The Maximum Labeled Path Problem*

Basile Couëtoux, Elie Nakache, and Yann Vaxès

Aix-Marseille Université, CNRS, LIF UMR 7279, 13288, Marseille, France.
{basile.couetoux,elie.nakache,yann.vaxes}@univ-amu.fr

Abstract. In this paper, we study the approximability of the Maximum Labeled Path problem: given a vertex-labeled directed acyclic graph D, find a path in D that collects a maximum number of distinct labels. For any $\epsilon > 0$, we provide a polynomial time approximation algorithm that computes a solution of value at least $OPT^{1-\epsilon}$ and a self-reduction showing that any constant ratio approximation algorithm for this problem can be converted into a PTAS. This last result, combined with the APX-hardness of the problem, shows that the problem cannot be approximated within any constant ratio unless $P = NP$.

1 Introduction

Optimization network design problems over labeled graphs have been widely studied in the literature [2–8, 10, 11]. Since these problems are usually NP-hard, they have been mainly investigated toward the goal of finding efficiently approximate solutions. Most of these studies consider edge-labels that represent kinds of connections and the objective is often to minimize the number of different kinds of connections used. Our motivation is different, we consider vertex-labels that represent membership to different components. Our goal is then to maximize the number of components visited by a path in a directed graph. More precisely, our problem is defined on a directed acyclic graph with labels on the vertices and the objective is to find a path visiting a maximum number of distinct labels. We call this problem Max-Labeled-Path. Actually, the vertex-labeled and edge-labeled versions of this problem are equivalent but the vertex-labeled version is closer to our initial motivation. To our knowledge, there is no prior work on this simple and natural problem. A related problem is the Min LP $s-t$ problem that asks to find a path between two vertices s and t minimizing the number of different labels in this path. In [7] Hassin et al achieve a $\sqrt{n}$ ratio for this problem and show that it is hard to approximate within $O(\log n)$. We used a similar approach for our hardness result but the maximization version requires a much more precise analysis. In contrast with the results of [7], we only consider the case of directed acyclic graph. This restriction is essential for our algorithms and makes our hardness results stronger. As illustrated above, this setting captures many situations in which a notion of time has to be taken into account.

One typical application of this problem arises in the following situation. An agent can move along the edges of a network and perform a set of tasks. He knows how long it takes to move between every pair of nodes, the time needed to execute each task and the subset of nodes where each task can be performed. The goal of the agent is to perform a maximum number of (distinct) tasks during a given amount of time. Using a discretization of the time, this problem can be modeled as a Max-Labeled-Path problem. Each vertex u_t of the DAG D corresponds to a possible location u of the agent at a given time t. There is an arc in D between a vertex u_t and a vertex $v_{t'}$ if the agent located in u at time t can perform the task of u and reach the node v before time t'. The label of each vertex corresponds to the task that can be performed in this

* An extended abstract of this paper appeared in the proceedings of the 40th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'14.

vertex (if several tasks can be performed in the same node then we can simply split this node into several copies and join them by an edge along which the agent can move without delay). In order to perform a maximum number of tasks the agent has to compute a path in D collecting a maximum number of distinct labels.

1.1 Contributions

In this paper, we report both positive and negative results about the Max-Labeled-Path. Namely, we prove that this problem does not admit a constant factor approximation algorithm unless $P = NP$ and we describe an algorithm that, for any fixed $\epsilon > 0$, returns in polynomial time a solution of value at least $OPT^{1-\epsilon}$ where OPT is the value of an optimal solution. In Section 2, the hardness proof starts with a reduction from Max-3SAT preserving the approximation and therefore proving that Max-Labeled-Path is APX-hard. In Section 3, a polynomial self-reduction shows that finding a solution on a more complex graph enables us to find a solution with a better ratio on the initial graph. This, combined with the APX-hardness of the problem, shows that the problem cannot be approximated within a constant ratio unless $P = NP$. In Section 4, we describe an $\sqrt{OPT}$-approximation algorithm for Max-Labeled-Path. This algorithm requires a specific preprocessing and an inductive analysis that uses the poset structure of the problem. Using this algorithm as a starting point, a more elaborate $OPT^{1-\epsilon}$-approximation is presented in Section 5. By using a scaling technique, this algorithm can be adapted to handle with the same performance guarantee a label weighted version of Max-Labeled-Path. Finally, the problem of collecting all the labels with a minimum number of paths is also addressed and, for any $\epsilon > 0$, a polynomial time algorithm that computes a solution within a factor $\frac{|\mathcal{L}|^\epsilon}{\epsilon}$ of the optimum is provided in Section 6, where $\mathcal{L}$ is the set of labels.

1.2 Preliminaries

A vertex-labeled Directed Acyclic Graph $D = (V, A)$ is a DAG whose vertices are labeled by a function $l : V \to \mathcal{L}$, with $\mathcal{L}$ any set representing the sets of labels. For each vertex $u \in V$, we denote by $\lambda(u)$ and call the *level* of u, the maximum number of vertices in a path having u as end-vertex. The *ith level set* L_i of D consists of all vertices $u \in V$ such that $\lambda(u) = i$. The vertices of L_1, i.e. having no ingoing arcs, are called the *sources* of D. The vertices having no outgoing arcs are called the sinks. Let k be the largest integer such that $L_k \neq \emptyset$. L_k is a subset of the sinks. Let P be a (directed) path in D. P is maximal by inclusion if and only if it connects a source to a sink. The set of labels *collected* by P is the set $\mathcal{L}^P = \{l(u) : u \in P\}$ of labels of vertices in P. More generally, the labels collected by a subset of vertices $S \in V$ is the set $\mathcal{L}^S = \{l(u) : u \in S\}$ of labels of vertices in S. Given a vertex-labeled DAG D, the problem Max-Labeled-Path consists in finding a path P in D maximizing the number of distinct labels collected by P, i.e. maximizing $|\mathcal{L}^P|$. Any solution can be extended into a maximal path without decreasing its value, therefore we only consider solutions that connect a source to a sink. In this paper, we consider only maximization problems. Let D be an instance of a maximization problem, we denote by $OPT(D)$ its optimum. We say that an algorithm *achieves a constant performance ratio* α, if for every instance D, it returns a solution of value at least $\alpha\, OPT(D)$. We say that an algorithm is an $f(OPT)$-approximation if, for every instance D, it returns a solution of value at least $f(OPT)$.

2 Maximum Labeled Path is APX-hard

In this section, we describe a reduction from Max-3SAT : Given a formula in conjunctive normal form with at most 3 variables per clause, find an assignment that satisfies the largest number of

clauses. This reduction establishes that MAX-LABELED-PATH is APX-hard even when restricted to instances satisfying the following conditions:

(C1) All maximal (by inclusion) paths of D contain the same number k of vertices.
(C2) D contains a path that collects all the labels, i.e. $OPT(D) = |\mathcal{L}|$.
(C3) D contains a path that collects each label exactly once, i.e. $OPT(D) = k = |\mathcal{L}|$.
(C4) $OPT(D) = k = |\mathcal{L}|$ is a power of two.

Note that (C4) is stronger than (C3) which is stronger than (C2). Applying our initial reduction to satisfiable instances of MAX-3SAT, we produce instances of MAX-LABELED-PATH satisfying conditions (C1) with $k \leq 3|\mathcal{L}|$ and (C2) and prove Theorem 2. Then, we proceed in two steps: first we establish the APX-hardness for instances satisfying conditions (C1) and (C3) in Theorem 3 and then the APX-hardness for instances satisfying conditions (C1) and (C4) in Theorem 4. In the next section we use a self-reduction of MAX-LABELED-PATH to prove that MAX-LABELED-PATH does not belong to APX. This self-reduction is valid only for instances satisfying conditions (C1) and (C4).

Theorem 1. *(Håstad [9]) Assuming $P \neq NP$, no polynomial-time algorithm can achieve a performance ratio exceeding $\frac{7}{8}$ for* MAX-3SAT *even when restricted to satisfiable instances of the problem.*

Theorem 2. *Assuming $P \neq NP$, no polynomial-time algorithm can achieve a performance ratio exceeding $\frac{7}{8}$ for* MAX-LABELED-PATH *even when restricted to instances satisfying conditions (C1) with $k \leq 3|\mathcal{L}|$ and (C2).*

Before proving Theorem 2, we prove the following lemma showing that (C1) is not a strong requirement in the sense that each instance of MAX-LABELED-PATH can be converted into an equivalent instance satisfying (C1).

Lemma 1. *Given an instance D of* MAX-LABELED-PATH*, it is possible to construct an instance D' satisfying condition (C1) and such that there exists a mapping between the set of maximal paths in D and the set of maximal paths in D' preserving the number of labels collected.*

Proof. We obtain D' from D by first splitting each arc uv of D such that $\lambda(v) > \lambda(u) + 1$ into a path consisting of $\lambda(v) - \lambda(u)$ arcs and assigning to the internal vertices of this path the label of u. Then, we consider every sink w that belongs to a level set L_i distinct from the last level set L_k and add a path of length $k - i$ starting in w whose vertices receive the label $l(w)$ in order to create new sinks with level k. Clearly, the level of a vertex u in D is the same as its level in D'. Moreover, any arc in D' connects two vertices lying in two consecutive level sets and all sinks belong to the last level set L_k. Therefore, all maximal paths have the same number k of vertices, i.e. D' satisfies (C1) (see Fig. 1). Moreover, given a path P' in D' one can compute in polynomial time a path P in D collecting the same number of labels and vice versa. □

Proof (of Theorem 2). Given an instance F of MAX-3SAT, we define an instance $D_F = (V, A)$ of MAX-LABELED-PATH as follows. Let $\{w^1, w^2, ..., w^q\}$ be the set of variables of F. For all $j \in \{1, ..., q\}$, we denote by $|w^j|$ the number of occurrences of the literal w^j and by $|\neg w^j|$ the number of occurrences of its negation. We create $|w^j| + |\neg w^j|$ vertices and call them $w^j_1, w^j_2, ..., w^j_{|w^j|}$ and $\neg w^j_1, \neg w^j_2, ..., \neg w^j_{|\neg w^j|}$. We connect in a directed path $P(w^j)$ the vertices which represent the literal w^j, i.e. we create an arc (w^j_i, w^j_{i+1}) for all $i \in \{1, \ldots, |w^j| - 1\}$. In the same way, we connect in a directed path $P(\neg w^j)$ the vertices representing $\neg w^j$. For all $j \in \{1, ..., q-1\}$, we connect by an arc the last vertices of $P(w^j)$ and $P(\neg w^j)$ to the first vertices of $P(w^{j+1})$ and $P(\neg w^{j+1})$. Let

us define the labeling function $l : V \to \mathcal{L} := \{1, \ldots, m\}$ where m is the cardinality of the set of clauses $\{C_1, C_2, \ldots, C_m\}$ of F. There is a one to one correspondence between the occurrences of the literals in the clauses and the vertices of D_F. A vertex u receives the label j if u corresponds to an occurrence of a literal in the clause C_j (see Fig. 1).

Applying the reduction to a satisfiable instance F of Max-3SAT, we obtain an instance D_F of Max-Labeled-Path that contains a path collecting all the labels, i.e. that satisfies condition (C2). Moreover, since each clause contains at most three literals, the number k of vertices in a maximal path of D_F is at most thrice the number m of labels, i.e. $k \leq 3m$. In the resulting graph D_F, each maximal path P is a path from a vertex in $\{w_1^1, \neg w_1^1\}$ to a vertex in $\{w^q_{|w^q|}, \neg w^q_{|\neg w^q|}\}$ that contains for all $j \in \{1, \ldots, q\}$ either $P(w_j)$ or $P(\neg w_j)$ but not both. Therefore, it represents in an obvious way an assignment of the variables ($w_j = true \Leftrightarrow P(w_j) \subset P$). From the choice of the labeling of vertices in D_F, it is easy to verify that an assignment of the variables satisfying n clauses corresponds to a maximal path collecting n labels. This transformation produces in polynomial time an instance D_F satisfying the conditions (C2) with $k \leq 3|\mathcal{L}|$. It remains to ensure (C1), this can be done by applying the transformation of Lemma 1. Together with Theorem 1, this concludes the proof of Theorem 2. □

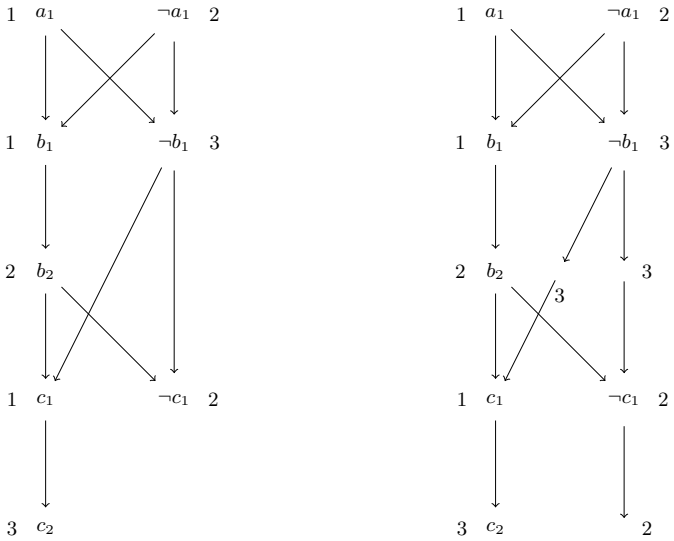

Fig. 1. The digraph D_F for the formula $F = (a \vee b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (\neg b \vee c)$ before the transformation of Lemma 1 (to the left) and after (to the right).

The next step consists in showing that the problem Max-Labeled-Path remains APX-hard even when restricted to instances such that all maximal paths have the same number of vertices and containing a path collecting each label exactly once.

Theorem 3. *Assuming $P \neq NP$, no polynomial time algorithm can achieve a performance ratio exceeding $\frac{23}{24}$ for* Max-Labeled-Path *even when restricted to instances satisfying (C1) and (C3).*

Proof. Consider a DAG $D = (V, A)$ with a labeling function l that satisfies the conditions (C1) with $k \leq 3|\mathcal{L}|$ and (C2). Every maximal path in D contains the same number k of vertices. Let

$m := |\mathcal{L}| \leq k$ be the number of labels of vertices in D. We construct a DAG D' by adding to D, for each vertex $v \in V$, a set $\{v^1, \ldots, v^r\}$ of $r := k - m$ copies of the vertex v. There is an arc between two vertices in D' if and only if there is an arc between their preimages in D (the preimage of a vertex $v \in V$ is v itself). Every maximal path in D' corresponds to a maximal path in D, in particular it contains exactly k vertices. The set of labels of D' is $\mathcal{L}' := \mathcal{L} \cup \{m+1, m+2, \ldots, m+r = k\}$. For each vertex v of D and each integer $j \in \{1, 2, \ldots, r\}$ the label of the vertex v^j is $m + j$. The labels in D' of the vertices that belong to D remain unchanged. We call the resulting instance D' the *extension* of the instance D.

The following two lemmata establish a close relationship between the optimum of the instances D and D'.

Lemma 2. *If there is a path in D collecting n labels then there is a path in D' collecting $n + r$ labels. If there is a path in D' collecting n labels then there is a path in D collecting at least $n - r$ labels.*

The proof of Lemma 2 follows by construction of D'.

Lemma 3. *If there exists a polynomial time algorithm that achieves a performance ratio $1 - \epsilon$ for* Max-Labeled-Path *restricted to instances satisfying conditions (C1) and (C3) then there exists a polynomial time algorithm that achieves a performance ratio $1 - 3\epsilon$ for* Max-Labeled-Path *restricted to instances satisfying conditions (C1) with $k \leq 3|\mathcal{L}|$ and (C2).*

Proof. Suppose there exists a polynomial time algorithm ALG$'$ that achieves a performance ratio $1 - \epsilon$ for Max-Labeled-Path restricted to instances D' satisfying conditions (C1) and (C3). For such an instance D', this algorithm computes in polynomial time a path P' collecting $(1 - \epsilon)OPT(D')$ labels where $OPT(D')$ is both the number k of vertices in a maximal path of D' and the number $|\mathcal{L}'|$ of labels in D', i.e. $OPT(D') = |\mathcal{L}'| = k$. For all instances D of Max-Labeled-Path satisfying the conditions (C1) with $k \leq 3|\mathcal{L}|$ and (C2), the following algorithm ALG, that uses ALG$'$ as routine, computes a path P that collects $(1 - 3\epsilon)OPT(D)$ labels where $OPT(D) = |\mathcal{L}|$ is the number of labels in D.

Algorithm 1: ALG(D) : a maximal path in D that collects $(1 - 3\epsilon)OPT(D)$ labels

Compute the extension D' of the instance D ;
Perform ALG$'$ on D' to compute a path P' that collects $(1 - \epsilon)OPT(D')$ labels of D';
Return the projection P of the path P' on D ;

To prove the correctness of algorithm ALG, first notice that the extension D' of D contains a path that collects each label of D' exactly once. Indeed, by assumption, the instance D satisfies condition (C2), thus there exists a path P^* in D that collects $m = |\mathcal{L}|$ labels. By Lemma 2, the extension of P^* to D' collects $m + (k - m) = k = |\mathcal{L}'|$ labels. It remains to prove that the path P returned by ALG collects at least $(1 - 3\epsilon)|\mathcal{L}|$ labels. By the choice of ALG$'$, P' collects $(1 - \epsilon)k$ labels. By Lemma 2, P collects at least $(1 - \epsilon)k - (k - m) = m - \epsilon k$ labels. Since D satisfies the condition $k \leq 3m$, P collects at least $(1 - 3\epsilon)m = (1 - 3\epsilon)OPT(D)$ labels. □

To complete the proof of Theorem 3, suppose that there exists a polynomial time algorithm ALG$'$ achieving a ratio exceeding $\frac{23}{24}$ for the problem Max-Labeled-Path restricted to the instances satisfying conditions (C1) and (C3). Then, by Lemma 3, we deduce that there exists a polynomial time algorithm ALG achieving a ratio exceeding $\frac{7}{8}$ for the problem Max-Labeled-Path restricted to the instances satisfying conditions (C1) with $k \leq 3|\mathcal{L}|$ and (C2), this cannot occur by Theorem 2, unless $P = NP$. □

The last result of this section shows that Max-Labeled-Path remains APX-hard if we add the condition that the number of vertices in any maximal path is a power of two.

Theorem 4. *Assuming $P \neq NP$, no polynomial time algorithm can achieve a performance ratio exceeding $\frac{47}{48}$ for* Max-Labeled-Path *even when restricted to instances satisfying conditions (C1) and (C4).*

Proof. Consider a DAG $D = (V, A)$ with a labeling function l that satisfies the conditions (C1) and (C3). Every maximal path in D contains the same number $k = |\mathcal{L}|$ of vertices. Let p be the smallest integer such that $k \leq 2^p$. The set $\mathcal{L}'$ of labels of D' is $\mathcal{L} \cup \{k+1, \ldots, k+r = 2^p = k'\}$. Let T be the set of vertices of D with no outgoing arcs. We consider the DAG D' obtained from D by adding a directed path $Q = (q_1, q_2, \ldots, q_r)$ and by connecting via an arc each vertex $t \in T$ to the vertex q_1. For $i = 1, \ldots, r$, we assign to the vertex q_i the new label $k + i$. The labels of the vertices of D' that belong to D remain unchanged. Using the above transformation, the fact $k' = 2^p \leq 2k$ and a proof similar to the one of Lemma 3, it is possible to derive a polynomial time algorithm ALG achieving a ratio $1 - 2\epsilon$ for instances of Max-Labeled-Path satisfying conditions (C1) and (C3) from a polynomial time algorithm ALG$'$ achieving a ratio $1 - \epsilon$ for instances of Max-Labeled-Path satisfying conditions (C1) and (C4). This last assertion and Theorem 3 imply Theorem 4 exactly in the same way that Lemma 3 and Theorem 2 imply Theorem 3. □

3 Maximum Labeled Path does not belong to APX

In this section, using a self-reduction of the problem Max-Labeled-Path, we will prove the following result:

Theorem 5. *Assuming $P \neq NP$, no polynomial time algorithm can achieve a constant performance ratio for* Max-Labeled-Path *even when restricted to instances satisfying conditions (C1) and (C4).*

3.1 Self-reduction

In Section 3, we consider only instances of Max-Labeled-Path satisfying conditions (C1) and (C4). Namely, a DAG $D = (V, A)$ whose vertices are labeled by a function $l : V \to \mathcal{L} = \{1, \ldots, k\}$ such that there exists a path collecting each label exactly once and the number $k = |\mathcal{L}|$ of vertices in any maximal path is a power of two. We will prove that such instances of the problem Max-Labeled-Path cannot be approximated in polynomial time within a constant factor. For the sake of simplicity, we also assume that there is only one source s and one sink t. Therefore, any maximal path is an st-path (i.e. a path from s to t) and all vertices of D belong to an st-path. Recall that, for each vertex $u \in V$, $\lambda(u)$ is the number of vertices in an su-path (all such paths have the same length because D satisfies (C4)). For all $u \in V$, $\lambda(s) = 1 \leq \lambda(u) \leq k = \lambda(t)$.

Pseudo square and pseudo cubic acyclic digraph. The *pseudo square digraph* $\bar{D}$ of D is obtained from D by replacing each vertex $u \in V$ by a copy D_u of the digraph D. We denote by v_u the copy of the vertex $v \in V$ in the digraph D_u. There is an arc $v_u w_u$ in $\bar{D}$ if and only if there is an arc vw in D. In addition to the arcs of the subgraphs D_u, $u \in V$, we add to $\bar{D}$ an arc $t_u s_v$ for each arc from uv in D. The *pseudo cubic digraph* $\tilde{D}$ of D is obtained from $\bar{D}$ by replacing each vertex v_u of $\bar{D}$ by a path $P(v_u)$ with k vertices. Each arc entering a vertex v_u in $\bar{D}$ is replaced by an arc of $\tilde{D}$ entering the first vertex of $P(v_u)$. Analogously, each arc leaving

the vertex v_u in $\bar{D}$ is replaced by an arc of $\tilde{D}$ leaving the last vertex of $P(v_u)$ (see Fig. 2). We define a new instance of Max-Labeled-Path on the digraph $\tilde{D}$ with the first vertex of $P(s_s)$ as a source and the last vertex of $P(t_t)$ as a sink and a labeling function $\tilde{l}$ defined as follows.

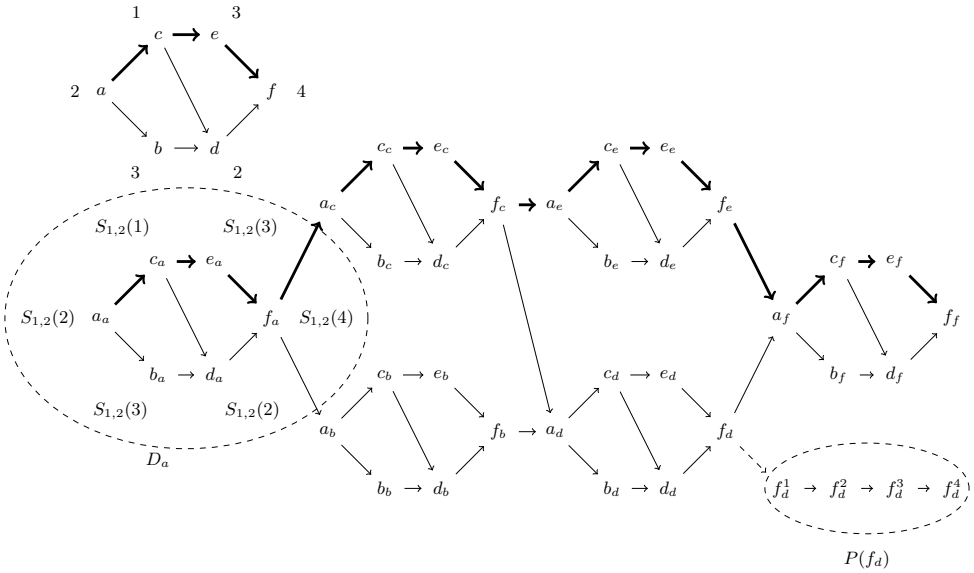

Fig. 2. An example of pseudo square digraph $\bar{D}$ with $k = |\mathcal{L}| = 4$. An optimal path P in D and the corresponding optimal path $\bar{P}$ in $\bar{D}$ are drawn in bold. In the subgraph D_a, each vertex v of $\bar{D}$ is labeled by the subset of labels received by the vertices of the path $P(v)$ of $\tilde{D}$. In $\tilde{D}$, the vertex f_d of $\bar{D}$ is replaced by the path $P(f_d) = (f_d^1, f_d^2, f_d^3, f_d^4)$.

Labeling. Let v_u be a vertex of $\tilde{D}$, the set of labels of the vertices of $P(v_u)$ will depend on the labels of u and v in D and on the level of u in D. Since a path from the source to the sink visits either all vertices of $P(v_u)$ or none of them, our labeling function assigns a set of labels to the path $P(v_u)$ and does not precise the order in which the labels appear on $P(v_u)$. The set of labels $\tilde{\mathcal{L}}$ used to define the labeling of $\tilde{D}$ consists of k disjoint subsets $\tilde{\mathcal{L}}_1, \ldots, \tilde{\mathcal{L}}_k$ such that $|\tilde{\mathcal{L}}_1| = \ldots = |\tilde{\mathcal{L}}_k| = k^2$. For each label $c \in \mathcal{L}$ and each level $i \in \{1, \ldots, k\}$, we construct a partition $\mathcal{S}_{i,c} := \{S_{i,c}(c') : c' \in \mathcal{L}\}$ of $\tilde{\mathcal{L}}_c$ into k subsets of size k such that any two subsets arising from different partitions intersect in exactly one label, i.e. if $i_1 \neq i_2$ then for all $c', c'' \in \mathcal{L}$, $|S_{i_1,c}(c') \cap S_{i_2,c}(c'')| = 1$. Since k^2 is a power of two ($k^2 = 2^r$), such partitions can be easily constructed as classes of parallel lines of a finite affine plane (each class of parallel lines induces a partition in which the subsets are the lines). The construction of finite affine planes from finite fields is described for instance in [1]. This construction can be done in polynomial time in the size of D by first identifying an irreducible polynomial of degree r by brute force and then constructing the corresponding finite fields $GF(2^r)$. The labeling function $\tilde{l}$ assigns to the vertices of $P(v_u)$ the labels that belong to the subset $S_{\lambda(u),l(u)}(l(v))$ of the partition $\mathcal{S}_{\lambda(u),l(u)}$.

Claim. There is a path in $\tilde{D}$ that collects each label in $\tilde{\mathcal{L}}$ exactly once.

Proof. Let P be the path of D collecting all the labels in $\mathcal{L}$. Consider the path $\tilde{P}$ passing via each subgraph D_u for all $u \in P$ and such that the subpath $\tilde{P}_u$ of $\tilde{P}$ inside the subgraph D_u consists of the vertices v_u for all $v \in P$ (see Fig 2). Since P collects each label in $\mathcal{L}$ once, the subpath $\tilde{P}_u$ collects every subset of the partition $\mathcal{S}_{\lambda(u),l(u)}$. This implies that $\tilde{P}_u$ collects each label of $\tilde{\mathcal{L}}_{l(u)}$ once. Applying this assertion to all vertices $u \in P$ and using again that P collects each label in $\mathcal{L}$, we conclude that $\tilde{P}$ collects all the labels of $\tilde{\mathcal{L}} = \bigcup_{u \in P} \tilde{\mathcal{L}}_{l(u)}$ once. □

The previous claim and the fact that $|\tilde{\mathcal{L}}|$ is a power of two ensure that $\tilde{D}$ is an instance of Max-Labeled-Path satisfying the conditions of (C1) and (C4). Clearly, the instance $\tilde{D}$ can be constructed in polynomial time from the instance D.

3.2 Proof of Theorem 5

Let g denote the reciprocal function on the interval $]0, 1]$ of the following continuous and strictly increasing function h:

$$h(x) := \begin{cases} h_1(x) := x(x^2 - x + 1) \text{ if } 0 < x < \frac{1}{2}; \\ h_2(x) := x^2 - \frac{1}{4}x + \frac{1}{4} \quad \text{if } \frac{1}{2} \leq x \leq 1. \end{cases}$$

Lemma 4. *For each $0 < \beta < 1$, the sequence β_n defined by $\beta_0 = \beta$ and $\beta_{n+1} = g(\beta_n)$ has a limit of 1.*

Proof. If $x \in [\frac{1}{2}, 1[$ then the difference $h_2(x) - x = x^2 - \frac{5}{4}x + \frac{1}{4} = (x-1)(x-\frac{1}{4})$ is negative. Now suppose that $x \in]0, \frac{1}{2}[$, since $\frac{h_1(x)}{x} = x^2 - x + 1 < 1$, $h_1(x) < x$. We conclude that for all $x \in]0, 1[$, $h(x) < x$. Since h is strictly increasing on the interval $]0, 1[$, so is g and applying it on the two sides of this inequality, we obtain that $x < g(x)$ for all $x \in]0, 1[$. This implies that the sequence β_n is strictly increasing and bounded by 1. Therefore its limit is a real l such that $g(l) = l \Leftrightarrow l = h(l)$. Since $\beta_0 > 0$ and $\frac{h_1(l)}{l} = l^2 - l + 1 < 1$ for all $l \in]0, \frac{1}{2}[$, we deduce that $l \geq \frac{1}{2}$. Finally, using $h_2(l) - l = (l-1)(l-\frac{1}{4}) = 0$ and $l > \frac{1}{4}$, we conclude that the limit of the sequence β_n is 1. □

In the next section, we show the following results:

Lemma 5. *Given any path Q in $\tilde{D}$ that collects at least βk^3 labels, a path P in D that collects at least $g(\beta)k$ labels can be computed in polynomial time.*

This construction itself implies the following lemma:

Lemma 6. *If there is a polynomial-time algorithm with a ratio β for* Max-Labeled-Path *then there is a polynomial-time algorithm with a ratio $g(\beta)$ for* Max-Labeled-Path.

Proof. Suppose there exists a polynomial time algorithm ALG_β with a ratio at least β for Max-Labeled-Path. Let D be an instance of Max-Labeled-Path, we use the following algorithm:

Algorithm 2: ALG(D) : a maximal path in D that collects $g(\beta)k$ labels

Construct the digraph $\tilde{D}$ from the digraph D;
Perform ALG_β to obtain a path Q of $\tilde{D}$ that collects βk^3 labels;
Derive from Q a path P of D that collects at least $g(\beta)k$ labels;
Return P;

This algorithm is clearly polynomial because all the steps are, thus we have a polynomial time algorithm with a ratio $g(\beta)$ for Max-Labeled-Path. □

Suppose there exists an approximation algorithm with a constant factor β for Max-Labeled-Path. By Lemma 4, there exists an integer n such that $\beta_n > \frac{47}{48}$. Applying n times Lemma 6, we derive a polynomial-time algorithm for the problem Max-Labeled-Path with a ratio exceeding $\frac{47}{48}$. A similar argument shows that any constant factor approximation algorithm for Max-Labeled-Path can be converted into a PTAS for this problem. Such an algorithm does not exist unless $P = NP$ by Theorem 4. Assuming Lemma 5, this concludes the proof of Theorem 5.

3.3 Proof of Lemma 5

We explain how to construct in polynomial time a path P in D that collects a set $\mathcal{L}^P \subseteq \mathcal{L}$ containing at least $g(\beta)k$ labels from a path Q in $\tilde{D}$ that collects a set $\tilde{\mathcal{L}}^Q \subseteq \tilde{\mathcal{L}}$ containing at least βk^3 labels. We denote by $V^Q \subseteq V$ the set of vertices u such that Q passes via D_u and by $\mathcal{L}^Q \subseteq \mathcal{L}$ the set of labels of the vertices in V^Q. For each vertex $u \in V^Q$, we define $W_u^Q \subseteq V$ the set of vertices v such that Q contains $P(v_u)$ as a subpath and by $\mathcal{L}_u^Q \subseteq \mathcal{L}$ the set of labels of the vertices in W_u^Q. Let $\alpha_u := |\mathcal{L}_u^Q|/k$. We will prove that either $|\mathcal{L}^Q| \geq g(\beta)k$ or there exists a vertex $u \in V^Q$ such that $|\mathcal{L}_u^Q| = \alpha_u k \geq g(\beta)k$. In the first case, the vertices of V^Q induce in D a path that collects $g(\beta)k$ labels. In the second case, the vertices of Q that belong to the subgraph D_u induce in D a path that collects $g(\beta)k$ labels. Therefore, if one of the two assertions hold, one can derive in polynomial time a path P of D collecting $g(\beta)k$ labels and we are done.

Suppose by way of contradiction that none of the two assertions hold. Namely, $|\mathcal{L}^Q| < g(\beta)k$ and for all $u \in V^Q$, $\alpha_u < g(\beta)$. Let c be a label in $\mathcal{L}^Q$. We denote by $V_c^Q \subseteq V^Q$ the set of vertices $u \in V^Q$ such that $l(u) = c$ and we define $\alpha_c := \max_{u \in V_c^Q} \alpha_u$ and $u_c := \arg\max_{u \in V_c^Q} \alpha_u$. By assumption, $\alpha_c < g(\beta)$. In D_{u_c}, Q collects $\sum_{c' \in \mathcal{L}_{u_c}^Q} |S_{c,\lambda(u)}(c')| = \sum_{c' \in \mathcal{L}_{u_c}^Q} k = \alpha_c k^2$ labels.

Let u be a vertex of $V_c^Q - \{u_c\}$. The number of labels collected by Q in D_u that are not collected by Q in D_{u_c} is the sum over all labels $c' \in \mathcal{L}_u^Q$ of

$$\begin{aligned}
\left| S_{c,\lambda(u)}(c') - \bigcup_{c'' \in \mathcal{L}_{u_c}^Q} S_{c,\lambda(u_c)}(c'') \right| &= k - \left| \bigcup_{c'' \in \mathcal{L}_{u_c}^Q} \left(S_{c,\lambda(u)}(c') \cap S_{c,\lambda(u_c)}(c'') \right) \right| \\
&= k - \sum_{c'' \in \mathcal{L}_{u_c}^Q} \left| S_{c,\lambda(u)}(c') \cap S_{c,\lambda(u_c)}(c'') \right| \\
&= k - \sum_{c'' \in \mathcal{L}_{u_c}^Q} 1 \\
&= k - \alpha_c k
\end{aligned}$$

The first equation follows from $\left|S_{c,\lambda(u)}(c')\right| = k$ and set properties. For the second equation, recall that the family $\{S_{c,\lambda(u_c)}(c'') : c'' \in \mathcal{L}_{u_c}^Q\}$ is a partition of $\tilde{\mathcal{L}}_c$. The choice of the partitions used to define the labeling function of $\tilde{D}$ ensures that $\left|S_{c,\lambda(u)}(c') \cap S_{c,\lambda(u_c)}(c'')\right| = 1$ and yields the third equation. For the last equation, we use $|\mathcal{L}_{u_c}^Q| = \alpha_c k$. We conclude that the number of labels collected by Q in D_u and not collected by Q in D_{u_c} is $|\mathcal{L}_u^Q|(k - \alpha_c k)$. Since $(k - \alpha_c k) \geq 0$ and $|\mathcal{L}_u^Q| = \alpha_u k \leq \alpha_c k$, this number is at most $\alpha_c k(k - \alpha_c k)$.

Using this bound for all vertices $u \in V_c^Q - \{u_c\}$ and the fact that $\alpha_c k^2$ labels are collected by Q in D_{u_c}, we obtain the following bound on the number of labels of $\tilde{\mathcal{L}}_c$ collected by Q:

$$\begin{aligned}
\left| \tilde{\mathcal{L}}^Q \cap \tilde{\mathcal{L}}_c \right| &\leq \alpha_c k^2 + (|V_c^Q| - 1)\alpha_c k(k - \alpha_c k) \\
&\leq k^2(\alpha_c + \alpha_c(|V_c^Q| - 1)(1 - \alpha_c))
\end{aligned}$$

Summing over all labels $c \in \mathcal{L}^Q$, we obtain that the total number of labels collected by Q is upper bounded as follows:

$$\begin{aligned}\left|\tilde{\mathcal{L}}^Q\right| &\leq k^2 \sum_{c\in\mathcal{L}^Q} \left(\alpha_c + \alpha_c(|V_c^Q|-1)(1-\alpha_c)\right) \\ &< k^2 \sum_{c\in\mathcal{L}^Q} \left(g(\beta) + \alpha_c(|V_c^Q|-1)(1-\alpha_c)\right) \qquad (*)\end{aligned}$$

This last inequality is obtained using the initial assumption $\alpha_c < g(\beta)$.

We distinguish two cases depending on the value of $g(\beta)$. First, suppose that $g(\beta) \geq \frac{1}{2}$. Note that the maximum $\frac{1}{4}$ of the function $x(1-x)$ on the interval $[0,1]$ is realized for $x = \frac{1}{2}$. Therefore for all $c \in \mathcal{L}^Q$, $\alpha_c\,(1-\alpha_c) \leq \frac{1}{4}$ and we derive from $(*)$:

$$\begin{aligned}\left|\tilde{\mathcal{L}}^Q\right| &< k^2 \sum_{c\in\mathcal{L}^Q} \left(g(\beta) + \frac{1}{4}(|V_c^Q|-1)\right) \\ &< k^2 \left(\left(g(\beta)-\frac{1}{4}\right)\sum_{c\in\mathcal{L}^Q} 1 + \frac{1}{4}\sum_{c\in\mathcal{L}^Q}|V_c^Q|\right) \\ &< k^2 \left(\left(g(\beta)-\frac{1}{4}\right)g(\beta)k + \frac{1}{4}k\right) \\ &< k^3 \left(g(\beta)^2 - \frac{1}{4}g(\beta) + \frac{1}{4}\right) \\ &< k^3\,(h(g(\beta))) \\ &< k^3\beta\end{aligned}$$

In the third inequality, the upper bound on the left operand follows from the initial assumption $g(\beta)k > |\mathcal{L}^Q| = \sum_{c\in\mathcal{L}^Q} 1$ and $(g(\beta)-\frac{1}{4}) \geq 0$. The upper bound on the right operand follows from the fact that any path in D from s to t contains exactly k vertices, therefore $\sum_{c\in\mathcal{L}^Q}|V_c^Q| = k$. The last equation contradicts the choice of Q and concludes the proof for the case $g(\beta) \geq \frac{1}{2}$.

Now, suppose that $g(\beta) < \frac{1}{2}$. Since the function $x(1-x)$ is a strictly increasing function on the interval $]0, \frac{1}{2}[$ and $|V_c^Q|-1 \geq 0$ for all $c \in \mathcal{L}^Q$, we can replace α_c by $g(\beta)$ in the inequation $(*)$:

$$\begin{aligned}|\tilde{\mathcal{L}}^Q| &< k^2 \sum_{c\in\mathcal{L}^Q} \left(g(\beta) + g(\beta)(|V_c^Q|-1)\,(1-g(\beta))\right) \\ &< k^2 g(\beta)\left(\sum_{c\in\mathcal{L}^Q} 1 - (1-g(\beta)) + |V_c^Q|\,(1-g(\beta))\right) \\ &< k^2 g(\beta)\left(g(\beta)\sum_{c\in\mathcal{L}^Q} 1 + (1-g(\beta))\sum_{c\in\mathcal{L}^Q}|V_c^Q|\right) \\ &< k^2 g(\beta)\left(g(\beta)^2 k + (1-g(\beta))\,k\right) \\ &< k^3 g(\beta)\left(g(\beta)^2 - g(\beta) + 1\right) \\ &< k^3 h(g(\beta)) \\ &< k^3\beta\end{aligned}$$

Again we use $\sum_{c\in\mathcal{L}^Q} 1 < g(\beta)k$ and $\sum_{c\in\mathcal{L}^Q}|V_c^Q| = k$ to derive the fourth inequality. In the two cases, we obtain a contradiction with the assumption that the path Q collects at least βk^3 labels. This concludes the proof of Lemma 5.

4 $\sqrt{OPT}$-approximation algorithm

4.1 Algorithm

In this section, we consider the following weighted version of Max-Labeled-Path: given a DAG D, a labeling function $l : V \to \mathcal{L}$ and a weight function $w : \mathcal{L} \to \mathbb{N}$ defined on the set of labels, the problem Max-Weighted-Labeled-Path consists in computing a path P collecting a set of labels of maximum total weight, i.e. a path P such that $w(P) := \sum_{l \in \mathcal{L}^P} w(l)$ is maximum. Note that the weight of a label that appears several times in P counts only once.

We describe a polynomial time algorithm that computes for each instance D of Max-Weighted-Labeled-Path, a path P of D whose total weight is at least $\sqrt{OPT(D)}$. Again, for the sake of simplicity, we assume that there is only one source s and one sink t. Our algorithm can be easily adapted to handle the case with several sources and several sinks. Without loss of generality, we can also suppose that (i) the source s of D has no label (an equivalent instance satisfying this condition can be obtained by adding to D a new source s' with no label and an arc $s's$); (ii) all other vertices have a label of positive weight (a vertex having a label of weight zero can be removed by replacing each path of length two passing through such a vertex by an arc.) First, we define a function $F : V \to \mathbb{N}$ such that $F(u)$ can be computed for all vertices $u \in V$ in time $O(|V|^3)$. Then, we prove that, for any vertex $u \in V$, $F(u)$ is an upper bound on the total weight of labels collected by an su-path. Finally, we describe an algorithm that computes for any vertex $u \in V$ a path P such that $w(P)$ is at least $\lfloor\sqrt{F(u)}\rfloor$. Applying this algorithm to t, we obtain an st-path that collects labels of total weight at least $\lfloor\sqrt{OPT}\rfloor$.

For each pair of vertices $u, v \in V$, let $D_{u,v}$ be the subgraph of D induced by the vertices lying on an uv-path. We denote by $\Gamma(u, v)$ the total weight of the labels collected by the vertices of $D_{u,v}$. Let $F : V \to \mathbb{N}$ be the function recursively defined as follows :

$$F(u) := \begin{cases} w(s), \text{ if } u = s \ ; \\ \max\limits_{P \in \mathcal{P}^u} \min\limits_{xy \in P} F(x) + \Gamma(y, u), \text{ otherwise.} \end{cases}$$

where $\mathcal{P}^u$ denotes the set of paths from s to u. Let $P(u)$ be a path in $\mathcal{P}^u$ that realizes the maximum, i.e. such that $F(u) = \min\limits_{xy \in P(u)} F(x) + \Gamma(y, u)$.

The following lemma shows that, for any vertex $u \in V$, $F(u)$ is an upper bound on the total weight of labels that can be collected by an su-path.

Lemma 7. *If $P = (s = u_0, u_1, ..., u_n = u)$ is an su-path then $F(u) \geq w(P)$.*

Proof. By induction on n. For $n = 0$, $F(u_0) = F(s) = w(s)$. For $n > 0$, consider a path $P = (s = u_0, u_1, ..., u_n = u)$. For any $i = 1, \ldots, n$, let P_i be the path $(u_0, u_1, ..., u_i)$. The weight of the path $(u_i, ..., u_n)$ is at least $w(P) - w(P_{i-1})$ and it belongs to $D_{u_i,u}$, therefore $\Gamma(u_i, u) \geq w(P) - w(P_{i-1})$. Since, by induction, $F(u_{i-1}) \geq w(P_{i-1})$, $F(u_{i-1}) + \Gamma(u_i, u) \geq w(P)$ for any $i = 1, \ldots, n$ yielding $F(u) \geq w(P)$. □

Corollary 1. *If OPT is the maximum weight of labels that can be collected by a path from s to t then $F(t) \geq OPT$.*

Suppose that $F(v)$ and $P(v)$ have been already computed for all $v \in V$, we explain below how to compute them in $O(|V|^3)$. Let u be a vertex in V. The algorithm `ComputePath` returns an su-path that collects labels of total weight at least $\lfloor\sqrt{F(u)}\rfloor$. By Corollary 1, applying this procedure with $u = t$ we obtain an st-path that collects labels of total weight at least $\lfloor\sqrt{OPT}\rfloor$.

Algorithm 3: `ComputePath`$(u \in V)$: a path P from s to u such that $w(P) \geq \lfloor\sqrt{F(u)}\rfloor$

```
if u = s then
    return (s)
else
    Let xy be an arc of P(u) with F(x) ≤ (⌊√F(u)⌋ − 1)² and F(y) ≥ (⌊√F(u)⌋ − 1)² ;
    P' ← ComputePath(y) ;
    if w(P') ≥ ⌊√F(u)⌋ then
        return P'.Q where Q is any yu-path ;
    else
        Perform a BFS in D_{y,u} to find a vertex v having a label not collected by P' ;
        return P'.Q where Q is any yu-path passing via the vertex v ;
```

The following lemma is useful to prove that the algorithm `ComputePath` is correct.

Lemma 8. *There is an arc xy in $P(u)$ such that $F(x) \leq (\lfloor\sqrt{F(u)}\rfloor-1)^2$ and $F(y) \geq (\lfloor\sqrt{F(u)}\rfloor-1)^2$. Moreover, for any such arc, $\Gamma(y,u) \geq \lfloor\sqrt{F(u)}\rfloor$.*

Proof. The first assertion is true because $F(s) = 0 \leq (\lfloor\sqrt{F(u)}\rfloor-1)^2$ and $F(u) \geq (\lfloor\sqrt{F(u)}\rfloor-1)^2$. To verify the second assertion, let xy be an arc such that $F(x) \leq (\lfloor\sqrt{F(u)}\rfloor - 1)^2$ and $F(y) \geq (\lfloor\sqrt{F(u)}\rfloor - 1)^2$. Since $xy \in P(u)$, $F(x) + \Gamma(y,u) \geq F(u)$. This implies $\Gamma(y,u) \geq F(u) - F(x) \geq \lfloor\sqrt{F(u)}\rfloor^2 - (\lfloor\sqrt{F(u)}\rfloor - 1)^2 = 2\lfloor\sqrt{F(u)}\rfloor - 1 \geq \lfloor\sqrt{F(u)}\rfloor$. □

Theorem 6. `ComputePath`(u) *computes a path P such that $w(P) \geq \lfloor\sqrt{F(u)}\rfloor$.*

Proof. We proceed by induction on the number of recursive calls. If $u = s$ the algorithm returns the path (s) that collects the label $l(s)$ of weight $F(s) = w(s)$. Otherwise, the first assertion of Lemma 8 ensures that $P(u)$ contains an arc xy such that $F(x) \leq (\lfloor\sqrt{F(u)}\rfloor - 1)^2$ and $F(y) \geq (\lfloor\sqrt{F(u)}\rfloor - 1)^2$. By induction hypothesis, `ComputePath`(y) returns a path P' such that $w(P') \geq \lfloor\sqrt{F(u)}\rfloor - 1$. If $w(P') \geq \lfloor\sqrt{F(u)}\rfloor$, then the path $P'.Q$ returned by the algorithm is a correct answer. Now, suppose that $w(P') = \lfloor\sqrt{F(u)}\rfloor - 1$. By Lemma 8, $\Gamma(y,u) \geq \lfloor\sqrt{F(u)}\rfloor$. This implies that $D_{y,u}$ contains at least one label not collected by P' (and thus distinct from $l(y)$). A BFS traversal of $D_{y,u}$ will find a vertex $v \neq y$ having this label together with a path Q from y to u passing via v. Since $w(l(v)) \geq 1$, the path $P'.Q$ that collects labels of total weight at least $\lfloor\sqrt{F(u)}\rfloor - 1 + w(l(v))$ is a correct answer. □

Finally, the following algorithm computes $F(u)$ and $P(u)$ for every vertex $u \in V$ in time $O(|V|^3)$. First, we compute the transitive closure of the graph D represented by a matrix $M(u,v)$, $u \in V, v \in V$, such that $M(u,v) = 1$ if there is an uv-path in D, and $M(u,v) = 0$ otherwise. Then, we compute for every pair of vertices uv the set of labels $L(u,v)$ of the vertices of $D(u,v)$ (recall that $\Gamma(u,v) = |L(u,v)|$). This can be done in $O(|V|^3)$ using the matrix M. Finally, we consider the vertices of V in increasing order of their distance to s, i.e. when we compute $F(u)$ the value of $F(x)$ have been already computed for all vertices $x \in D(s,u)$. Therefore, computing $F(u)$ reduces to solve a bottleneck shortest path problem in the graph $D(s,u)$ where the capacity of an arc xy is $F(x) + \Gamma(y,u)$. This problem can be solved in linear time in a DAG using a *Dijkstra-like* algorithm. Notice that an implicit representation of $D(s,u)$ is available from the matrix M and the representation of D. Therefore, computing $F(u)$ and $P(u)$ for every vertex $u \in V$ can be done in time $O(|V|^3)$.

4.2 Tight example for `ComputePath`

In this section, we describe an instance of the problem Max-Labeled-Path showing that our analysis of algorithm `ComputePath` is tight even in the unweighted case, i.e $w(l) = 1$ for all $l \in \mathcal{L}$. Let D_9 be the graph of Fig 3. Let s be the source and t the sink of this graph. One can easily check that $F(u) = \lambda(u)$, for all $u \in V$. In particular, $F(t) = 9$. Indeed, with respect to the function F each vertex located in the upper path is equivalent to the vertex in same level set located in the lower path. Moreover, $F(x) + \Gamma(y, t)$ is at least 9, for each arc xy in the upper path. Therefore, we can suppose that the path $P(t)$ chosen by the algorithm is the upper path. When the algorithm `ComputePath` is invoked with t as parameter, it explores $P(t)$ until it finds an arc xy such that $F(x) \leq 4$ and $F(y) \geq 4$. Then it recursively calls `ComputePath`(y) that may return the upper path P' from s to y that collects two labels 1 and 10. Then, the BFS in $D_{y,t}$ find the vertex v located to the right of y and labeled with 9. The path Q from y to t which passes via v may be again the upper yt-path. Finally, `ComputePath`(t) may return the upper path which collects 3 labels whereas the optimum is the lower path which collects 9 labels. This example shows that our analysis of the algorithm `ComputePath` is tight. Clearly, arbitrarily large examples can be obtained using an analogous construction.

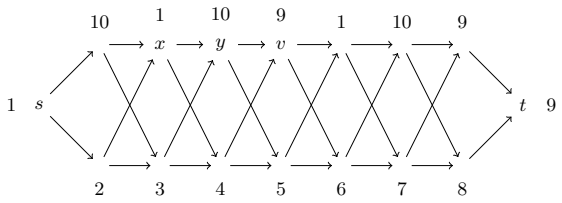

Fig. 3. D_9, a tight example for `ComputePath`

5 $OPT^{1-\epsilon}$-approximation for Max-Labeled-Path

In this section, we describe a method for improving the performance guarantee of our $\sqrt{OPT}$-approximation for Max-Labeled-Path. We first describe the method in the unweigted case and then explain how to handle weigths on the labels. Namely, we provide a construction that tranforms an algorithm that returns paths collecting $cOPT^{\alpha}$ labels into an algorithm that returns $c'OPT^{\alpha'}$ labels where $\alpha' = \frac{1}{2-\alpha}$ and $c' = (\frac{2}{3}c)^{\alpha}$. Starting with our $\sqrt{OPT}$-approximation algorithm and applying n time this transformation, we obtain an algorithm that returns paths collecting at least $(\frac{2}{3})^{\frac{n}{2}}OPT^{\frac{n}{n+1}}$ labels. For any $\epsilon > 0$, there exists an integer n such that $\frac{n}{n+1} > 1 - \epsilon$, thus, for OPT large enough, $(\frac{2}{3})^{\frac{n}{2}}OPT^{\frac{n}{n+1}} \geq OPT^{1-\epsilon}$ (if OPT is bounded then an optimal solution can be computed in polynomial time by dynamic programming).

Theorem 7. *For any $\epsilon > 0$ and any instance D of* Max-Labeled-Path*, a path collecting $OPT(D)^{1-\epsilon}$ labels can be computed in polynomial time.*

5.1 Preliminaries

Let D be an instance of Max-Labeled-Path, i.e. a DAG labeled by a function $l : V(D) \to \mathcal{L}$. Consider a subset $\mathcal{L}' \subseteq \mathcal{L}$ of the labels, the projection of the instance D on $\mathcal{L} - \mathcal{L}'$ is the instance

$D - \mathcal{L}'$ obtained from D by removing the labels of $\mathcal{L}'$. In the projected instance, some vertices may have no label. These vertices can be removed by replacing each path of length two passing through such a vertex by an arc (as it was done in Section 4.1 for vertices having zero weight). Recall that for any pair of vertices $x, y \in V$, $D_{x,y}$ denotes the digraph induced by the vertices lying on an xy-path. A vertex y is said to be *covered* by a vertex x if $y \in V(D_{x,t})$. By extension, y is said to be *covered* by a set of vertices U if at least one vertex of U *covers* y. The sources of a set of vertices U, denoted by sources(U), are the vertices of the minimal subset of U covering all vertices of U, i.e. a vertex u belongs to sources(U) if and only if $u \in U$ and no other vertex of U covers u.

5.2 Proof of Theorem 7

Let A be an algorithm that given a labeled DAG D, with a unique source $s \in V(D)$, and an integer β, returns a collection of paths $\{P_u : u \in U\}$ where P_u is an su-path that collects exactly β labels. For any $0 < \alpha < 1$, we say that $A \in \mathcal{A}_\alpha$ if there exists a constant c such that for any instance D and for any sy-path of D collecting p labels, the collection of paths obtained by running A with parameter $\beta = cp^\alpha$ on D contains an su-path P_u such that u covers y.

The following algorithm ByLevels$_{\beta,k}$ uses an algorithm $A \in \mathcal{A}_\alpha$ as an oracle. It takes as input a labeled DAG D with a unique source $s \in V(D)$ and returns a collection of paths $\{P_u : u \in U\}$ where P_u is a path between s and u that collects exactly βk labels. We will show below that for appropriate choices of parameters β and k, ByLevels$_{\beta,k}$ belongs to $\mathcal{A}_{\frac{1}{2-\alpha}}$. The idea of this algorithm is to split the DAG D into k levels and to collect β labels in each level. The levels are characterized by the subsets of vertices $S_0, \ldots, S_k$. The subset S_i consists of vertices v ending a path $P(v)$ collecting $i\beta$ labels discovered during iterations $1, 2, \ldots, i$. The ith level consists of all vertices which are covered by S_{i-1} but not by S_i. During the ith iteration, for each vertex v in S_{i-1}, the algorithm A is called on the instance obtained from D by removing the $i\beta$ labels collected by $P(v)$ and all vertices not lying on a path between the vertex v and the sink t. The set of vertices U_v for which algorithm A returns a path Q_u is added to S_i. The path P_u obtained by concatenating Q_u to the path P_v collects exactly $i\beta$ distinct labels. At the end of the ith iteration, the vertices of S_i covered by other vertices of S_i are removed from S_i.

Algorithm 4: ByLevels$_{\beta,k}(D)$: a collection $\{P_u : u \in U\}$ of su-paths collecting βk labels

```
S_0 = {s}, P(s) = {s};
for i = 1 ... k do
    S_i ← ∅;
    forall v ∈ S_{i-1} do
        Let D_{v,t} − L^{P(v)} be the projection on L − L^{P(v)} of the instance D_{v,t};
        Let {Q_u : u ∈ U_v} be the paths returned by A with parameter β on D_{v,t} − L^{P(v)};
        S_i ← S_i ∪ U_v;
        forall u ∈ U_v do
            P(u) ← P(v).Q(u);
    S_i ← sources(S_i);
return {P_u : u ∈ S_k}
```

Lemma 9. *For any labeled DAG D and any integers β and k,* ByLevels$_{\beta,k}(D)$ *returns a collection of paths collecting βk labels.*

Proof. Without loss of generality, we suppose that the source s of D has no label (an equivalent instance satisfying this condition can be obtained by adding to D a new source s' with no label and an arc $s's$). By induction on i, we prove that a path collecting exactly βi labels exists between s and every vertex $u \in S_i$. Since $S_0 = \{s\}$ and s has no label, the condition is true for $i = 0$. Now, suppose that the condition is satisfied for $i-1$. During ith iteration of algorithm `ByLevels`, if a vertex u is added to S_i then algorithm A has computed a path Q_u collecting β labels between a vertex $v \in S_{i-1}$ and u. By induction, there exists a path $P(v)$ collecting $\beta(i-1)$ labels between s and v. Since Q_u has been computed in a labeled graph in which the labels collected by P_v have been removed, the path $P_v.Q_u$ collects exactly βi labels. □

Let D be a DAG containing an sy-path P collecting p distinct labels. We denote $u_1, \ldots, u_p$ the vertices where a new label is encountered for the first time in path P. For convenience, we fix also $u_0 = s$. Since s has no label, $u_0 \neq u_1$. We denote $\widetilde{\beta} = (\beta/c)^{\frac{1}{\alpha}}$ and define recursively the function μ as follows:

$$\mu(0) := 0$$
$$\mu(i+1) := \mu(i) + i\beta + \widetilde{\beta}$$

Lemma 10. *For all i such that $\mu(i) \leq p$, the subset of vertices S_i computed by* $\texttt{ByLevels}_{\beta,k}(D)$ covers $u_{\mu(i)}$.

Proof. By induction. $S_0 = \{s\}$ covers $u_0 = s$. Now, suppose that $v \in S_i$ covers $u_{\mu(i)}$. The subpath of P between $u_{\mu(i)+1}$ and $u_{\mu(i+1)}$ belongs to $D_{v,t}$ and contains $i\beta + \widetilde{\beta}$ distinct labels. Since $|\mathcal{L}^{P(v)}| = i\beta$, the labeled DAG $D_{v,t}$ obtained from D by removing the labels of P_v and the vertices not covered by v contains a path ending in $u_{\mu(i+1)}$ and collecting at least $\widetilde{\beta}$ distinct labels. Since algorithm $A \in \mathcal{A}_\alpha$ is called with parameter β on $D_{v,t}$, it returns a path ending in an ancestor of $u_{\mu(i+1)}$. □

Now consider a labeled DAG D, containing an sy-path P with p distinct labels. The procedure $\texttt{ByLevels}_{\beta,k}$ called on D with $\beta = (\frac{2c^{2/\alpha}p}{3})^{\frac{\alpha}{2-\alpha}}$ and $k = \frac{\beta^{\frac{1}{\alpha}-1}}{c^{\frac{1}{\alpha}}}$ returns a path collecting $\beta k = (\frac{2}{3}cp)^{\frac{1}{2-\alpha}}$ labels ending in a vertex v covering y. Indeed, since by definition, $\mu(k) = k\widetilde{\beta} + \frac{k(k-1)}{2}\beta \leq k\widetilde{\beta} + \frac{k^2}{2}\beta$, we deduce

$$\mu\big(\frac{\beta^{\frac{1}{\alpha}-1}}{c^{\frac{1}{\alpha}}}\big) \leq \frac{3}{2c^{\frac{2}{\alpha}}}\beta^{\frac{2-\alpha}{\alpha}} = p.$$

Therefore, $u_{\mu(k)}$ is a vertex of P and, by Lemma 10, S_k must contain a vertex v covering $u_{\mu(k)}$. By Lemma 9, $\texttt{ByLevels}_{\beta,k}$ returns a path ending in v and collecting exactly $\beta k = c'p^{\alpha'}$ labels with $c' = (\frac{2}{3}c)^{\frac{1}{2-\alpha}}$ and $\alpha' = \frac{1}{2-\alpha}$ and thus $\text{ByLevels}_{\beta,k} \in \mathcal{A}_{\frac{1}{2-\alpha}}$.

The above construction transforms any polynomial time algorithm $A \in \mathcal{A}_\alpha$, into a polynomial time algorithm $A' \in \mathcal{A}_{\frac{1}{2-\alpha}}$ thus concluding the proof of Theorem 7.

5.3 Weighted version

First notice Max-Weighted-Labeled-Path can be reduced to Max-Labeled-Path by assigning to each label $l \in \mathcal{L}$ of weight $w(l)$ a set W_l of $w(l)$ labels of weight 1 and splitting each vertex u into a path consisting of $w(l(u))$ vertices labeled the labels of the set $W_{l(u)}$. This reduction is polynomial only if the weights of the labels are polynomial with respect to the size

of the initial instance. This is clearly not true in general. However, by scaling the weights of the labels, we can ensure polynomial weights while preserving the approximation guarantee. Let $\lambda := \lfloor \frac{OPT(D)}{2n} \rfloor$ where n is the number of vertices in the DAG D. We suppose $\lambda \geq 1$ because otherwise no scaling is required. The weights of the instance D' obtained from D by replacing the weight $w(l)$ of each label $l \in \mathcal{L}$ by $\lfloor w(l)/\lambda \rfloor$ is clearly polynomial with respect to the size of D. Therefore, the reduction of D' to the unweighted version leads to a polynomial time algorithm. Let P be the path returned by our $OPT^{1-\epsilon}$-approximation on the unweighted instance. The weight (before scaling) of the labels collected by P is at least

$$\begin{aligned} \lambda \times (OPT(D'))^{1-\epsilon} &\geq (\lambda \times OPT(D'))^{1-\epsilon} \\ &\geq (OPT(D) - n\lambda)^{1-\epsilon} \\ &\geq (\tfrac{1}{2}OPT(D))^{1-\epsilon} \end{aligned}$$

The first inequality holds since $\lambda \geq 1$. The optimum solution S of D on the scaled instance has a weigth greater than $w(S)/\lambda - |S|$, therefore $\lambda OPT(D') \geq OPT(D) - n\lambda$. This yields the second inequality. Finally, the last inequality follows by definition of λ. We conclude that the algorithm described in this subsection is an $OPT^{1-\epsilon}$-approximation for Max-Weighted-Labeled-Path.

6 Collecting all the labels with a minimum number of paths

In this section, we study a problem closely related to Max-Labeled-Path. Given a labeled DAG, the problem MPCL consists in finding a Minimum number of Paths Collecting all the Labels. Using results of previous sections, we prove that MPCL does not belong to APX and describe for any fixed real $\epsilon > 0$, an algorithm that computes in polynomial time a solution of MPCL within a factor $|\mathcal{L}|^{\epsilon}$ of the optimum.

Theorem 8. *Assuming $P \neq NP$, no polynomial time algorithm can achieve a constant performance ratio for* MPCL *even when restricted to instances D containing a path that collects all the labels, i.e such that $OPT(D) = 1$.*

Proof. Suppose, by way of contradiction, that there exists an algorithm ALG with constant performance ratio α for MPCL. Let D be a labeled DAG with a path collecting all the labels, ALG computes in polynomial time at most α paths collecting all the labels of D. One of these paths must collect at least $|\mathcal{L}|/\alpha$ labels. Therefore the existence of a constant factor approximation algorithm for MPCL implies the existence of a constant factor approximation algorithm for Max-Labeled-Path which is impossible unless $P = NP$ by Theorem 5. □

6.1 Computing a solution within $\frac{|\mathcal{L}|^{\epsilon}}{\epsilon}$ of the optimum

Theorem 9. *For any $\epsilon > 0$, there exists an algorithm A such that for any labeled DAG D, A computes in polynomial time a solution of the instance D of* MPCL *within a factor $\frac{|\mathcal{L}(D)|^{\epsilon}}{\epsilon}$ of the optimum.*

Proof. Let δ be the optimal value on D, we construct D' as δ copies of D: $D_1, \ldots, D_{\delta}$ such that for all $i < \delta$, there is an arc from every sink of D_i to every source of D_{i+1}. Therefore there exists a path collecting all the labels in D'. Moreover, from a set of k paths in D' that collects all the labels we can extract a set of $k\delta$ paths in D that collects all the labels. Guessing $\delta \leq |\mathcal{L}(D)|$, D' can be constructed in polynomial time. We will now describe an algorithm on D'. By Theorem 7, for any $\epsilon > 0$, there exists an algorithm A_{ϵ} for Max-Labeled-Path which computes a path collecting $|\mathcal{L}(D')|^{1-\epsilon}$ labels.

The related algorithm A for MPCL is the following: we apply A_ϵ on D' to find a path P_0, then we remove the labels of P_0 from D' and we apply again A_ϵ to find a path P_1, we repeat this process until all labels were collected. The value of the solution returned by the algorithm A is the number of times the algorithm A_ϵ has to be processed on D'.

We define the function $f(x) := (|\mathcal{L}(D')|^\epsilon - \epsilon x)^{1/\epsilon}$ which is positive, convex and decreasing over the real interval $[0, \frac{|\mathcal{L}(D')|^\epsilon}{\epsilon}]$. We will prove by induction on i that, for any positive integer i, $f(i)$ is an upper bound on the number of labels remaining after the ith iteration. Since $f(0) = |\mathcal{L}(D')|$, this property holds for $i = 0$.

Since f is a convex function, we have

$$\begin{aligned} f(x+1) &\geq f(x) + f'(x) \\ &= f(x) - (|\mathcal{L}(D')|^\epsilon - \epsilon x)^{\frac{1}{\epsilon}-1} \\ &= f(x) - f(x)^{1-\epsilon} \end{aligned}$$

By induction, after the ith iteration, it remains $r \leq f(i)$ labels. Since there exists a path collecting all of them, A_ϵ returns a path P with at least $r^{1-\epsilon}$ labels. After removing these labels, it remains at most $r - r^{1-\epsilon} \leq f(i) - f(i)^{1-\epsilon}$ labels in D' because the function $x - x^{1-\epsilon}$ is increasing and $r \leq f(i)$. Therefore, $f(i+1)$ is indeed an upper bound on the number of labels remaining after the $(i+1)$th iteration.
Since $f(\frac{|\mathcal{L}(D')|^\epsilon}{\epsilon}) = 0$, there is no label left after $\lceil \frac{|\mathcal{L}(D')|^\epsilon}{\epsilon} \rceil$ steps, i.e. the number of paths returned by A on D' is at most $\frac{|\mathcal{L}(D')|^\epsilon}{\epsilon}$. We can therefore compute in polynomial time a solution on D of value at most $\frac{|\mathcal{L}(D)|^\epsilon}{\epsilon}\delta$. □

7 Conclusion

In this paper, the APX-hardness of Max-Labeled-Path is established through a simple reduction from MAX-3SAT. Then, using a self-reduction, it is shown that Max-Labeled-Path can not be approximated within a constant ratio unless $P = NP$. In view of these negative results, a $\sqrt{OPT}$-approximation algorithm is given for the weighted version of the problem. Starting with this algorithm and applying a transformation technique, an algorithm with approximation guarantee of $OPT^{1-\epsilon}$ for any $\epsilon > 0$ is obtained. In addition, the paper studies the problem of finding a minimum number of paths collecting all the labels. Using similar techniques it is shown that neither this problem belongs to APX, and a polynomial time algorithm computing a solution with an approximation guarantee of $\frac{\mathcal{L}^\epsilon}{\epsilon}$ for any fixed ϵ is given.

We are interested in the following open questions related to Max-Labeled-Path. A large gap still remains between our algorithm and our hardness results. As a first step to fill this gap, one can ask if it is possible to design $\log(OPT)$ factor approximation algorithm for Max-Labeled-Path. A natural restriction of the problem would be to limit the number of times each label occurs. This question looks already non trivial when a label can be repeated only twice. The problem has several parameter that could be used for fixed parameter tractability. The problem is FPT when the parameter is the number of labels, but the maximum length of a path seems a good parameter to study. Finally, a weakly exponential exact algorithm is an interesting approach for this problem that is quite hard from the approximation point of view.

Acknowledgment. We are grateful to Jérôme Monnot for suggesting the use of a self-reduction to prove the hardness result of Section 3.

References

1. L.M. Batten, *Combinatorics of Finite Geometries,* Cambridge University Press, New York, 1997.
2. H. Broersma and X. Li, Spanning Trees with Many or Few Colors in Edge-Colored Graphs, *Discussiones Mathematicae Graph Theory* **17** (1997) 259–269.
3. H. Broersma, X. Li, G. J. Woeginger, and S. Zhang, Paths and Cycles in Colored Graphs, *Australasian Journal on Combinatorics* **31** (2005) 299–311.
4. T. Brüggemann, J. Monnot, and G. J. Woeginger, Local search for the minimum label spanning tree problem with bounded color classes, *Operations Research Letters* **31** (2003) 195?201.
5. R.-S. Chang and S.-J. Leu, The minimum labeling spanning trees, *Information Processing Letters* **31** (2003), 195–201.
6. B. Couëtoux, L. Gourvès, J. Monnot and O. Telelis, Labeled Traveling Salesman Problems: Complexity and approximation, *Discrete Optimization* **7** (2010) 74-85.
7. R. Hassin, J. Monnot and D. Segev, Approximation Algorithms and Hardness Results for Labeled Connectivity Problems, *Journal of Combinatorial Optimization* **14** (2007) 437–453.
8. R. Hassin, J. Monnot, and D. Segev, The Complexity of Bottleneck Labeled Graph Problems, In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, Lecture Notes in Computer Science, **4769**, Springer-Verlag, Berlin, 2007, 328–340.
9. J. Håstad, Some optimal inapproximability results, *Journal of ACM* **48** (2001) 798–859.
10. S. O. Krumke and H.-C. Wirth, Approximation Algorithms and Hardness Results for Labeled Connectivity Problems, *Information Processing Letters* **66** (1998) 81–85.
11. J. Monnot, The labeled perfect matching in bipartite graphs, *Information Processing Letters* **96** (2005) 81–88.