$. If we consider to have budget $B = 2$ and to run Algorithm 1, we first have that any leaf in $v \in T$ has $g(v, 1) = g(v, 2) = 1 \cdot (p(T(a)) - p(T(v))) = 7$. Then we start considering the rest of the nodes in post-order, thus we have: $g(d, 2) = 14 + 1 \cdot (p(T(a) - T(d)) = 19)$, $g(d, 1) = 12$, $g(e, 1) = 7 + 2 \cdot 6 = 19$. Now, note that when considering node c we have that $\max \{g(\psi(c), b_l) + g(\delta(c), b_r)\} = g(e, 1) + g(d, 1)$ for any $b_l, b_r \in \{0, \dots, b\}$, $b_l + b_r = b$, that is, we are considering a solution that had added an edge in the left sub-tree and an edge in the right sub-tree. Finally, $g(c, 2) = g(e, 1) + g(d, 1) + 2 \cdot (p(T(a)) - p(T(c))) = 35$.

Figure 2. Consider the node c with $w_c = 2$, $p_v = 1 \forall v \in V$ and $B = 2$. We have: $g(d, 2) = 19$, $g(d, 1) = 12$, and $g(e, 1) = 7 + 2(6) = 19$. Therefore $g(c, 2) = g(e, 1) + g(d, 1) + w_c \cdot (p(T(a)) - p(T(c))) = 35$.

Theorem 3. *Algorithm 1 finds an optimal solution for MCI if the graph is a binary tree.*

Proof. Let us assume by contradiction that v and b are, respectively, the first node and the first budget for which Algorithm 1 computes a non-maximum gain at Line 9 of Algorithm 1, that is $g(v,b) < g^*(v,b)$, where $g^*(v,b)$ is the maximum gain for tree $T(v)$ and budget b. Let S^* be an optimal solution that achieves $g^*(v,b)$ and let S_l^*, S_r^* be the edges in S^* that starts from leaves in $T(\psi(v))$ and $T(\delta(v))$, respectively. Let $b_l^* = |S_l^*|$ and $b_r^* = |S_r^*|$. Then, the gain of the optimal solution S^* is: $g^*(v,b) = g(\psi(v), b_l^*) + g(\delta(v), b_r^*) + w_v \cdot (p(T(r)) - p(T(v)))$.

Since by hypothesis $g(v,b)$ is the first time for which Algorithm 1 does not find the maximum gain and since the cost $(p(T(r)) - p(T(v))) \cdot w_v$ does not depend on the edges selected in the left and right sub-trees of v, this implies that at Line 9 Algorithm 1 must select

$$g(v,b) = g(\psi(v), b_l^*) + g(\delta(v), b_r^*) + (p(T(r)) - p(T(v))) \cdot w_v = g^*(v,b),$$

thus contradicting $g(v,b) < g^*(v,b)$. □

For each node v, the algorithm computes $B+1$ values of function g. Therefore, the variables $g(v,b)$ can be seen for example as a matrix of dimension $|V| \times (B+1)$. For each entry of the matrix, we need to compute the maximum among $B+1$ gains (see Equation (1)) because the number of budgets that we need to try to combine to find the solution is $\binom{B+2-1}{B} = B+1$. Thus, it follows that Algorithm 1 takes $\mathcal{O}(|V|B^2)$ time. Note that $B \in \mathcal{O}(|V|)$ because we limit the new edges to be of the form leaf-root. Moreover, the space complexity is $O(|V|B)$.

4.2. General Trees

In the following, we present an algorithm that requires $\mathcal{O}(|V|)$ time and space to transform any generic rooted tree $T = (V, E)$ into an equivalent binary tree $T' = (V \cup U, E')$, following a tree transformation proposed in [27] by adding at most $|V| - 3$ *dummy* node.

Given a generic rooted tree $T = (V, E)$, let us transform it into a rooted binary tree $T' = (V \cup U, E')$ with weights w', p' by adding a set of *dummy* nodes U as follows:

1. Let the root r of T be the root of T'.
2. For each non-leaf node v, let $v_1, v_2, \ldots v_l$ be the children of v:
   (a) Add edge (v, v_1) to E'.
   (b) If $l = 2$, add (v, v_2) to E'.
   (c) If $l > 2$, add $l-2$ dummy nodes $u_{v_2}, u_{v_3}, \ldots, u_{v_{l-2}}, u_{v_{l-1}}$.
   (d) Add edge (v, u_{v_2}) and edges $(u_{v_i}, u_{v_{i+1}})$ to E', for each $2 \leq i \leq l-2$.
   (e) Add edge (u_{v_i}, v_i) to E', for each $2 \leq i \leq l-1$.
   (f) If $l > 2$, add edge $(u_{i_{l-1}}, v_l)$ to E'.
3. If $v \in V$, then $w'_v = w_v$, otherwise $w'_v = 0$ and $p'_v = p_v$, otherwise $p'_v = 0$.

See Figure 3 for an example of the transformation.

Note that $p(R(v,T')) = p(R(v,T))$ for any node v in T due to the fact that the added dummy nodes have $p'_v = 0$; moreover, dummy nodes do not increase the objective function because they have the weight set to zero, i.e., any dummy node v will have $w'_v \cdot p(R(v,T')) = 0$.

For each node $v \in T'$ and solution S to MCI in T', let $f'(S) = \sum_{v \in V} w_v p(T(v,S))$. It is easy to see that by applying Algorithm 1 to T' we obtain an optimal solution with respect to f'. Moreover, for each solution S to T', $f'(S) = f(S)$. Note that a solution S for T' that connects leaves of T' to its root is a feasible solution also for T since T and T' have the same root and leaves.

(**a**) General tree $T = (V, E)$ (**b**) Transformed binary tree $T' = (V \cup U, E')$

Figure 3. Example of transformation from general tree to binary tree.

5. Polynomial-Time Algorithm for DAG with a Single Source or a Single Sink

In this section, we focus on the case of weighted DAGs in which we have a single source node or a single sink node. We first describe our greedy algorithm to approximate MCI on DAGs with a single source. Then, we show how to modify the algorithm for the case of DAGs with a single sink.

In the case of a single source, by using the property of Lemma 2, we restrict our choices to the edges that connect sinks nodes to the source. Let us denote S as the set of edges added to G. With a little abuse of notation, we also use S to denote the set of sinks from which the edges in S start. Note that no information is lost in this way, since we have a DAG with a single source, at most one edge is added for each sink. In fact, if a second edge is added to a sink, this edge would not bring any increase to the objective function.

The Greedy algorithm for MCI on DAGs with a single source (see Algorithm 2), starts with an empty solution $S = \emptyset$ and repeatedly adds to S the edge e' that maximises the function $f(S \cup \{e'\})$. The edge e' is chosen from the set E' of edges (u, s), where u is a sink in V, not already inserted in S, and s is the single source in V (see Lines 2 and 4).

To implement the Greedy Algorithm with a single source, some preprocessing is required. First, for each node $v \in V$, we perform a DFS visit on G to compute $R(v)$ and $p(R(v))$. We store $p(R(v, S))$ in a vector ρ of size $|V|$. Every time a new edge is added to the solution S, each entry of ρ is updated in constant time because for each node v, $p(R(v, S))$ is either equal to $p(R(v))$ or $p(R(s))$, as we explain below. To compute the gain of adding the edge $e = (u, s)$, we need to find all the nodes $R^T(u, S)$ that reach u in $G(S)$. $R^T(u, S)$ is computed by performing a DFS visit starting from u on the reverse graph $G^T(S)$ of $G(S)$. Note that the reverse graph G^T of G is initially computed in a preprocessing phase in $\mathcal{O}(|V| + |E|)$ time, and after every new edge is added to S, $G^T(S)$ is updated in constant time. Finally, to compute $f(S \cup \{e\})$ for $e = (u, s)$, observe that $f(S \cup \{e\}) = f(S) + \sum_{z \in R^T(u,S)} w_z(p(R(s)) - p(R(z, S)))$. After selecting the edge $e' = (u, s)$ that maximise $f(S \cup \{e'\})$, we update S and we set $p(R(z, S)) = p(R(s))$ for each node $z \in R^T(u, S)$ in vector ρ because z reaches s traversing the edge $e' = (u, s)$ and inherits the reachability of $R(s)$.

The Greedy algorithm with a single source requires $\mathcal{O}(B|V||E|)$ time. Namely, for each edge $e = (u, s) \in E'$, to compute $f(S \cup \{e\})$ it is required $\mathcal{O}(|E|)$ time to compute $R^T(u, S)$ on G^T, and $\mathcal{O}(|V|)$ time to compute $\sum_{z \in R^T(u,S)} w_z(p(R(s)) - p(R(z, S)))$. Since there are $\mathcal{O}(|V|)$ sinks, the computation of the maximum value at Line 4 costs $\mathcal{O}(|V||E|)$ time. Selected e', $\mathcal{O}(|V|)$ time is spent to update ρ. Since at most B edges are added, the Greedy algorithm requires $\mathcal{O}(B|V||E|)$.

In the following, we describe the algorithm to compute a solution in the case of DAGs with a single sink. The main differences compared to the previous case mainly concern the preprocessing phase of the algorithm. The greedy algorithm remains the same except that we change the set E' from which the edges are chosen in the following way: we substitute E' on Line 2 with the set of edges (d, v), where v is a source in V and d is the only sink in V (see Line 2) by Lemma 2.

It is worth noting that, in this case, differently from before, the value of $R^T(d, S)$ is always equal to V by definition since all the nodes reach the sink, on the other hand, the reachability of a node v does not assume only the values V or $R(v)$. This implies that we it is not required to perform any DFS visit of G^T to compute $f(S \cup \{(d, v)\})$, in fact, the vicinity of any other node depends on the set of added

edges and has to be recomputed each time by performing a DFS visit on the augmented graph. Hence, $f(S \cup \{(d,v)\}) = f(S) + \sum_{z \in V} w_z \cdot p(R(v,S) \setminus R(d,S))$. Since for computing $f(S \cup \{(d,v)\})$, we must compute $|V|$ DFS visits, the overall cost of the Greedy algorithm with a single sink increases by a factor of $|V|$ with respect to the case of DAG with a single source, thus becoming $\mathcal{O}(B|V|^2|E|)$. Namely, for each edge $e = (u,r) \in E'$, to compute $f(S \cup \{e\})$ it is required $\mathcal{O}(|V||E|)$ time to compute $|V|$ DFS visits in $G(S)$, one for each node in V, and $\mathcal{O}(|V|)$ time to update R and to compute $\sum_{z \in R^T(u,S)} w_z(|V| - |R(z,S)|)$. Since there are $\mathcal{O}(|V|)$ sinks, the computation of the maximum value at Line 4 costs $\mathcal{O}(|V|^2|E|)$ time. Then, since at most B edges are added, the Greedy algorithm requires $\mathcal{O}(B|V|^2|E|)$.

Observe that Algorithm 2 that we have just described in the case of a single source can also be used on trees in place of Algorithm 1. However, the complexity of the Greedy algorithm will be $\mathcal{O}(|V|^2 B)$ that is greater than the complexity of Algorithm 1, which is $\mathcal{O}(|V|B^2)$.

To give a lower bound on the approximation ratio of Algorithm 2, we show that the objective function $f(S)$ is *monotone and submodular*. Recall that, for a ground set N, a function $z : 2^N \to \mathbb{R}$ is said to be submodular, if it satisfies the following property of diminishing marginal returns: for any pair of sets $S \subseteq T \subseteq N$ and for any element $e \in N \setminus T$, $z(S \cup \{e\}) - z(S) \geq z(T \cup \{e\}) - z(T)$. This allows us to apply the result by Nemhauser et al. [28]: Given a finite set N, an integer k', and a real-valued function z defined on the set of subsets of N, the problem of finding a set $S \subseteq N$ such that $|S| \leq k'$ and $z(S)$ is maximum can be $1 - \frac{1}{e}$ approximated by starting with the empty set, and repeatedly adding the element that gives the maximal marginal gain, if z is monotone and submodular.

Recall that $f(S) = \sum_{v \in V} w_v p(R(v,S))$ and $w_v, p_v \in \mathbb{N}_{\geq 0}$. To prove that $f(S)$ is a monotone increasing and submodular function, we just need to show that $p(R(v,S))$ is monotone increasing and submodular, for each node $v \in V$ and solution S. This is due because a non-negative linear combination of monotone submodular functions is also monotone and submodular.

Algorithm 2: Greedy Algorithm for single source.

Input : DAG $G = (V,E)$, B, $w_v, p_v \ \forall v \in V$
Output: Set S of edges

1 $S := \emptyset$
2 $E' := \{e = (u,s) | u$ is sink and s is the only source$\}$
3 **while** $|S| \leq B$ **do**
4 $\quad e' := \arg\max_{e \in E' \setminus S} f(S \cup \{e\})$
5 $\quad S := S \cup \{e'\}$

Theorem 4. *Function $f(S)$ is monotone and submodular with respect to any feasible solution for MCI on DAGs with a single source.*

Proof. To prove that $f(S)$ is monotone, we prove that for each $v \in V$, $S \subseteq E'$, and $e = (t',s) \in E' \setminus S$, we have $p(R(v, S \cup \{e\})) \geq p(R(v,S))$.

We first notice that for each node $v \in V$ and solution S, if there exists an edge $(t,s) \in S$ such that $t \in R(v)$, then $p(R(v,S)) = p(R(s))$; otherwise, $p(R(v,S)) = p(R(v))$. The same holds for $p(R(v, S \cup \{e\}))$.

We analyse the following cases recalling that $e = (t',s)$:

- If there exists an edge $(t,s) \in S$ such that $t \in R(v)$, then $p(R(v, S \cup \{e\})) = p(R(v,S)) = p(R(s))$.
- Otherwise:
  - If $t' \in R(v)$, then $p(R(v, S \cup \{e\})) = p(R(s))$ and $p(R(v,S)) = p(R(v))$.
  - If $t' \notin R(v)$, then $p(R(v, S \cup \{e\})) = p(R(v,S)) = p(R(v))$.

It follows that $p(R(v, S \cup \{e\})) \geq p(R(v,S))$.

To prove that $f(S)$ is submodular, we prove that, for any node $v \in V$, any two solutions S, T of MCI such that $S \subseteq T$, and any edge $e = (t', s) \notin T$, where t' is a sink node, it holds:

$$p(R(v, S \cup \{e\})) - p(R(v, S)) \geq p(R(v, T \cup \{e\})) - p(R(v, T)). \tag{2}$$

We analyse the following cases:

- If there exists an edge $(t, s) \in S$ such that $t \in R(v)$, then, $p(R(v, S \cup \{e\})) = p(R(v, S)) = p(R(v, T \cup \{e\})) = p(R(v, T)) = p(R(s))$.
- Otherwise,
  - If there exists $(t'', s) \in T$ such that $t'' \in R(v)$, then $p(R(v, S \cup \{e\})) - p(R(v, S)) \geq 0 = p(R(v, T \cup \{e\})) - p(R(v, T))$ because $p(R(v, T \cup \{e\})) = p(R(v, T)) = p(R(s))$.
  - If for each $(t'', s) \in T$, $t'' \notin R(v)$, then $p(R(v, S \cup \{e\})) = p(R(v, T \cup \{e\}))$ and $p(R(v, S)) = p(R(v, T))$.

In all the cases the inequality in Equation (2) holds. □

Theorem 5. *Function $f(S)$ is monotone and submodular with respect to any feasible solution for MCI on DAGs with a single sink.*

Proof. To prove that $f(S)$ is monotone, we show that for each $v \in V$, $S \subseteq E'$, and $e = (d, s') \in E' \setminus S$, we have $p(R(v, S \cup \{e\})) \geq p(R(v, S))$. We observe that

$$\begin{aligned} R(v, S) &= R(v) \cup \bigcup_{(d, s_i) \in S} R(s_i) \\ R(v, S \cup \{e\}) &= R(v) \cup \bigcup_{(d, s_i) \in S} R(s_i) \cup R(s') = R(v, S) \cup R(s') \end{aligned} \tag{3}$$

Thus, $p(R(v, S)) \leq \sum_{u \in R(v,S) \cup R(s')} p_u$.

To prove that $f(S)$ is submodular, we prove that for any node $v \in V$, any two solutions S, T of MCI such that $S \subseteq T$, and any edge $e = (d, s') \notin T$, where s' is a source node:

$$p(R(v, S \cup \{e\})) - p(R(v, S)) \geq p(R(v, T \cup \{e\})) - p(R(v, T)). \tag{4}$$

We first make the following observations based on Equation (3):

- $R(v, S \cup \{e\}) = R(v, S) \cup R(s') = R(v, S) \cup (R(s') \setminus R(v, S))$ and
- $R(v, T \cup \{e\}) = R(v, T) \cup R(s') = R(v, T) \cup (R(s') \setminus R(v, T))$

Then, $p(R(v, S \cup \{e\})) - p(R(v, S)) = p(R(s') \setminus R(v, S))$ and $p(R(v, T \cup \{e\})) - p(R(v, T)) = p(R(s') \setminus R(v, T))$.

The inequality in Equation (4) follows by observing that $R(v, S) \subseteq R(v, T)$. □

Corollary 1. *Algorithm 2 provides a $\left(1 - \frac{1}{e}\right)$-approximation for the MCI problem either on DAG with a single source or with a single sink.*

6. Polynomial-Time Algorithm for DAG with Two Sources

In this section, we take a first step in the direction of studying the MCI problem in general DAGs, i.e., multiple sources and multiple sinks, by tackling the special case of weighted DAGs with two source nodes. In the following, we describe an approach that can be easily extended to the case of DAGs with a constant number of sources. However, this approach can be computationally heavy as the number of such sources increases.

We first prove an important property that we exploit to provide a polynomial-time algorithm for this case. The idea is that, if we add an initial edge to the solution, then any other added edge is toward the opposite source node. Using this property, we are able to reduce the number of possible solutions that we have to consider. Furthermore, this allows us to use the greedy algorithm that we proposed in the previous section to solve the problem.

In this section, we improve the notation by defining s_1, s_2 the two sources and by L_1, L_2 the sink nodes reachable from source s_1 and s_2, respectively. Note that we allow $L_1 \cap L_2 \neq \emptyset$, i.e., the two sources may share sink nodes.

Lemma 3. *Let S be a solution to the MCI problem in a graph $G = (V, E)$ with two sources s_1, s_2; then, there exists a solution S' such that $|S'| = |S|$ with $f(S') \geq f(S)$, and S' contains at most one edge e directed towards source s_1 (s_2, respectively), while all the other edges are directed towards s_2 (s_1, respectively).*

Proof. Let us first consider the following case: there exists in S at least one edge connecting a sink in L_2 with the source s_1, i.e., edge $(v_i, s_1) \in S$ with $v_i \in L_2$. Note that this case is equivalent to connecting a node in L_1 to s_2. Now, for any other edge $e_1 = (v_j, s_1) \in S$, we prove that we can change such edge to connect to source s_2, i.e., $e_2 = (v_j, s_2)$, without decreasing the objective function. In fact, note that, for any node $z \in R^T(v_j, S)$ and solution S, we have that $p(R(z, S))$ can be either equal to $p(R(V))$, if there exists a path that connect such node to source s_2, or equal to $R(s_1)$ by using edge e_1. Instead, by using edge e_2, we have that any node $z \in R^T(v_j, S)$ is able to reach any node in the graph, thus having profit $p(R(V))$. Therefore, we have that $f(S) \leq f((S \cup e_2) \setminus e_1)$ since any node that is able to reach sink v_j in solution $(S \cup e_2) \setminus e_1$ actually reaches all the nodes in the graph.

On the other hand, it must be that all edges in S are of the kind (v_i, s_1) with $v_i \in L_1$ or (v_i, s_2) with $v_i \in L_2$, i.e., all the edges from L_1 are toward source s_1 (s_2, respectively). Let us divide solution S in two subsets: S_1 containing all the edges of kind (v_i, s_1) with $v_i \in L_1$, and S_2, respectively, with edges of kind (v_i, s_2) with $v_i \in L_2$. Then, by changing one edge from S_1 in (v_j, s_2) and all the edges in S_2 toward s_1 we have increased the objective function, since now all the nodes in a path to one of the sources are able to reach any node in the graph. □

The lemma above shows us that we can build a solution for the MCI problem by selecting one initial edge and then we can use the greedy algorithm to choose the rest of the edges. In fact, after we have added the initial edge, we can exploit the algorithm given in Section 5, since the DAG now has only one source node and thus the submodularity holds. The underlying idea for the algorithm is to create $2 \cdot |L_1 \cup L_2|$ different instances by enumerating all possible combination of one initial edge, then, we run the greedy algorithm in each of this instances and we choose the best solution among them. Note that this set of instances is composed by four different kinds of edges: The edges from L_1 to s_2 (and from L_2 to s_1, respectively) and the edges from $L_1 \setminus L_2$ to s_1 (and from $L_2 \setminus L_1$ to s_2, respectively). In both cases, the algorithm considers s_1 (s_2, respectively) as the only source node. Note that, in both cases, we are considering to optimise an MCI instance with only one source node and thus the objective function is monotone and submodular, as proved in Section 5.

Moreover, since our algorithm tries all possible initial edges and then picks the best solution, it must be that the initial edge is also in the optimal solution, otherwise there would another instance with a greater value. Thus, the greedy algorithm gives us the same approximation ratio as before, i.e., $1 - 1/e$, and requires $\mathcal{O}(B|V|^2|E|)$ time, in fact note that all the possible instances that the algorithm tries are order of number of nodes, i.e., $|L_1 \cup L_2| \leq |V|$.

7. Conclusions and Future Works

In this work, we study the problem of improving the reachability of a graph. In particular, we introduce a new problem called the Maximum Connectivity Improvement (MCI) problem: given a directed graph and an integer B, the problem asks to add a set of edges of size at most B in order to maximise the overall weighted number of reachable nodes.

We firstly provide a reduction from the well-known Set Cover problem to prove that the MCI problem is *NP*-complete. This result holds even if weights and profits are unitary and the graph contains a single sink node or a single source node. Moreover, via a reduction from the Maximum Coverage problem, we prove that the MCI problem is *NP*-hard to approximate to within a factor greater than $1 - 1/e$ even in DAGs with a single source or a single sink.

We then propose a dynamic programming algorithm for the MCI problem on trees with a single source that produces optimal solutions in polynomial time. Then, we study the case of DAGs with a single source (or a single sink); in this case, we propose two $\left(1 - \frac{1}{e}\right)$-approximation algorithms that run in polynomial-time by showing that the objective function is monotone and submodular. Finally, we extend the latter algorithm for DAGs with one source to DAGs with two sources while keeping the same approximation guarantee.

As future works, we plan to extend our approach to general DAGs, i.e., with multiple sources and multiple sinks. Another possible extension is to solve the MCI problem by considering the budgeted version of the problem in which each edge can be added at a different budget cost. The goal is then to find a minimum cost set of pair (sink, source) to which add the edges.

Author Contributions: All authors have equally contributed to this work. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This work has been partially supported by the Italian MIUR PRIN 2017 Project ALGADIMAR "Algorithms, Games, and Digital Markets."

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cordasco, G.; Gargano, L.; Lafond, M.; Narayanan, L.; Rescigno, A.A.; Vaccaro, U.; Wu, K. Whom to befriend to influence people. *arXiv* **2016**, arXiv:1611.08687.
2. D'Angelo, G.; Severini, L.; Velaj, Y. Selecting nodes and buying links to maximize the information diffusion in a network. In Proceedings of the 42st International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, Aalborg, Denmark, 21–25 August 2017; Volume 83, pp. 75:1–75:14.
3. Corò, F.; D'Angelo, G.; Velaj, Y. Recommending Links to Maximize the Influence in Social Networks. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, 10–16 August 2019; pp. 2195–2201. doi:10.24963/ijcai.2019/304. [CrossRef]
4. Olsen, M.; Viglas, A. On the approximability of the link building problem. *Theor. Comput. Sci.* **2014**, *518*, 96–116. [CrossRef]
5. Papagelis, M. Refining social graph connectivity via shortcut edge addition. *ACM Trans. Knowl. Discov. Data (TKDD)* **2015**, *10*, 12. [CrossRef]
6. Jung, J.; Seo, D.; Lee, J. Counter-Based Broadcast Scheme Considering Reachability, Network Density, and Energy Efficiency for Wireless Sensor Networks. *Sensors* **2018**, *18*, 120. doi:10.3390/s18010120. [CrossRef] [PubMed]
7. Corò, F.; D'Angelo, G.; Velaj, Y. Recommending Links to Control Elections via Social Influence. *Algorithms* **2019**, *12*, 207. doi:10.3390/a12100207. [CrossRef]
8. Castiglioni, M.; Ferraioli, D.; Gatti, N. Election Control in Social Networks via Edge Addition or Removal. *arXiv* **2019**, arXiv:1911.06198.
9. Eswaran, K.P.; Tarjan, R.E. Augmentation problems. *SIAM J. Comput.* **1976**, *5*, 653–665. [CrossRef]
10. Demaine, E.D.; Zadimoghaddam, M. Minimizing the Diameter of a Network Using Shortcut Edges. In *Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6139, pp. 420–431.
11. Bilò, D.; Gualà, L.; Proietti, G. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theor. Comput. Sci.* **2012**, *417*, 12–22. doi:10.1016/j.tcs.2011.05.014. [CrossRef]
12. Ianni, M.D.; Gualà, L.; Rossi, G. Reducing the diameter of a unit disk graph via node addition. *Inf. Process. Lett.* **2015**, *115*, 845–850. doi:10.1016/j.ipl.2015.06.015. [CrossRef]

13. Bilò, D. Almost Optimal Algorithms for Diameter-Optimally Augmenting Trees. In Proceedings of the 29th International Symposium on Algorithms and Computation, ISAAC 2018, Jiaoxi, Yilan, Taiwan, 16–19 December 2018; pp. 40:1–40:13. doi:10.4230/LIPIcs.ISAAC.2018.40. [CrossRef]
14. Avrachenkov, K.; Litvak, N. The Effect of New Links on Google Pagerank. *Stoc. Models* **2006**, *22*, 319–331. [CrossRef]
15. Meyerson, A.; Tagiku, B. Minimizing Average Shortest Path Distances via Shortcut Edge Addition. In *Proceedings of the 13th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5687, pp. 272–285.
16. Crescenzi, P.; D'Angelo, G.; Severini, L.; Velaj, Y. Greedily improving our own centrality in a network. In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA 2015)*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9125, pp. 43–55.
17. Crescenzi, P.; D'Angelo, G.; Severini, L.; Velaj, Y. Greedily Improving Our Own Closeness Centrality in a Network. *TKDD* **2016**, *11*, 9:1–9:32. [CrossRef]
18. Ishakian, V.; Erdös, D.; Terzi, E.; Bestavros, A. A Framework for the Evaluation and Management of Network Centrality. In Proceedings of the 12th SIAM International Conference on Data Mining (SDM), Anaheim, CA, USA, 26–28 April 2012; pp. 427–438.
19. Kapron, B.M.; King, V.; Mountjoy, B. Dynamic graph connectivity in polylogarithmic worst case time. In Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, LA, USA, 6–8 January 2013; pp. 1131–1142. doi:10.1137/1.9781611973105.81. [CrossRef]
20. Nanongkai, D.; Saranurak, T.; Wulff-Nilsen, C. Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time. In Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, 15–17 October 2017; pp. 950–961. doi:10.1109/FOCS.2017.92. [CrossRef]
21. Demetrescu, C.; Italiano, G.F. Mantaining Dynamic Matrices for Fully Dynamic Transitive Closure. *Algorithmica* **2008**, *51*, 387–427. doi:10.1007/s00453-007-9051-4. [CrossRef]
22. Roditty, L.; Zwick, U. A Fully Dynamic Reachability Algorithm for Directed Graphs with an Almost Linear Update Time. *SIAM J. Comput.* **2016**, *45*, 712–733. doi:10.1137/13093618X. [CrossRef]
23. Corò, F.; D'Angelo, G.; Pinotti, C.M. On the Maximum Connectivity Improvement Problem. In Proceedings of the Algorithms for Sensor Systems—14th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2018, Helsinki, Finland, 23–24 August 2018; pp. 47–61. doi:10.1007/978-3-030-14094-6_4. [CrossRef]
24. Tarjan, R. Depth-first search and linear graph algorithms. *SIAM J. Comput.* **1972**, *1*, 146–160. [CrossRef]
25. Garey, M.R.; Johnson, D.S. *Computers and Intractability, A Guide to the Theory of NP-Completeness*; W.H. Freeman and Company: New York, NY, USA, 1979.
26. Feige, U. A Threshold of Ln N for Approximating Set Cover. *J. ACM* **1998**, *45*. [CrossRef]
27. Tamir, A. An O(pn^2). algorithm for the p-median and related problems on tree graphs. *Oper. Res. Lett.* **1996**, *19*, 59–64. [CrossRef]
28. Nemhauser, G.L.; Wolsey, L.A.; Fisher, M.L. An analysis of approximations for maximizing submodular set functions—I. *Math. Program.* **1978**, *14*, 265–294. [CrossRef]