Olympiads in Informatics, 2013, Vol. 7, 61–77

Algorithmic Results on a Novel Computational Problem

Minko MARKOV[1], Krassimir MANEV[2]
[1]*Department of Computing Systems, Faculty of Mathematics and Informatics*
"St. Kliment Ohridski" University of Sofia
5 J. Bourchier Blvd, P.O. Box 48, BG-1164 Sofia, Bulgaria
[2]*Department of Computer Science, New Bulgarian University*
21 Montevideo St., 1618 Sofia, Bulgaria
e-mail: minkom@fmi.uni-sofia.bg, kmanev@nbu.bg

Abstract. We propose a novel computational problem on undirected graphs, prove that it is $\mathcal{NP}$-complete, and design a linear-time algorithm for the special case when the graph is a cactus. The decision version of the problem is, given a graph and a train in it, the train being an edge sequence that forms a path with one vertex designated as the locomotive, can the train reach a certain target vertex, or not? The movement of the train consists of elementary, single-edge moves with the locomotive forward, the length of the train staying the same. Further, the movement of the train is restricted: the train cannot self-intersect.
Key words: train in a graph, $\mathcal{NP}$-completeness, algorithmic graph theory, cactus graphs.

1. Introduction

Pankov (2008) proposes the following problem, having already mentioned the concept of a train in a graph:

> **Task 2.** A graph is given. Firstly, the head H and the tail T of the train are in two neighbor vertices. Write a program finding one of the shortest ways to be passed by the train (moving forward only) in order to put its head to the primary position of T and its tail to the primary position of H.

Similar problems called PATH TRANSFERABILITY and PATH REVERSIBILITY are introduced and investigated by Torii (2008) though the results are not algorithmic. A vaguely similar concept under the name *snake* of length two or three is used for proving results on symmetric graphs by Tutte (1998).

We consider a modification of that problem, which is explained informally as follows. Imagine a network of train rails of unit length. The network can have arbitrary topology – it does not have to be planar and an arbitrary number of rails can meet at any junction. There is a train in the network, its locomotive being a point and its carriages being of unit length, each carriage being situated on a single rail. One endpoint of the train is the locomotive and the train can only move with the locomotive forward; unlike real trains,

it cannot move backwards. Initially, the locomotive is situated over some junction x of the network. The train travels by discrete moves with the locomotive forward. Each elementary move has unit length, therefore after it the locomotive arrives at some junction y such that x and y are adjacent in the network, i.e., they are at the endpoints of some rail. Junction y must have been unoccupied by the train before the move commences. The length of the train does not change during its travel, so as the locomotive arrives at y, some junction previously occupied by the train becomes free. Given the network and some initial position of the train and some target junction, the question is, can the locomotive reach the target. The corresponding optimization problem is, what is the minimum number of elementary moves in order to get the locomotive to the target.

In the degenerate case when the train has zero carriages and consists of a single point – namely, the locomotive – the problem is equivalent to the SHORTEST PATH problem on unweighted undirected graphs. However, in general that is not the case. Without loss of generality we can assume the corresponding graph is connected and thus the only reason the locomotive cannot start moving towards the target on some shortest path is that the body of the train gets into its way. Superficially, that problem is a minimization one. However, if the body of the train blocks every path between the locomotive and the target, we must find a long enough path inside the network so that the whole train can be "stored" in it, thus freeing a path for the locomotive to go to the target. It is well-known the LONGEST PATH problem is $\mathcal{NP}$-complete on general graphs (Garey and Johnson, 1979) and that is a strong indication that our problem is intractable, too. Indeed, we prove its $\mathcal{NP}$-completeness by a reduction from HAMILTON PATH.

However, on some restricted graph classes the problem is solvable in linear time. There is an obvious trivial algorithm for trees. We propose a linear-time algorithm for the said problem on cactus graphs.

2. Background

2.1. *Graphs*

We consider undirected graphs without multiple edges or self loops. Let $G = (V, E)$ be a graph. To *delete* a vertex u from G means to transform G into $G' = (V \setminus u, E \setminus \{e \in E \mid u \in e\})$. We write $G' = G - u$. If the vertex set of a graph G is not named explicitly we denote it by $V(G)$. Likewise, $E(G)$ is the edge set. To *remove* an edge (u, v) from G means to transform G into $G' = (V, E \setminus \{(u, v)\})$. The result of the edge removal operation is denoted by $G - e$. For any vertex $u \in V(G)$, by $N(u)$ we denote the set of all vertices adjacent to u.

A path in G is a sequence $p = u_0, e_0, u_1, e_1 \dots e_{n-1}, u_n$, for some $n \geqslant 1$, of alternating vertices $u_0, u_1 \dots u_n$ and edges $e_0, e_1 \dots e_{n-1}$ such that for $0 \leqslant i < n$, $e_i = (u_i, u_{i+1})$. u_0 and u_n are called *the endpoints of* p, and the remaining vertices are *the internal vertices of* p. *The length of* p, denoted by $|p|$, is n. The set of the vertices of p is denoted by $V(p)$.

If all vertices in a path are distinct, the path is *simple*. When we say *general path* we mean a path that is not necessarily simple. If p is a path such that the only repeating elements are $u_0 = u_n$ and $n \geqslant 4$, we say p is a *cycle*. *The length* of a path or a cycle z is the number of edges in it and is denoted by $|z|$. For any two vertices $u, v \in V(G)$, *the distance between* u *and* v, denoted by $\mathrm{dist}(u, v)$ is the length of a shortest path with endpoints u and v in G. If there is no path between u and v then we define that $\mathrm{dist}(u, v) = \infty$. For any vertex u and cycle s, the distance between u and s is the length of a shortest path connecting u to a vertex from s; we denote that by $\mathrm{dist}(u, s)$.

When we consider a path or cycle we actually think of the subgraph that has those vertices and edges, rather than some concrete description such as $u_1, e_1, u_2, e_2 \ldots e_{n-1}, u_n$. A path and its description are conceptually different objects, and likewise with cycles. Any simple path of length $\geqslant 1$ has two different descriptions and any cycle of $n \geqslant 3$ vertices has $2n$ different descriptions. Let us think of cycles as subgraphs. *An arc* in a cycle s is any path p such that the vertices and edges of p form a contiguous sequence in some description of s.

Since we do not consider multi-graphs, we define paths and cycles by listing only the vertices and not the edges.

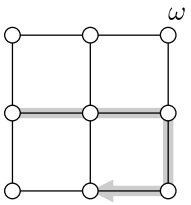
2.2. *Trains in Graphs*

A train in a graph G is a simple path $p = x_0, x_1 \ldots x_q$ in G. There is a direction associated with the train: the leftmost vertex x_0 in its description is considered its first vertex and we call it, *the locomotive*. The vertices of G are partitioned into *occupied* and *free*, the former ones being precisely the vertices that belong to the train. An elementary move of the train is transforming it into another path $p' = x', x_0, x_1 \ldots x_{q-1}$ where x' is a vertex such that $x' \in N(x_0)$ and x' is free before the move. The move is considered instantaneous. After it, vertex x_q is free and x' is occupied. *The target* is some vertex $\omega \in V(G)$ that may or may not be free initially. There is precisely one train in any graph we consider.

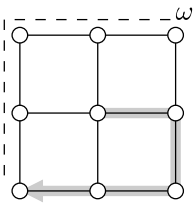
So far we have defined "train" as a temporary object being identified with its current position – after any move, the train is a different object. However, it is helpful to think of the train as permanent object whose current position is just one of its attributes.

The following two definitions are relative to some concrete position $x_0 \ldots x_q$ of the train. For any $\beta \in V(G)$, *a* β*-trajectory* is any general path $p = x_0, x_1 \ldots \beta$ such that the locomotive can reach β by moving along p. For any free vertex v, *a* v*-free path* is any simple path p with one endpoint the locomotive and the other endpoint v such that all its vertices at that moment, except for the locomotive, are free. A zero-length path in which the locomotive coincides with v is considered v-free, too. When we say simply *a free path* we mean an ω-free path; of course, ω must be a free vertex, unless it coincides with the locomotive.

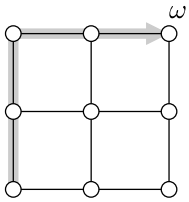
The goal is to compute whether a ω-trajectory exists, in the decision version of the problem, or to compute an ω-trajectory of minimum length, in the optimization version. It is important to realize that initially there can be no free path and yet there can exist an ω-trajectory. If a free path exists initially the answer to the decision problem is trivially



(a) The initial position. The locomotive is where the arrow tip is. All paths between the locomotive and ω are blocked by the train itself.



(b) After the first move there is a free path – see the dashed line.



(c) The goal is achieved. Now the locomotive is at ω, having moved along the free path from Figure 1(b).

Fig. 1. There is an ω-trajectory although initially the train blocks all its possible paths to ω.

YES. Figure 1(a) depicts a graph and a train in it. The train is drawn with a thick gray line and the locomotive is where the arrow tip is. Initially, there is no free path. However, the train can get into the position shown on Fig. 1(b). Now there is a free path outlined by a dashed line. Clearly, the locomotive can reach the target along the free path (Fig. 1(c)).

Note that if the length of the train is zero, i.e., if the locomotive consists of a single vertex-locomotive, the problem becomes the SHORTEST PATH problem, which is a very well-studied problem. Therefore, we assume the train is always longer than zero.

2.3. *Cactus Graphs*

A cactus graph, shortly cactus, is a connected graph G in which every edge is in at most one cycle. The edges of G are partitioned into *tree edges* and *cycle edges*, the former being the ones that are in zero cycles. For any cycle s in G, *the constituents of* s are the connected components of G left after the removal of all edges of s. For each vertex $u \in s$, the u-constituent of s is the constituent that u belongs to.

Let v be any vertex of G. Let $G_1, \ldots, G_t$ be the connected components of $G - v$. Let $v_1, \ldots, v_t$ be new vertices. We are going to use each of them as a replacement of v in precisely one of $G_1, \ldots, G_t$. For each G_i, $1 \leqslant i \leqslant t$, let G'_i be G_i plus vertex v_i plus the one or two edges that used to connect v to vertices from G_i; now these edges connect v_i to those one or two vertices. Finally, rename v_i to v in all G'_i. The renaming does not mean we identify all v_i vertices. It means we get t connected graphs, each having a vertex with name v. We say that $G'_1, \ldots, G'_t$ are *the fragments of* G *relative to* v.

3. Intractability Results

Let us recall the formal definitions of two computational problems first: the newly introduced TRAIN IN A GRAPH in its decision version and a version of the the classical HAMILTONIAN PATH (Garey and Johnson, 1979, pp. 60).

Computational Problem TRAIN IN A GRAPH
Generic Instance: Undirected graph G, a train in it, a target vertex ω
Question: Is there an ω-trajectory in G? □

Computational Problem HAMILTONIAN PATH BETWEEN TWO POINTS
Generic Instance: Undirected graph G, vertices u and v in G
Question: Is there a Hamiltonian Path in G with endpoints u and v? □

For brevity we call the latter problem simply HAMILTONIAN PATH. It is $\mathcal{NP}$-complete just as the common version of the problem (Garey and Johnson, 1979, pp. 60).

Theorem 1. TRAIN IN A GRAPH *is* $\mathcal{NP}$*-hard.*

Proof. The notation "$\propto$" stands for "Karp-reduces to". We prove that HAMILTON PATH $\propto$ TRAIN IN A GRAPH. Assume G, u, v is an instance of HAMILTON PATH. Let $|V(G)|$ be n. Let $Z = \{x, z_0, z_1 \dots z_{n-1}, \omega\}$ be a vertex set such that $Z \cap V(G) = \emptyset$. Let $G' = (V \cup Z, E')$, where $E' = E \cup \{(z_0, z_1), (z_1, z_2) \dots (z_{n-3}, z_{n-2}), (z_{n-2}, v), (v, x), (x, u), (v, \omega)\}$. Let $x, v, z_{n-2}, z_{n-3} \dots z_1, z_0$ be a train in G' with x as the locomotive. The length of the train is $n - 1$. Let ω be the target in G'. We constructed an instance of TRAIN IN A GRAPH. Figure 2 illustrates our construction. The original graph G is drawn with solid lines while the added vertices are black and the added edges are drawn with dotted lines.

We claim there is a Hamilton path with endpoints u and v in G iff there is an ω-trajectory in G'. Assume there is a Hamilton path p in G with endpoints u and v. As $|V(G)|$ is n, $|p| = n - 1$. Of course, initially there is no free path in G' but let the train move along p until the locomotive is at some vertex $w \in N(v)$. Since the train's length

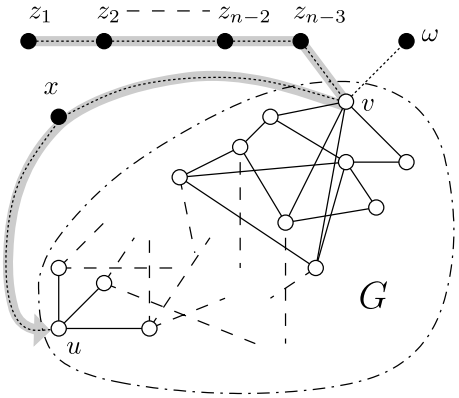


Fig. 2. The construction for the Karp reduction in Theorem 1. The original graph is G and the vertices u and v are in it. To G we add the vertices $z_0, \dots, z_{n-2}$, x, and ω, plus the dotted edges, obtaining G'. In G' we construct the train indicated by the thick gray line.

is $n-1$, it is obvious at that moment the other endpoint of the train is at x and thus v is a free vertex. With two elementary moves the locomotive reaches ω.

Now assume there is a way for the train to reach ω, starting at the described initial position. Clearly, the ultimate move for the locomotive is from v to ω and the penultimate one, from some vertex $w' \in N(v)$ to v. In order the train to make that penultimate move, it must be the case that v is free. However, in order v to be free at that moment, at the initial moment there has to be a simple path p' of length $\geqslant n-2$ in G; furthermore, v cannot be in p'. Obviously p' is a Hamilton path between u and some vertex $w' \in N(v)$ in $G-v$. It follows immediately that p', v is a Hamilton path in G. □

Theorem 2. TRAIN IN A GRAPH *is in* $\mathcal{NP}$.

Proof. Unlike plenty of other $\mathcal{NP}$-completeness proofs, the fact that this problem is in $\mathcal{NP}$ is not obvious. A natural choice of certificate is an ω-trajectory. However, the trajectory is a general path so it is not immediately obvious its length is polynomial in the input size.

We prove that for every YES-instance of TRAIN IN A GRAPH, there exists an ω-trajectory such that no vertex appears more than twice in it. Assume we are given an YES-instance of TRAIN IN A GRAPH, the graph being called G and the train, $x_0, x_2, \ldots, x_q$ with the locomotive at x_0. Let $X = \{x_0, x_1 \ldots x_q\}$. If there exists a free path initially, the claim is obviously true. Assume there is no free path initially. However, since this is a YES-instance, there is an ω-trajectory p that the locomotive moves along. If all vertices of p are unique, we are done with the proof. Assume there are repeating vertices in p. Think of p as a string of vertex names with x_0 at the left end. Let the leftmost repeating vertex in that string be α. That vertex α is the first vertex that the locomotive "sees" for a second time as it moves along p.

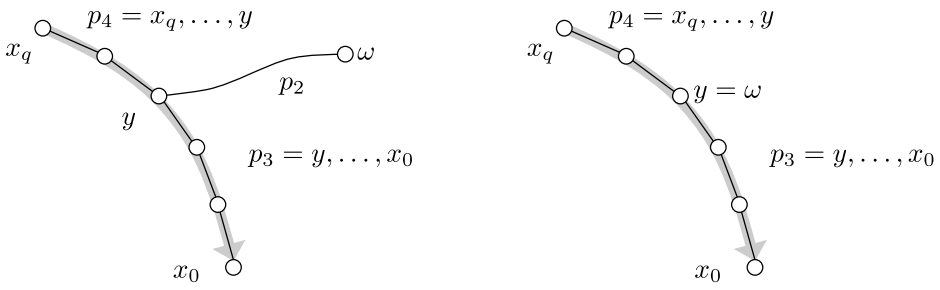
Now consider the moves of the train from its initial position until the locomotive hits α for the first time, moving along p. If at any moment during that sequence of moves there appears a free path, let the locomotive abandon the movement along p and instead follow the free path to ω. In this case there are no repeating vertices in the overall path the locomotive follows from its initial position to α.

Otherwise, consider the moment the locomotive gets to α for the first time. Clearly, the sub-path of p between the first and the second appearance of α inclusive forms a simple cycle s. Let the sub-path of p between x_0 and the first α exclusive be called p_1:

$$p = \underbrace{x_0 \ldots\ldots\ldots\ldots}_{p_1} \underbrace{\alpha \underbrace{\ldots\ldots\ldots\ldots}_{\text{no}\alpha\text{here}} \alpha}_{s} \ldots\ldots\ldots\ldots$$

Note that s and p_1 are vertex-disjoint because by construction α is the first repeated vertex during the movement of the train.

Relative to the initial moment, let y be the following vertex. If $\omega \notin X \setminus \{x_0\}$ then y is any vertex from X such that at the initial moment there is a path p_2 between ω and y that consists of free vertices, except for y. As G is connected, such an y exists. If



(a) ω is not train vertex. p_2 is a path connecting ω and the train.

(b) ω is a train vertex and p_2 is the single-vertex path ω.

Fig. 3. The two possible placements of ω and the initial train, the latter drawn with a thick gray line. p_3 is the subpath of the initial train between y and the locomotive. p_4 is the subpath of the initial train between y and x_q.

$\omega \in X \setminus \{x_0\}$ then $y = \omega$. Note that it makes no sense to consider the case $\omega = x_0$. The two said possibilities for y are shown on Fig. 3. Let p_3 be the subpath $y \ldots x_0$ of the the initial train. If ω is not a train vertex (Fig. 3(a)) then p_3 can be as small as the single vertex x_0. Otherwise (Fig. 3(b)), p_3 cannot coincide with x_0. Let p_4 is the subpath $x_q \ldots y$ of the initial train. Clearly, $V(p_3) \cup V(p_4) = X$.

To complete the proof, first assume $V(s) \cap (V(p_2) \cup X) = \emptyset$ (see Fig. 4(a)). Let the train move along p until the locomotive reaches α for the second time (Fig. 4(b)). At this moment, the path that consists of p_1, p_3, and p_2, in that order, is now free so the locomotive can reach ω along it.

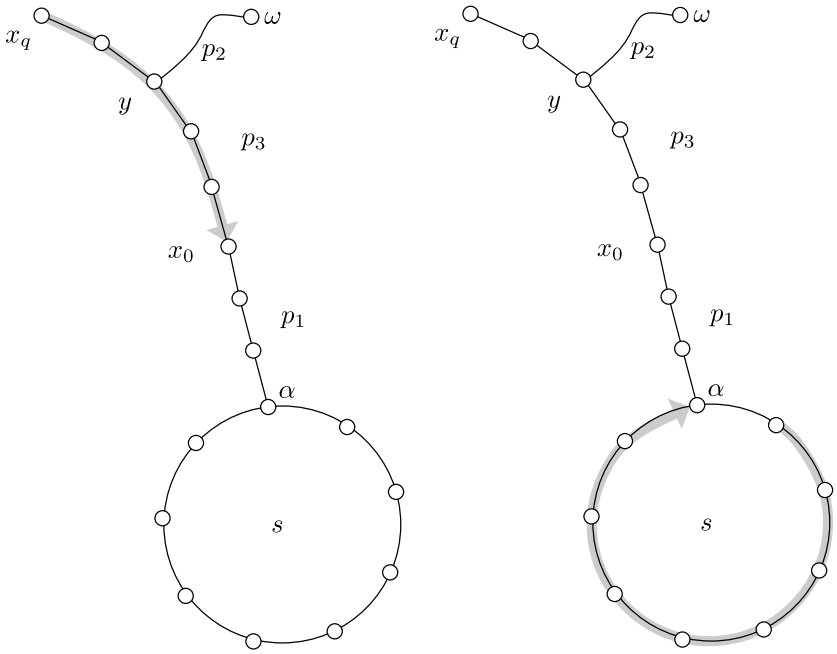
Now assume $V(s) \cap (V(p_2) \cup X) \neq \emptyset$. That overlap can be complicated as illustrated by Fig. 5(a). Whatever the overlap pattern is, clearly there exists a vertex $z \in V(s) \cap (V(p_2) \cup X)$ such that there are no vertices from s that are between z and ω *along the path union of* p_2, p_3 *and* p_4. On Fig. 5(a) the said overlap is only in p_3 and p_4 and vertex z is in p_4. The moment when the locomotive reaches z (see Fig. 5(b)) there is a free path. So the train abandons p and starts moving along that free path towards ω. □

COROLLARY 1. TRAIN IN A GRAPH is $\mathcal{NP}$-complete. □

4. A Linear-Time Algorithm on Cactus Graphs

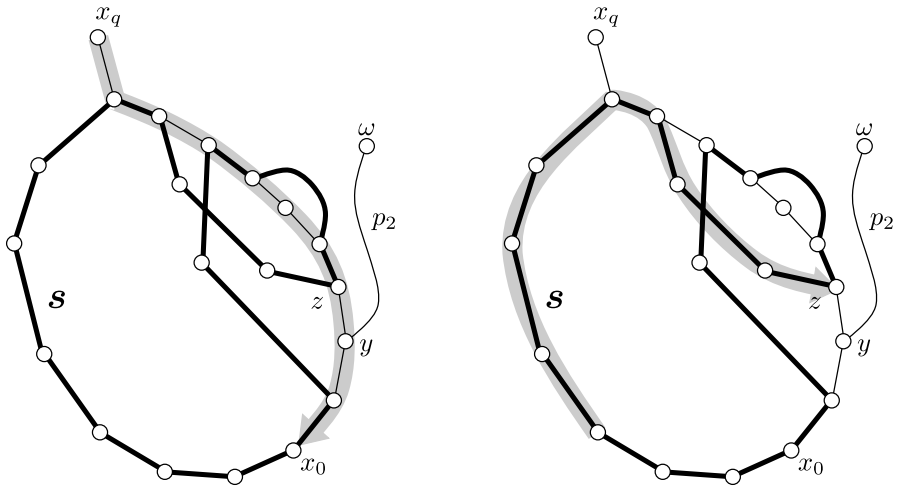
4.1. *The Precomputing and the Algorithm*

Although TRAIN IN A GRAPH is $\mathcal{NP}$-complete, it is still solvable by efficient algorithms on restricted graph classes. The solution on trees is trivial: root the tree at the vertex where initially the locomotive is and use any algorithm for tree traversal to find out if ω and the body of the train are in the same subtree relative to the root. If yes, the answer to the decision version is NO, else it is YES. In the latter case, the answer to the optimization problem is the length of the unique path between the root and ω.



(a) The initial position of the train. (b) The train has just made the U-turn.

Fig. 4. If s is disjoint with both p_2 and X, the train can use s to make a U-turn. After the U-turn there is an obvious free path.



(a) The initial position of the train.

(b) The locomotive, moving along s, reaches z. Now there is an obvious free path.

Fig. 5. s has common vertices with $V(p_2) \cup X$. s is drawn with thick black line.

Now assume the graph is a cactus graph. The cycles that are long enough so that the train can make a U-turn are of special importance. Informally speaking, the train can make a U-turn using cycle s iff it can enter s via some vertex $u \in s$, move only along s in either direction and exit the cycle via u. In order to accomplish that, the train's length must be smaller than $|s|$, otherwise u will not be free when the locomotive attempts to make the exit. It is clear that because of the nature of cacti, namely that cycles can have at most a vertex in common, any cycle is either long enough and the train can make the U-turn using it and no other cycles, or the cycle is useless with respect to making a U-turn. The cycle s cannot be of partial help for the U-turn: if the train exits s through a vertex that is not u, it has to leave s completely and make the U-turn in another cycle. It follows that those long enough cycles make the problem on cacti interesting. If there is none of them, the solution is completely analogous to the solution on trees.

The following algorithm solves the optimization version of TRAIN IN A GRAPH on cacti. Let $G = (V, E)$ be a cactus with vertex set $V = \{1, 2 \dots n\}$, a train $x_0, x_1 \dots x_q$, and target ω. *The long cycles* in the cactus are all cycles of length greater than q. Assume the locomotive is at x_0. Let PREPROCESSING be an algorithm that computes the following.

- For every edge $e \in E$ it computes whether e is a tree edge or a cycle edge.
- The array $D[1 \dots n]$ such that $\forall v \in V : \mathrm{dist}(x_0, v)$.
- In case that $q > 0$, a boolean value S set to FALSE iff ω, on the one hand, and $x_1 \dots x_q$, on the other hand, are in the same fragment of G relative to x_0.
- In case that $q > 0$ and (x_0, x_1) is a tree edge, assuming G' is the connected component of $G - (x_0, x_1)$ that contains x_0, the value

$$\delta = \min \{2 \times \mathrm{dist}(x_0, c) + |c| \mid c \text{ is a long cycle in } G'\}.$$

  If there are no long cycles in G', $\delta = \infty$.
- In case that $q > 0$ and (x_0, x_1) is a cycle edge,
  - the cycle s that (x_0, x_1) is in,
  - $|s|$,
  - the function $\psi(\,)$

$$\psi(u) = \min \{2 \times \mathrm{dist}(u, c) + |c| \mid c \text{ is a long cycle in } H_u\}$$

    for every vertex $u \in s$. H_u is the u-constituent of s. If there are no long cycles in some H_u then $\psi(u) = \infty$.
  - The set U of vertices from s that are not train vertices, plus x_0.
  - The vertex α from s such that ω is in the α-constituent of s. A boolean variable T is set to TRUE iff α is a train vertex.
  - If T is TRUE, U' is the set of vertices $u \in U$ such that the arc of s with endpoints α and u containing x_0 has length $\leqslant \lceil \frac{s}{2} \rceil - q - 1$.
  - If T is FALSE, U'' is the set of vertices $u \in U$ such that the arc of s with endpoints x_0 and u avoiding x_1 has length $\leqslant \lceil \frac{s}{2} \rceil - q - 1$.

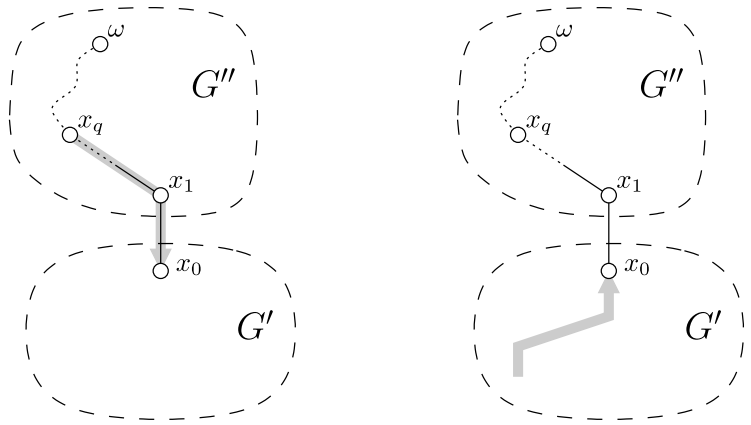
- The length m of the arc of s with endpoints x_0 and α that does not contain x_1.
- $\forall u \in U$, the value $\mathrm{arc}'(u)$: the length of the arc of s with endpoints x_0 and u that avoids vertex x_1.
- $\forall u \in U$, the value $\mathrm{arc}(u)$: the minimum of the lengths of the two arcs of s with endpoints u and α.

Both U' and U'' are defined by differences. If the respective quantity is negative then the set is empty. It is easy to see that PREPROCESSING can be implemented by a modified DFS.

TRAIN IN A CACTUS G: cactus; $x_1 \dots x_q$: train in G; ω: the target
1. PREPROCESSING$(G, (x_0, x_1 \dots x_q), \omega)$
2. **if** $x_0 = \omega$
3. **return** 0
4. **if** $q = 0$ **or** ($q > 0$ **and** S)
5. **return** $D[\omega]$
6. **if** (x_0, x_1) is tree edge
7. **return** $\delta + D[\omega]$
8. **if** T
9. **if** $q \geqslant |s|$
10. $\mathrm{tmp} \leftarrow \min\{\mathrm{arc}'(u) + \psi(u) + \mathrm{arc}(u) \mid u \in U\}$
11. **else**
12. $\mathrm{tmp} \leftarrow \min\{\mathrm{arc}'(u) + \psi(u) + \mathrm{arc}(u) \mid u \in U'\}$
13. $\mathrm{tmp} \leftarrow \min\{\mathrm{m}, \mathrm{tmp}\}$
14. **else**
15. $\mathrm{tmp} \leftarrow \min\{\mathrm{arc}'(u) + \psi(u) + \mathrm{arc}(u) \mid u \in U''\}$
16. $\mathrm{tmp} \leftarrow \min\{\mathrm{m}, \mathrm{tmp}\}$
17. **return** $\mathrm{tmp} + \mathrm{dist}(\alpha, \omega)$

4.2. *Verification*

Assume that PREPROCESSING is correct. If the locomotive is at the target, the length of the shortest trajectory is obviously 0. Line 3 takes case of that case. If the train consists of a single vertex, namely the locomotive, the problem is the same as computing a shortest path in an undirected graph. By assumption, the array $D[1 \dots n]$ is stores the lengths of shortest paths in G with respect to x_0, so $D[\omega]$ is the answer (line 5). If the train is longer than a single vertex but the target is in some fragment H relative to x_0 such that x_1 is not in H (and consequently, $x_2, \dots, x_q$ are not in H), the answer is again $D[\omega]$. To see why, note that any shortest path between x_0 and ω lies solely in H, which follows from the fact that x_0 is a cut vertex; if x_0 was not a cut vertex there would be only one fragment relative to it. By assumption, S is set to TRUE by PREPROCESSING iff the target and x_1 are in different fragments. Therefore, the returned value $D[\omega]$ is the correct answer in this case, too.



(a) The initial train.

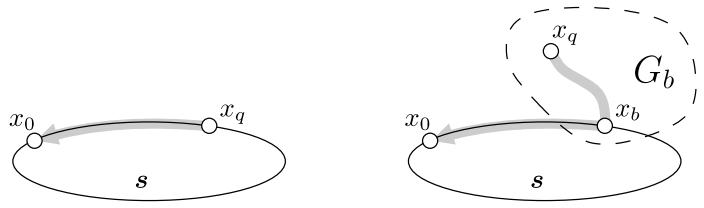
(b) If the train can make a U-turn in G', the locomotive can get back to x_0, this time the body of the train not blocking the paths to ω.

Fig. 6. (x_0, x_1) is a tree edge.

In the remainder of the proof the train is longer than a single vertex and x_1 and ω are in the same fragment relative to x_0. Suppose the train's first edge (x_0, x_1) is a tree edge (see Fig. 6(a)). Let G' be the connected component of G that remains after the removal of (x_0, x_1). In other words, G' consists of all fragments relative to x_0, except for the one containing x_1, being "glued together" at x_0. We argue if there are no long cycles in G' then the locomotive can never reach the target. Indeed, suppose there are no long cycles in G' and consider any possible sequence of moves. If the locomotive traverses a tree edge it only gets further from the target. If it moves along a cycle it still cannot make a U-turn and, if it exits the cycle, it exits via a vertex different from the vertex of entry, again finding itself even further from the target. As G is finite, at some moment either the locomotive will be stuck in a vertex of degree 1 or in a cycle vertex of degree 2 but being blocked by the body of the train. In this case the variable δ is set to ∞ by PREPROCESSING and therefore the returned value (line 7) is correct.

On the other hand, if there is at least one long cycle in G', the locomotive can reach the target. Because of the nature of cactus graphs, the locomotive has to get back to vertex x_0, with its body in G' (see Fig. 6(b)). Now it can reach ω via a free path in G''. The cost of the solution is the length of the trajectory used to reach a long cycle, make the U-turn and get back to x_0, plus the length of the chosen free path from x_0 to ω. A minimum is obtained when both the trajectory and the free path are minimum. The minimum length of a such a free path is the value stored in $D[\omega]$. The minimum length of such a trajectory is the value δ precomputed by PREPROCESSING. It follows the assignment at line 7 is correct in the current case, too.

It remains to consider the subcase when (x_0, x_1) is a cycle edge. Because of the nature of cacti that edge cannot belong to more than one cycle. We call the cycle s. It is obvious that the overlap between the train and s is a contiguous sequence of vertices from s. If G



(a) The whole train in on the cycle.

(b) Only some initial part of the train is on the cycle.

Fig. 7. (x_0, x_1) is a cycle edge.

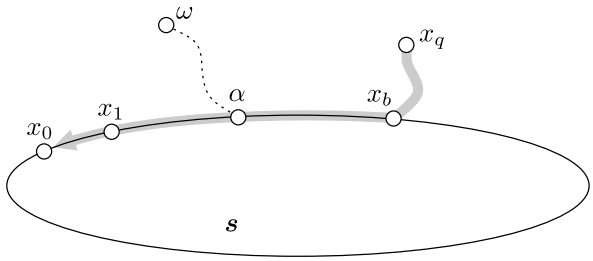


Fig. 8. α is a train vertex. The body of the train at the initial moment blocks all paths between x_0 and ω.

were a general graph that would not be necessarily true but it is true for cacti: either x_q is a vertex from s in which case the whole train lies over an arc of s (Fig. 7(a)), or there is some intermediate train vertex x_b, $2 \leqslant b \leqslant q-1$, such that all train vertices after x_b are in the x_b-constituent – call it G_b – of s (Fig. 7(b)). We distinguish the following two subcases. Call α the cycle vertex such that ω is in the α-constituent of s. Such an α exists because the graph is connected. So, we distinguish the subcase when α is a train vertex from the subcase when α is not a train vertex.

First assume α is a train vertex. Consider G (Fig. 8). If the train is longer than or equal to the length of the cycle, there is only one way to get the locomotive to ω: use a long cycle inside some u-constituent of s for a U-turn so that to bring the locomotive back to the cycle, namely at vertex u, with the body of the train being in the u-constituent. An additional limitation is that the cycle vertex u must not be one of $x_1, x_2, \ldots, x_b$. Figure 9 shows the train having performed the U-turn and gotten back to u. If such a u does not exist then the algorithm must return ∞. Now note that if the case is the current one, the variable T (line 8) has been set to TRUE on the assumption that PREPROCESSING is correct, and further the condition at line 9 is TRUE. So, the assignment at line 10 takes place: if such a u does not exist, the algorithm assigns ∞ to u and then returns ∞ at line 17. On the other hand, if such a u exist, it is a vertex from U (as defined in the description of PREPROCESSING). The overall trajectory consists four legs, as seen from the locomotive:

- From x_0 to u. There is only one choice for the direction along s. By assumption, $\mathrm{arc}'(u)$ stores the length of this leg.

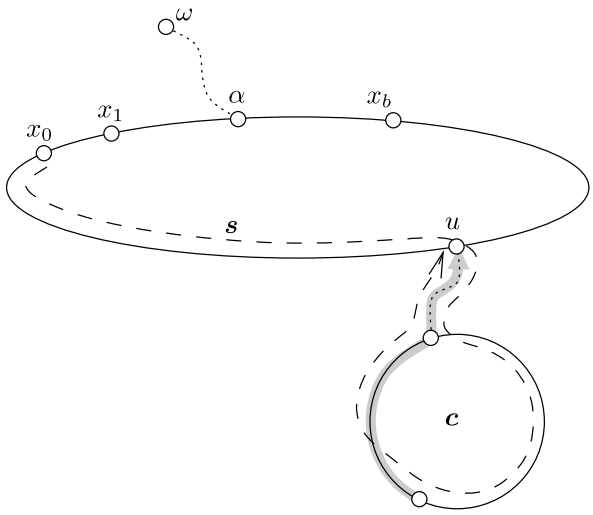


Fig. 9. The train is longer than the cycle but still there is an ω-trajectory. Using some cycle vertex u and a long cycle c in the u-constituent, the train performs a U-turn and the locomotive is again at u – see the dashed line. Now the body of the train does not block the paths between x_0 and ω.

- From u via c, making the U-turn, and back to u. By assumption, that is the value $\psi(u)$.
- From u to α. There are two choices for the direction along x, corresponding to the two arcs of s with endpoint u and α. By assumption, $\mathrm{arc}(u)$ stores the minimum of their lengths.
- From α to ω.

The last leg is independent of the choice of u and its contribution to the length of the overall trajectory is the $\mathrm{dist}(\alpha, \omega)$ term at line 17. The sum of the other three legs depends on the choice of u and therefore seeking the minimum at line 10 is the correct thing to do.

Now suppose that α is a train vertex and the train is shorter than the cycle. In this case the train does not have to perform a U-turn necessarily. If it just goes along the cycle there will be a free path at the moment the locomotive is in $N(x_b)$. It follows that now we always have a solution that is at least $m + \mathrm{dist}(\alpha, \omega)$ (recall the definition of m in the description of PREPROCESSING), as shown on Fig. 10. It is guaranteed that the moment before the locomotive gets at α, vertex α will be free.

However, that is not the only way to reach the goal under the current assumptions. Even if the cycle is long enough relative to the length of the train, it may still be possible to get the locomotive to α by performing a U-turn within some u-component of s (Fig. 11(a)) and going backwards along the cycle to x_1 and then to α (Fig. 11(b)). The overall trajectory may be shorter than the trajectory that the other way round s. Note that if using a U-turn is to be beneficial, after getting to u for the second time, the locomotive should go along s in the opposite direction to the one before – if it goes in the same direction as before, the U-turn maneuver is just a useless detour (Fig. 11(c)).

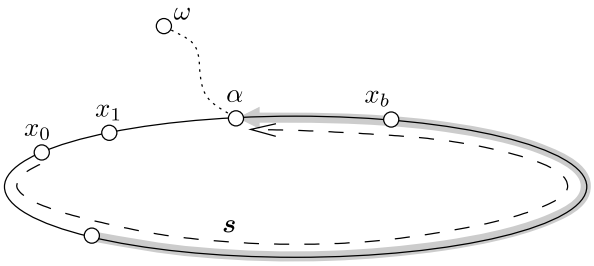
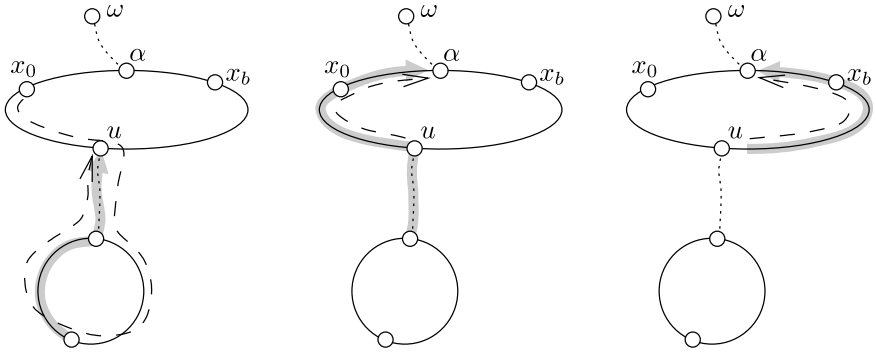


Fig. 10. The train is shorter than the cycle. One way to reach the goal is to slide the locomotive along the cycle until it reaches α – see the dashed line – and then perform the obvious thing.



(a) Having made the U-turn and gotten back to u.

(b) Going backwards to x_0 and then to α may yield a shorter way than the one shown on Figure 10.

(c) However, going forward from u to α is overall always a worse way than the one shown on Figure 10.

Fig. 11. The train is shorter than the cycle. Another way to reach the goal is to make a U-turn inside some u-constituent.

Now consider the possible locations of that vertex u in s. Let us fix a direction on s such that x_0, α, and x_1 appear in that order in that direction. On all our figures that direction is counter-clockwise. Let *the antipodal vertex of* α be the vertex in s that is at distance $\left\lceil \frac{s}{2} \right\rceil$ away from α in the said direction. Informally speaking, it makes no sense to choose vertex u past the antipodal vertex because if the locomotive gets to the antipodal vertex, the best thing to do is to let it slide in the same direction along the cycle until it hits α. Furthermore, we can improve the maximum distance from α that the potential vertex u can be. Since a U-turn inside the u-constituent costs at least $q + 1$ moves, the u vertex should not be further away from α (in the said direction) than $\left\lceil \frac{s}{2} \right\rceil - q - 1$. Indeed, the definition of U' in the description of PREPROCESSING provides that.

Note that the u vertex cannot possibly be past x_b (in the said direction). Assume the opposite: u is past x_b but before α as shown on the figure on the left. Let k, l and t be the distances shown there. Obviously, $k+l+t = |s|$. In order the U-turn in the u-constituent to make sense, it must be the case that $k + q + 1 + k + t < k + l$, because it has to

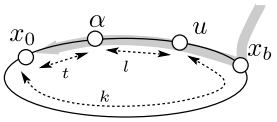


Fig. 12. Vertex u cannot be past vertex x_b in the counter-clockwise direction from x_0.

more beneficial the train to go to u in the said direction (cost k), make the U-turn (at least $q+1$), go back to x_1 in the opposite direction (cost k) and then go to α still in the opposite direction (cost t), rather than go in the said direction from x_1 to α (cost $k+l$). But

$$\begin{aligned} k+q+1+k+t < k+l &\leftrightarrow k+q+1+t < l \leftrightarrow k+q+1+t+l < 2l \\ &\leftrightarrow |s|+q+1 < 2l \end{aligned}$$

and it is impossible to be the case that $|s|+q+1 < 2l$ because both $|s| > l$ and $q > l$; $q > l$ because $q > |s|$. It follows that u cannot be a vertex outside U, thus the definition of U' as a subset of U is correct.

It remains to consider only one more subcase: α is not a train vertex. In this case the relevant part of the algorithm is lines 15 and 16 because by assumption, the variable T is FALSE. The locomotive can always reach ω in this case because there is an obvious solution with cost $m + \mathrm{dist}(\alpha, \omega)$. Note that the assignments at lines 16 and 17 provide the output is at most $m + \mathrm{dist}(\alpha, \omega)$. That value correspond to a solution in which the locomotive slides directly from x_0 to α along s. However, there may be a better way to reach α. The locomotive can reach α from the other direction. To accomplish that, the train has to make a U-turn using some long cycle c inside some u-constituent of s first. That maneuver makes sense only if u is one of the vertices from U''; for any vertex from s past U'' (in the direction away from x_0, avoiding x_1) it is better not to try a U-turn even if there is a long cycle in the u-constituent. And indeed, the code at line 15 ties to find a suitable U-turn only in the constituents of vertices from U''. The addition $\mathrm{arc}'(u) + \psi(u) + \mathrm{arc}(u)$ precisely corresponds to going from x_0 to u (in the appropriate direction), making the U-turn and going back from u via x_0 to α. Because u is a vertex from U'', the shorter arc between u and α is the one containing x_1. Figure 13 illustrates the possibility we just discussed.

That concludes the proof of correctness.

4.3. *Time Complexity Analysis*

Let G be the cactus we run our algorithm on with $n = |V(G)|$ and $m = |E(G)|$. Clearly, $m = \Theta(n)$. PREPROCESSING runs in time $\Theta(n+m)$ because that is the running time of DFS in general and PREPROCESSING can be implemented as a modified DFS, keeping the running time in the same asymptotic bound. For cacti, $\Theta(n+m)$ is $\Theta(n)$. So, PREPROCESSING takes $\Theta(n)$ time. The three minima at lines 10, 11, and 15 take $\Theta(n)$ time at worst, and the other computations are constant-time. It follows TRAIN IN A CACTUS runs in time $\Theta(n)$.

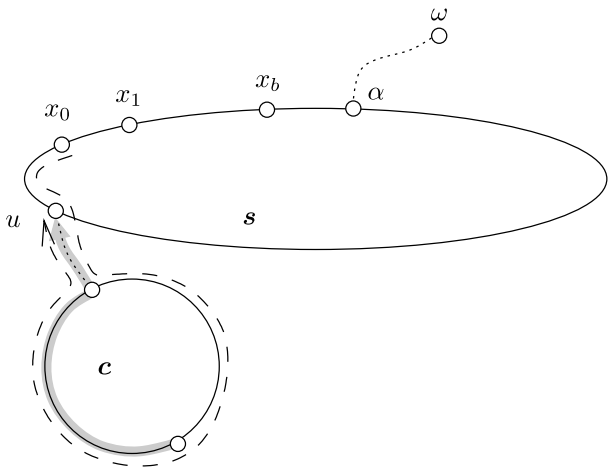


Fig. 13. Although α is not a train vertex initially it makes sense to perform a U-turn and then reach α "backwards" via the shorter cycle arch containing x_0 rather then simply follow the cycle forward to α.

5. Conclusions

We have constructed a linear-time algorithm for novel computational graph problem on a restricted graph class, the problem being $\mathcal{NP}$-complete in general. Our algorithm is relatively straightforward and easy to grasp and implement, which makes it practical.

There are numerous possibilities for future research stemming from this one. Since cacti are outerplanars, one may try to develop fast algorithms for the same problem on more general graph classes, for example outerplanars or even on partial 2-trees. Another way to generalize this result is to modify the rules of the train movement, for example letting the train gain or lose length or move bidirectionally in some circumstances.

References

Garey, M., Johnson, D. (1979). *Computers and Intractability*. W.H. Freeman and Co.

Pankov, P. (2008). Naturalness naturalness in tasks for olympiads in informatics. *Olympiads in Informatics*, 2, 115–121.

Torii, R. (2008). Path transferability of graphs. *Discrete Mathematics*, 308(17), 3782–3804.

Tutte, W.T. (1998). *Graph Theory As I Have Known It*. Oxford University Press, Inc.

M. Markov is an assistant professor of discrete mathematics and algorithms at Sofa University, Sofia. Bulgaria, PhD in computer science.

K. Manev is a professor of discrete mathematics and algorithms at New Bulgarian University, Sofia, Bulgaria, PhD in computer science. He was a member of Bulgarian National Committee for olympiads in informatics since 1982 and president of NC from 1998 to 2002. He was member of the organizing team of IOI 1989, IOI 1990, vice president of IOI 2009 and a leader of Bulgarian team for IOI in 1989, 1998, 1999, 2000 and 2005. From 2000 to 2003 he was elected member of IC of IOI, since 2005 to 2010 represented in IC the Host country of IOI 2009, and during IOI 2010 was elected as a member of IC of IOI again for the period 2010–2013.