On the Shortest Path Game [☆]

Andreas Darmann

Institute of Public Economics, University of Graz,
Universitaetsstr. 15, 8010 Graz, Austria,
andreas.darmann@uni-graz.at

Ulrich Pferschy, Joachim Schauer

Department of Statistics and Operations Research, University of Graz,
Universitaetsstr. 15, 8010 Graz, Austria,
{pferschy, joachim.schauer}@uni-graz.at

Abstract

In this work we address a game theoretic variant of the shortest path problem, in which two decision makers (agents/players) move together along the edges of a graph from a given starting vertex to a given destination. The two players take turns in deciding in each vertex which edge to traverse next. The decider in each vertex also has to pay the cost of the chosen edge. We want to determine the path where each player minimizes its costs taking into account that also the other player acts in a selfish and rational way. Such a solution is a subgame perfect equilibrium and can be determined by backward induction in the game tree of the associated finite game in extensive form.

We show that finding such a path is PSPACE-complete even for bipartite graphs both for the directed and the undirected version. The latter result is a surprising deviation from the complexity status of the closely related game Geography.

On the other hand, we can give polynomial time algorithms for directed acyclic graphs and for cactus graphs even in the undirected case. The latter is based on a decomposition of the graph into components and their resolution by a number of fairly involved dynamic programming arrays. Finally, we give some arguments about closing the gap of the complexity status for graphs of bounded treewidth.

Keywords: shortest path problem, game theory, computational complexity, cactus graph

[☆]Andreas Darmann was supported by the Austrian Science Fund (FWF): [P 23724-G11]. Ulrich Pferschy and Joachim Schauer were supported by the Austrian Science Fund (FWF): [P 23829-N13].

1. Introduction

We are given a directed graph $G = (V, A)$ with vertex set V and arc set A with positive costs $c(u, v)$ for each arc $(u, v) \in A$ and two designated vertices $s, t \in V$. The aim of Shortest Path Game is to find a directed path from s to t in the following setting: The game is played by two players (or agents) A and B who start in s and always move together along arcs of the graph. In each vertex the players take turns to select the next vertex to be visited among all neighboring vertices of the current vertex with player A taking the first decision in s. The player deciding in the current vertex also has to pay the cost of the chosen arc. Each player wants to minimize the total arc costs it has to pay. The game continues until the players reach the destination vertex t.[1] Later, we will also consider the same problem on an undirected graph $G = (V, E)$ with edge set E which is quite different in several aspects.

Even in a connected graph, it is easy to see that the players may get stuck at some point and reach a vertex where the current player has no emanating arc to select. To avoid such a deadlock, we restrict the players in every decision to choose an arc (or edge, in the undirected case) which still permits a feasible path from the current vertex to the destination t (which is computationally easy to check).

> **(R1)** No player can select an arc which does not permit a path to vertex t.

In the undirected case, it could be argued that for a connected graph this restriction is not needed since the players can always leave dead ends again by going back the path they came. However, this would imply cycles of even length which we will rule out as described further below.

In classical game theory the above scenario can be seen as a finite game in extensive form. All feasible decisions for the players can be represented in a game tree, where each node corresponds to the decision of a certain player in a vertex of the graph G.

The standard procedure to determine equilibria in a game tree is *backward induction* (see Osborne [11, ch. 5]). This means that for each node in the game tree, whose child nodes are all leaves, the associated player can reach a decision by simply choosing the best of all child nodes w.r.t. their allocated total cost, i.e. the cost of the corresponding path in G attributed to the current player. Then these leaf nodes can be deleted and the pair of costs of the chosen leaf is moved to its parent node. In this way, we can move upwards in the game tree towards the root and settle all decisions along the way.

[1] We will always assume that a path from s to t exists.

This backward induction procedure implies a strategy for each player, i.e. a rule specifying for each node of the game tree associated with this player which arc to select in the corresponding vertex of G: *Always choose the arc according to the result of backward induction.* Such a strategy for both players is a *Nash equilibrium* and also a so-called *subgame perfect equilibrium* (a slightly stronger property), since the decisions made in the underlying backward induction procedure are also optimal for every subtree.[2]

The outcome, if both players follow this strategy, is a unique path from s to t in G corresponding to the unique **s**ubgame **p**erfect **e**quilibrium (SPE) which we will call ***spe*-path**. A *spe*-path for Shortest Path Game is the particular solution in the game tree with minimal cost for both selfish players under the assumption that they have complete and perfect information of the game and know that the opponent will also strive for its own selfish optimal value.

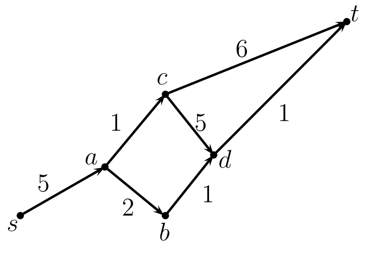
Clearly, such a *spe*-path path can be computed in exponential time by exploring the full game tree. It is the main goal of this paper to study the complexity status of finding this *spe*-path. In particular, we want to establish the hardness of computation for general graphs and identify special graph classes where a *spe*-path can be found without exploring the exponential size game tree.

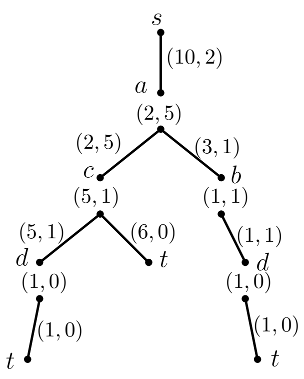
An illustration is given in Example 1. Note that in general game theory one considers only outcomes of strategies and their payoffs, i.e. costs of paths from s to t in our scenario. In this paper we will consider in each node of the game tree the cost for each player for moving from the corresponding vertex v of G towards t, since the cost of the path from s to v does not influence the decision in v. This allows us to solve identical subtrees that appear in multiple places of the game tree only once and use the resulting optimal subpath on all positions.

Example 1. *Consider the following graph and the associated game tree. The* spe*-path is determined by backward induction and represented by ordered pairs of cost values* (x, y) *meaning that the decider in a given vertex has to pay a total value of* x *whereas the opponent has to pay a value of* y.

Note that if the game would be played cooperatively the shortest path of value 9 *would give a lower total cost than the optimal solution of* $(10, 2)$ *in our case.*

[2]In order to guarantee a unique solution of such a game and thus a specific subgame perfect Nash equilibrium, we have to define a tie-breaking rule. We will use the "optimistic case", where in case of indifference a player chooses the option with lowest possible cost for the other player. If both players have the same cost, the corresponding paths in the graph are completely equivalent. Assigning arbitrary but fixed numbers to each vertex in the beginning, e.g. $1, \dots, n$, we choose the path to a vertex with lowest vertex number.

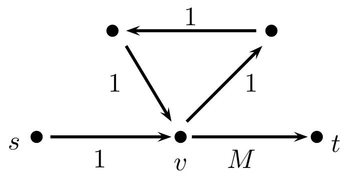




In this setting finding the *spe*-path for the two players is not an optimization problem as dealt with in combinatorial optimization but rather the identification of two sequences of decisions for the two players fulfilling a certain property in the game tree.

Note that there is a conceptual problem in this finite game model. There may occur cases where the game is infinite as illustrated by the following example.

Example 2. *Consider the graph depicted below. Player B has to decide in vertex v whether to pay the cost $M \gg 2$ or enter the cycle of length 3. In the latter case, the players move along the cycle and then A has to decide in v with the same two options as before for player B. In order to avoid paying M both players may choose to enter the cycle whenever it is their turn to decide in v leading to an infinite game.*



Since there is no concept in game theory to construct an equilibrium for an infinite game in extensive form (only bargaining models such as the Rubinstein bargaining game are well studied, cf. Osborne [11, ch. 16]) we have to impose reasonable conditions to guarantee finiteness of the game. An obvious restriction would be to rule out any cycle by requiring that each vertex may be visited at most once. Indeed, a cycle of even length can not be seen as a reasonable choice for any player since it only increases the total cost of both players. Thus, a *spe*-path in a finite game will never contain a cycle of even length. However, in the above example it would be perfectly reasonable for B to enter the cycle of *odd* length and thus switch the role of the decider in v. Therefore, a second visit of a vertex may well make sense. However, if also A enters the cycle in the next visit of v, two rounds through the odd cycle constitute a cycle of even length which we rejected before. Based on these arguments we will impose the following restriction:

> **(R2)** The players can not select an arc which implies necessarily a cycle of even length.

Note that (R2) implies that an odd cycle may be part of the solution path, but it may not be traversed twice (thus reversing the switching between the players) since this would constitute a cycle of even length. In the remainder of the paper we use "cycle" for any closed walk, also if vertices are visited multiple times. It also follows that each player can decide in each vertex at most once and any arc can be used at most once by each agent.

A different approach to guarantee finiteness of the game would be to impose an upper bound on the total cost accrued by each player. This may seem relevant especially for answering the decision version where we ask whether the costs of the *spe*-path for both players remain below given bounds (see Section 2). Clearly, such an upper bound limits the height of the decision tree and permits a clear answer to problems such as Example 2. In that case, the precise value of the bound defines which of the two players has to pay M after a possibly large number of rounds through the cycle. As if anticipating this outcome, backward induction will cause the unfortunate player to accept the cost M the first time it has to decide in v. However, it should be pointed out that a slight change in the upper bound may completely turn around the situation and cause the other player to pay M. Thus, the outcome of the game would depend on the remainder of the division of the bound by the cycle cost. This would cause a highly erratic solution structure and does not permit a consistent answer to the decision problem.

1.1. Related literature

A closely related game is known as GEOGRAPHY (see Schaefer [12]). It is played on a directed graph with no costs. Starting from a designated vertex $s \in V$, the two players move together and take turns in selecting the next vertex. The objective of the game is quite different from SHORTEST PATH GAME, namely, the game ends as soon as the players get stuck in a vertex and the player who has no arc left for moving on loses the game. Moreover, there is a further restriction that in GEOGRAPHY each arc may be used at most once.

Schaefer [12] already showed PSPACE-completeness of GEOGRAPHY. Lichtenstein and Sipser [10] proved that the variant VERTEX GEOGRAPHY, where each vertex cannot be visited more than once, is PSPACE-complete for planar bipartite graphs of bounded degree. This was done as an intermediate step for showing that GO is PSPACE-complete. Fraenkel and Simonson [8] gave polynomial time algorithms for GEOGRAPHY and VERTEX GEOGRAPHY when played on directed acyclic graphs. In Fraenkel et al. [9] it was proved that also the undirected variant of GEOGRAPHY is PSPACE-complete. However, if restricted to bipartite graphs they provided a polynomial time algorithm by using linear algebraic methods on the bipartite adjacency matrix of the underlying graph. Note that this result is in contrast to the PSPACE-completeness result of Section 3 for SHORTEST PATH GAME on bipartite undirected graphs. Bodlaender [2] showed that directed VERTEX GEOGRAPHY is linear time solvable on graphs of bounded treewidth. For directed GEOGRAPHY such a result was shown under the additional restriction that the degree of every vertex is bounded by a constant - the unrestricted variant however is still open.

Recently, the *spe*-path of SHORTEST PATH GAME was used in Darmann et al. [4] as a criterion for sharing the cost of the shortest path (in the classical sense) between two players. A different variant of two players taking turns in the decision on a combinatorial optimization problem and each of them optimizing its own objective function was recently considered for the Subset Sum problem by Darmann et al. [5].

1.2. Our contribution

We introduce the concept of *spe*-path resulting from backward induction in a game tree with complete and perfect information, where two players pursue the optimization of their own objective functions in a purely rational way. Thus, a solution concept for the underlying game is determined which incorporates in every step all anticipated decisions of future steps.

The main question we ask in this work concerns the complexity status of computing such a *spe*-path, if the game consists in the joint exploration of a path from a source to a sink. We believe that questions of this type could be an interesting topic also for other problems on graphs and beyond.

We can show in Section 2.1 that for *directed graphs* SHORTEST PATH GAME is PSPACE-complete even for bipartite graphs, while for acyclic directed graphs a linear time algorithm is given in Section 2.2. These results are in line with results from the literature for the related game GEOGRAPHY.

On the other hand, for *undirected graphs* we can show in Section 3 that again SHORTEST PATH GAME is PSPACE-complete even for bipartite graphs by a fairly complicated reduction from QUANTIFIED 3-SAT while the related problem GEOGRAPHY is polynomially solvable on undirected bipartite graphs. This surprising difference shows that finding paths with minimal costs can lead to dramatically harder problems than paths concerned only with reachability.

In Section 4 we give a fairly involved algorithm to determine the *spe*-path on undirected *cactus graphs* in polynomial time. It is based on several dynamic programming arrays and a partitioning of the graph into components. The running time of the algorithm can be bounded by $O(n^2)$, where n denotes the number of vertices in the graph. The easier case of directed cactus graphs follows as a consequence. We also argue that an extension of this partitioning technique to slightly more general graphs, such as outerplanar graphs, is impossible.

A preliminary version describing some of the results given in this paper but omitting most of the proofs and only sketching the algorithms appeared as Darmann et al. [6].

2. Spe-paths for Shortest Path Game on directed graphs

In general, there seems to be no way to avoid the exploration of the exponential decision tree. In fact, we can show a strong negative result. Let us first define the following decision problem.

> Shortest Path Game:
> Given a weighted graph G with dedicated vertices s, t and two positive values C_A, C_B, does the *spe*-path yield costs $c(A) \leq C_A$ and $c(B) \leq C_B$?

This means that we ask whether the unique subgame perfect equilibrium of the associated extensive form game fulfills the given cost bounds C_A and C_B.

2.1. PSPACE*-completeness*

The PSPACE-completeness of Shortest Path Game on general graphs can be shown by constructing an instance of the problem such that the *spe*-path decides the winner of Vertex Geography. In fact, we can give an even stronger result with little additional effort.

Theorem 1. Shortest Path Game *is* PSPACE*-complete even for bipartite directed graphs.*

Proof. Inclusion in PSPACE can be shown easily by considering that the height of the game tree is bounded by $2|A|$. Hence, we can determine the *spe*-path in polynomial space by exploring the game tree in a DFS-way. In every node currently under consideration we have to keep a list of decisions (i.e. neighboring vertices in the graph) still remaining to be explored and the cost of the currently preferred subpath among all the options starting in this node that were already explored. By the DFS-processing there are at most $2|A|$ nodes on the path from the root to the current node for which this information has to be kept.

We provide a simple reduction from Vertex Geography, which is known to be PSPACE-complete for planar bipartite directed graphs where the in-degree and the out-degree of a vertex is bounded by two and the degree is bounded by three (Lichtenstein and Sipser [10]). For a given instance of Vertex Geography we construct an instance of Shortest Path Game, such that the *spe*-path

path decides the winner of VERTEX GEOGRAPHY: Given the planar bipartite directed graph $G = (V, A)$ of VERTEX GEOGRAPHY with starting vertex s, we can two-color the vertices of V because G is bipartite. For the two-coloring, we use the colors red and green and color the vertices such that s is a green vertex. We create a new graph H for SHORTEST PATH GAME as follows: First we assign a cost ε to every arc $e \in E$. Then we introduce a new vertex t which we color red, and an arc of weight $M \gg 0$ from each green vertex to t. Next, introduce a green vertex z and an arc of weight $M \gg 0$ from each red vertex to z. Finally, introduce an arc of cost ε from z to t. The constants C_A and C_B are set to $C_A = 2\,\varepsilon$ and $C_B = M$. This means that a "yes"-instance corresponds to player A winning VERTEX GEOGRAPHY. It is not hard to see that H is a bipartite directed graph. Note that since the constructed graph is bipartite the rule of VERTEX GEOGRAPHY saying that each arc can be used at most once is equivalent to (R2) in SHORTEST PATH GAME.

Whenever a player gets stuck in a vertex playing VERTEX GEOGRAPHY, it would be possible to continue the path in H towards t (possibly via z) by choosing the arc of cost M. On the other hand, both players will avoid to use such a costly arc as long as possible and only one such arc will be chosen. Thus, the *spe*-path for SHORTEST PATH GAME will incur $\leq 2\,\varepsilon$ cost to one player and exactly cost M to the other player, who is thereby identified as the loser of VERTEX GEOGRAPHY. This follows from the fact that if both players follow the *spe*-path, they can anticipate the loser. If it is A, then this player will immediately go from s to t. If it is player B, then A will choose an arc with cost ε, then B will go to z paying M, and A from z to t at cost ε. □

Note that the result of Theorem 1 also follows from Theorem 3 be replacing each edge in the undirected graph by two directed arcs. However, we believe that the connection to GEOGRAPHY established in the above proof is interesting in its own right.

Remark:
The above theorem is true even under the following restrictions: the indegree of the vertices is bounded by two, the outdegree is bounded by three, and the degree of the vertices is bounded by four. However the planarity is lost and the method for getting a planar graph described in Lichtenstein and Sipser [10] does not carry over to SHORTEST PATH GAME in an obvious way.

2.2. Directed acyclic graphs

If the underlying graph G is acyclic we can do better by devising a strongly polynomial time dynamic programming algorithm. It is related to a dynamic programming scheme for the longest path problem in acyclic directed graphs.

For each vertex $v \in V$ we define $S(v) := \{u \mid (v, u) \in A\}$ as the set of successors of v. Then we define the following two dynamic programming arrays for each vertex v:

$p_d(v)$: minimal path cost to go from v to t for the player *deciding* in v.

$p_f(v)$: minimal path cost to go from v to t for the *follower*, i.e. the player **not** deciding in v.

The two arrays are initialized as follows:

$p_d(t) = p_f(t) = 0$

$p_d(v) = p_f(v) = \infty$ for all $v \in V, v \neq t$.

Starting from the destination vertex t and moving backwards in the graph we iteratively compute the values of $p_d(v)$ and $p_f(v)$. Note that each such entry is computed only once and never updated later.

Algorithm 1 Finding *spe*-paths for SHORTEST PATH GAME on acyclic directed graphs.

```
1: repeat
2:     find $v \in V$ with $p_d(v) = \infty$ such that $p_d(u) \neq \infty$ for all $u \in S(v)$
3:     let $u' := \arg\min\{c(v,u) + p_f(u) \mid u \in S(v)\}$
4:     $p_d(v) := c(v,u') + p_f(u')$
5:     $p_f(v) := p_d(u')$
6: until $p_d(s) \neq \infty$
```

In each iteration of the **repeat**-loop one entry $p_d(v)$ is reduced from ∞ for some vertex $v \in V$. Thus the algorithm terminates after $|V| - 1$ iterations, but we have to show that in each iteration a vertex v is found in line 2. Assume otherwise: If no vertex v remains with $p_d(v) = \infty$, then also $p_d(s) \neq \infty$ and we would have stopped the algorithm before. If for all vertices $v \in V$ with $p_d(v) = \infty$ there exists a vertex $u \in S(v)$ with $p_d(u) = \infty$, then we could apply the same argument to u. Thus, also u has a successor u_s with $p_d(u_s) = \infty$. Iterating this argument we can build a path from v to u and to u_s and so on. By construction, this path never ends, because otherwise the last vertex of the path would fulfill the conditions of line 2. But this means that the path is a cycle in contradiction to the assumption that G is acyclic.

Note that this algorithm resembles in some way the classical Dijkstra algorithm, but moves backwards from sink back to source. Its running time is linear in the number of arcs: In a preprocessing step, assign to each vertex u a label with the number of successors $|S(u)|$. Whenever a vertex v is given a distance value (line 4) decrement the label of all vertices u with $(u,v) \in A$ by one. If a label reaches 0, u can be put into a set of vertices fulfilling the condition of line 2. For each such candidate, the best successor according to line 3 can be found be going through all emanating arcs. Thus, each arc generates a constant number of computation steps and we have the following:

Theorem 2. *The* spe*-path of* SHORTEST PATH GAME *on acyclic directed graphs can be computed in* $O(|A|)$ *time.*

In Section 4 we will consider another special graph class and derive a polynomial time algorithm for Shortest Path Game on undirected cactus graphs. We will argue that this result immediately yields a polynomial time algorithm also for directed cactus graphs as stated in Corollary 5.

3. Spe-Paths for Shortest Path Game on undirected graphs

We will provide a reduction from the following problem Quantified 3-Sat which is known to be PSPACE-complete (Stockmeyer and Meyer [13]).

Definition (Quantified 3-Sat):

> GIVEN: Set $X = \{x_1, \dots, x_n\}$ of variables and a quantified Boolean formula
>
> $$F = (\exists x_1)(\forall x_2)(\exists x_3) \dots (\forall x_n)\, \phi(x_1, \dots, x_n)$$
>
> where ϕ is a propositional formula over X in 3-CNF (i.e., in conjunctive normal form with exactly three literals per clause).
>
> QUESTION: Is F true?

Let $C_1, \dots, C_m$ denote the clauses that make up ϕ, i.e., $\phi(x_1, \dots, x_n) = C_1 \wedge C_2 \wedge \dots C_m$, where each C_i, $1 \leq i \leq m$, contains exactly three literals. Quantified 3-Sat can be interpreted as the following game (cf. Fraenkel and Goldschmidt [7]): There are two players (the existential- and the universal-player) moving alternately, starting with the existential-player. The i-th move consists of assigning a truth value ("true" or "false") to variable x_i. After n moves, the existential-player wins if and only if the produced assignment makes ϕ true.

3.1. PSPACE*-completeness*

The following result shows a notable difference between Shortest Path Game and Geography, since Fraenkel et al. [9] showed that Geography is polynomially solvable on undirected bipartite graphs (while PSPACE-complete on general undirected graphs).

Theorem 3. Shortest Path Game *on undirected graphs is* PSPACE*-complete, even for bipartite graphs.*

Proof. Inclusion in PSPACE follows from a similar argument as in the proof of Theorem 1. Given an instance $\mathcal{Q}$ of Quantified 3-Sat we construct an instance $\mathcal{S}$ of Shortest Path Game by creating an undirected graph $G = (V, E)$ as follows. The vertices are 2-colored (using the colors red and green) to show that G is bipartite. To construct G, we introduce (see Figure 1):

- green vertices d, p, r, red vertices w, q, t
- edges $\{p, q\}$, $\{r, t\}$, $\{w, d\}$ and $\{d, t\}$
- for each clause C_j, a green vertex c_j
- for each even i, $2 \leq i \leq n$, an "octagon", i.e.,
  - red vertices $v_{i,0}, v_{i,2}, v_{i,4}, v_{i,6}$
  - green vertices $v_{i,1}, v_{i,3}, v_{i,5}, v_{i,7}$
  - edges $\{v_{i,\ell}, v_{i,\ell+1}\}$, $0 \leq \ell \leq 6$, and edge $\{v_{i,7}, v_{i,0}\}$
- for each odd i, $1 \leq i \leq n$, a "hexagon", i.e.,
  - green vertices $v_{i,0}, v_{i,2}, v_{i,4}$
  - red vertices $v_{i,1}, v_{i,3}, v_{i,5}$
  - edges $\{v_{i,\ell}, v_{i,\ell+1}\}$, $0 \leq \ell \leq 4$, and edge $\{v_{i,5}, v_{i,0}\}$

In order to connect these parts, we introduce:

- for each even i, $2 \leq i \leq n$
  - a green vertex u_i
  - edges $\{v_{i-1,3}, u_i\}$, $\{u_i, v_{i,0}\}$ and the edges $\{v_{i,2}, r\}$, $\{v_{i,6}, r\}$
  - edge $\{v_{i,4}, v_{i+1,0}\}$, where $v_{n+1,0} := p$
  - for each clause C_j, the edge $\{v_{i,2}, c_j\}$ if $x_i \in C_j$ and $\{v_{i,6}, c_j\}$ if $\bar{x}_i \in C_j$
- for each odd i, $1 \leq i \leq n$
  - the edges $\{v_{i,1}, r\}$, $\{v_{i,5}, r\}$
  - for each clause C_j, the edge $\{v_{i,1}, c_j\}$ if $x_i \in C_j$ and $\{v_{i,5}, c_j\}$ if $\bar{x}_i \in C_j$
- for each j, $1 \leq j \leq m$,
  - edges $\{q, c_j\}$ and $\{w, c_j\}$

Abusing notation, for $1 \leq i \leq n$, let $x_i := \{v_{i,0}, v_{i,1}\}$, and $\bar{x}_i := \{v_{i,0}, v_{i,5}\}$ if i is odd resp. $\bar{x}_i := \{v_{i,0}, v_{i,7}\}$ if i is even, i.e., we identify a literal with an edge of the some label. For illustration, in Figure 1 we assume $C_1 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ and $C_2 = (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_n)$.

Finally, we define the edge costs.[3] We start with the edges emanating from vertex w: The cost of edge $\{w, d\}$ is $c\{w, d\} = 0$, all other edges emanating

[3]In the introduction edge costs were defined to be strictly positive. For simplicity we use zero costs in this proof, but these could be easily replaced by some small $\varepsilon > 0$.

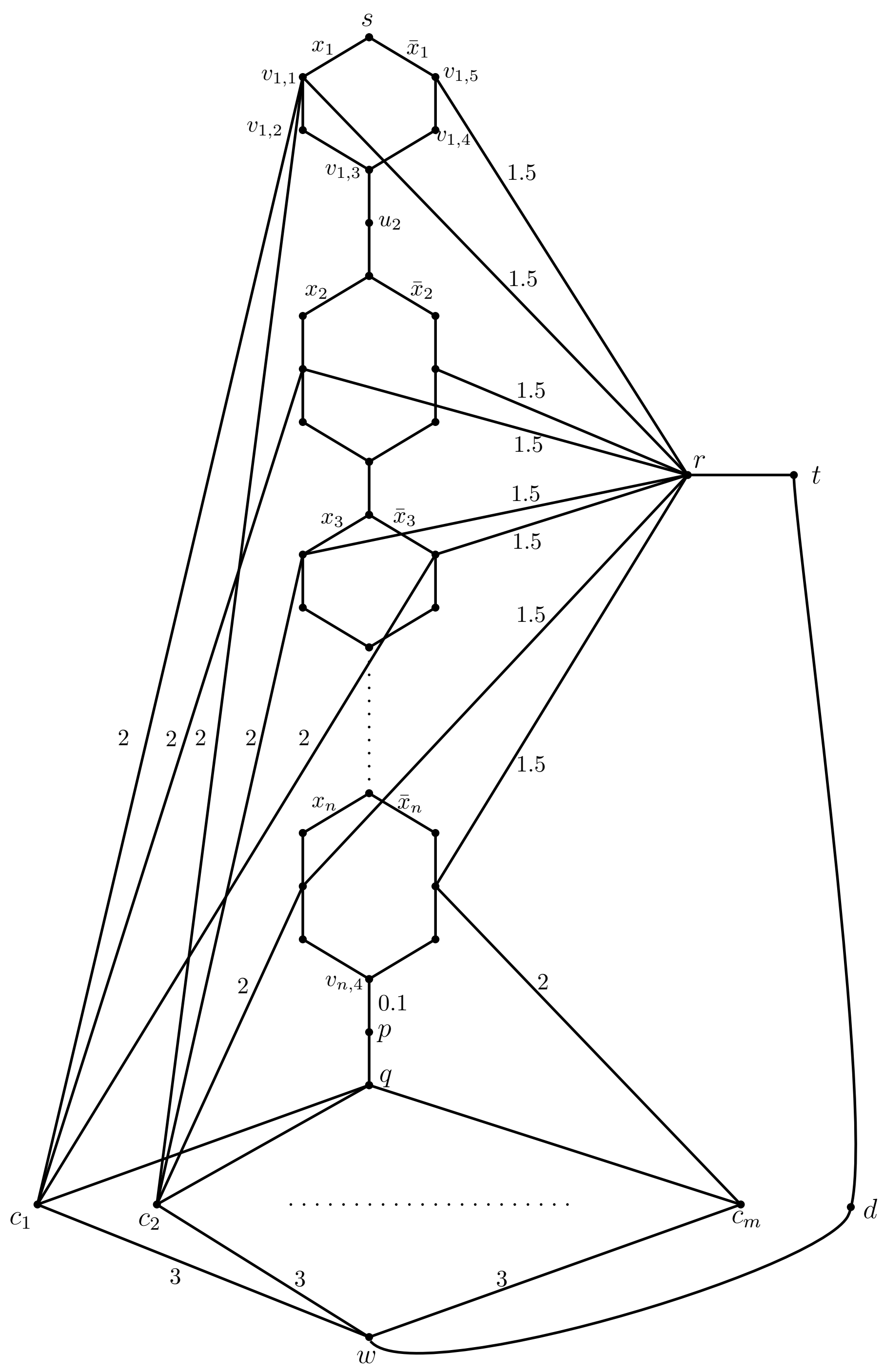


Figure 1: Undirected graph G of instance $\mathcal{S}$ constructed for an instance $\mathcal{Q}$ of Quantified 3-Sat with $C_1 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ and $C_2 = (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_n)$.

from w have cost 3. The edge $\{r, t\}$ has cost $c(\{r, t\}) = 0$, while each other edge emanating from r has cost 1.5. The edge $\{v_{n,4}, p\}$ has cost $c(\{v_{n,4}, p\}) = 0.1$. For each $1 \leq j \leq m$, the edges emanating from vertex c_j which do not correspond to $\{c_j, w\}$ or $\{c_j, q\}$ have cost 2. The remaining edges have zero cost.
Note that from the fact that each edge connects a green with a red vertex, it immediately follows that G is a bipartite graph. Now, in $\mathcal{S}$ we set $C_A = 0$ and $C_B = 1.5$, and ask if the *spe*-path yields costs $c(A) \leq C_A$ and $c(B) \leq C_B$.

Claim. $\mathcal{Q}$ is a "yes"-instance of QUANTIFIED 3-SAT $\Leftrightarrow$ $\mathcal{S}$ is a "yes"-instance of SHORTEST PATH GAME.

Proof of claim. Player A starts with the first move along an edge emanating from s, i.e., A moves along x_1 or $\bar{x}_1$. Due to the fact that s is a green vertex and G is bipartite, we can conclude that in a green vertex, A needs to choose the next edge, and in a red vertex, it is B's turn to choose the next edge.
W.l.o.g. assume A moves along x_1 (which, as we will see later, corresponds to setting to true x_1 in instance $\mathcal{Q}$). Because of (R2), player B cannot move back to s along x_1. Now, B has three choices. B either moves along (i) edge $\{v_{1,1}, r\}$ with cost 1.5, or (ii) edge $\{v_{1,1}, v_{1,2}\}$ with zero cost, or (iii) an edge of cost 2 connecting $v_{1,1}$ with some vertex c_j.
CASE I: B moves along $\{v_{1,1}, r\}$.
Then A will use the edge (r, t) of zero cost in order to minimize her own total cost and *the game ends with* $c(A) = 0$ *and* $c(B) = 1.5$.

As a consequence of the outcome in Case I, B will not choose (iii), because B would end with $c(B) \geq 2$ in that case. Note that each vertex $v_{i,1}$ for odd i (resp. $v_{i,2}$ for even i) is a red vertex, i.e., an analogous situation applies in each such vertex. Thus, we can observe the following:

> **(Observation 1)** Player B does not move along an edge of cost 2, i.e., an edge $\{v_{i,1}, c_j\}$ for odd i resp. $\{v_{i,2}, c_j\}$ for even i.

CASE II: B moves along $\{v_{1,1}, v_{1,2}\}$.
In $v_{1,2}$, player A needs to move along $\{v_{1,2}, v_{1,3}\}$ (again, A cannot move back to $v_{1,2}$ along $\{v_{1,1}, v_{1,2}\}$ due to (R2)). At $v_{1,3}$, it is again player B's turn.
CASE IIa: B moves along $\{v_{1,3}, v_{1,4}\}$.
Then A necessarily moves along $\{v_{1,4}, v_{1,5}\}$, leaving B with the decision in $v_{1,5}$. B must not move along $\bar{x}_1$ because of (R2). Because of Observation 1, this means that B moves along $\{v_{1,5}, r\}$ of cost 1.5. Clearly, A then chooses edge (r, t), and again *the game ends with* $c(A) = 0$ *and* $c(B) = 1.5$.
CASE IIb: B moves along $\{v_{1,3}, u_2\}$.
In the next step, A moves to $v_{2,0}$. Now, it is B's turn to pick x_2 or $\bar{x}_2$, i.e., to decide which literal she sets to true in instance $\mathcal{Q}$. W.l.o.g. assume B moves along $\bar{x}_2$. In the next step A has only one edge to move along, leaving B with the decision in $v_{2,6}$. Analogously to above, *the game ends with* $c(A) = 0$ *and* $c(B) = 1.5$ if B moves along $\{v_{2,6}, r\}$. If B does not move along $\{v_{2,6}, r\}$, the players continue until $v_{2,4}$ is reached, because B does not choose an edge of cost

2 (see Observation 1). Again, it is B's turn:
If B moves along $\{v_{2,4}, v_{2,3}\}$, A necessarily moves along $\{v_{2,3}, v_{2,2}\}$. In $v_{2,2}$, B is not allowed to move along $\{v_{2,2}, v_{2,1}\}$ because this would result in a deadlock (A would be unable to move along another edge due to (R1)). With Observation 1, this means that B moves along $\{v_{2,2}, r\}$; i.e., again *the game ends with* $c(A) = 0$ *and* $c(B) = 1.5$.
Assume that B moves along $\{v_{2,4}, v_{3,0}\}$, leaving A to choose in $v_{3,0}$ between x_3 and $\bar{x}_3$. It is easy to see that in what follows, repeatedly analogous decisions need to be made. As a consequence, we can observe the following.

> **(Observation 2)** If Shortest Path Game ends before vertex $v_{n,4}$ is reached, it ends with $c(A) = 0$ and $c(B) = 1.5$.
>
> **(Observation 3)** If vertex $v_{n,4}$ is visited, then for even i player B has chosen between x_i and $\bar{x}_i$, while for odd i player B has chosen between x_i and $\bar{x}_i$.

Assume that vertex $v_{n,4}$ is visited, where it is B's turn to take the next decision. Analogously to above we know that, if B does not move along $\{v_{n,4}, p\}$, then again the game ends with $c(A) = 0$ and $c(B) = 1.5$. Assume B moves along $\{v_{n,4}, p\}$ with cost 0.1. Clearly, in the next step A has to move along $\{p, q\}$ leaving B with the decision in vertex q, i.e., B has to choose a vertex c_j (i.e, a clause C_j) to move to.
Let ϕ be the truth assignment that sets to true exactly the literals (edges) chosen by a player so far. We will argue that B aims at picking a clause C_j which is not satisfied by ϕ. We will illustrate this by assuming that B decides to move to vertex c_1 representing clause $C_1 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$.
If C_1 is not satisfied by ϕ, i.e., if none of the literals $\bar{x}_1, \bar{x}_2, x_3$ has been moved along, then all edges $x_1, x_2, \bar{x}_3$ have been used already. As a result, A is not allowed to move along the edges $\{c_1, v_{1,1}\}$, $\{c_1, v_{2,2}\}$ and $\{c_1, v_{3,5}\}$ because of (R2). Again due to (R2), A must not move back to q along edge $\{c_1, q\}$. Thus, A needs to move along $\{c_1, w\}$ imposing a cost of 3 on player A. In the next steps, B obviously moves along edge $\{w, d\}$, implying that A moves along $\{d, t\}$. *Thus, the game ends with* $c(A) = 3$ *and* $c(B) = 0.1$.
On the other hand, if at least one of the literals $\bar{x}_1, \bar{x}_2, x_3$ have been used so far, then at least one of the edges $x_1, x_2, \bar{x}_3$ have not already been used. As a consequence, A – trying to avoid the expensive edge $\{c_1, w\}$ – is able to use one of the edges $\{c_1, v_{1,1}\}$, $\{c_1, v_{2,2}\}$ and $\{c_1, v_{3,5}\}$. W.l.o.g. assume that $\bar{x}_3$ has not been used and that A now moves along $\{c_1, v_{3,5}\}$ with cost 2. At $v_{3,5}$, B cannot move along $\bar{x}_3$ due to (R2). If B moves along $\{v_{3,5}, v_{3,4}\}$, a deadlock arises: A is neither able to move back to $v_{3,5}$ along $\{v_{3,5}, v_{3,4}\}$, nor to move along $\{v_{3,4}, v_{3,3}\}$, because this would violate (R2) since both $v_{3,3}$ and $v_{3,5}$ have already been visited. Due to (R1), player B hence must not use $\{v_{3,5}, v_{3,4}\}$. Consequently, player B possibly needs to choose between an edge of cost 2 or edge $\{v_{3,5}, r\}$ with cost 1.5. Clearly, B chooses the latter edge, since in vertex r player A will move along $\{r, t\}$. *Thus, the game ends with* $c(A) = 2$ *and* $c(B) = 0.1 + 1.5 = 1.6$.

Summing up, B will move along $\{v_{n,4}, p\}$ if and only if there is a clause which is not satisfied by ϕ, because otherwise B would have been better off by moving along an edge towards r earlier, resulting in an outcome with $c(A) = 0$ and $c(B) = 1.5$. In other words, instance $\mathcal{S}$ of Shortest Path Game ends with $c(A) = 0$ and if and only if $\mathcal{Q}$ is a "yes"-instance of Quantified 3-Sat. □

Remark. Using the reduction provided in the above proof, it is not hard to show that Shortest Path Game remains PSPACE-complete in bipartite undirected graphs, even if (R2) is relaxed in the way that we only require an edge not to be used more than once, but do not rule out cycles of any length.

3.2. Undirected acyclic graphs (trees)

Complementing Section 2.2 we observe that in undirected acyclic graphs, i.e., trees, Shortest Path Game becomes trivial: Since in a tree there is exactly one path between two dedicated vertices, the two player have no choice but to follow the unique path from s to t. Taking a diversion to a side branch, the players would always have to go back the same way they came implying a cycle of even length.

4. Shortest Path Game on undirected cactus graphs

Since Shortest Path Game is trivial on undirected trees and rather easy to solve in polynomial time on directed acyclic graphs we try to push the bar a bit higher and find polynomial algorithms on more general graph classes. In this section we will show that a cactus graph still allows a polynomial solution of Shortest Path Game. This is mainly due to a decomposition structure which allows to solve components of the graph to optimality independently from the solution in the remaining graph. However, we will illustrate by an example in Section 4.4 that more general graph, such as outerplanar graphs, which are a superset of cactus graphs, do not allow such a decomposition of the solution structure. This gives a certain indication that Shortest Path Game might become computationally intractable on slightly more general graphs.

A *cactus graph* (also known as *Husimi tree*) is a graph where each edge is contained in at most one simple cycle. Equivalently, any two simple cycles have at most one vertex in common. This means that one could contract each cycle into a vertex in a unique way and obtain a tree. Note that cactus graphs are a subclass of series-parallel graphs and thus have treewidth at most 2.

Considering Shortest Path Game, it is easy to see that the union of all simple paths from s to t define a subgraph G' consisting of a unique sequence of edges (which are bridges of the graph) and simple cycles. We will call G' the *connection strip* between s and t. All other vertices of the graph are "dead end streets", i.e. edges and cycles branching off from G' (see Figure 2). In the *spe*-path vertices in $G \setminus G'$ could be included only to change the role of the decision

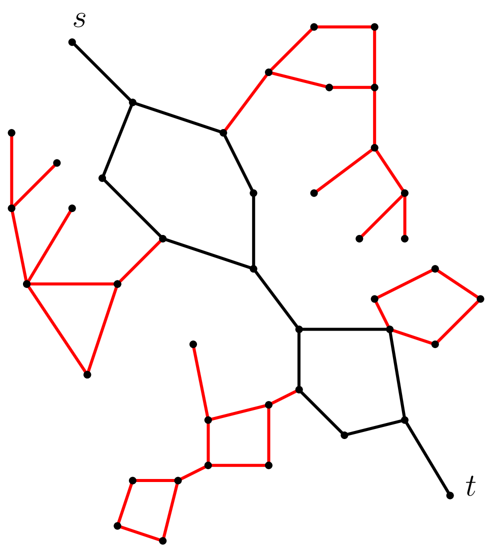


Figure 2: Graph G with connection strip G' (in black) and branches $G \setminus G'$ (in red).

maker in a vertex of G'. Clearly, any such deviation from G' must be a cycle rooted in some vertex of G'. Moreover, by (R2) only cycles (not necessarily simple) of odd length might be traversed in this way. This structural property gives rise to a preprocessing step where all vertices in $G \setminus G'$ are contracted into a *swap option* in a vertex $v \in G'$ (see Figure 3) with cost $(sw_d(v), sw_f(v))$ meaning that if the path of the two players reaches a certain vertex $v \in G'$, the current decider has the option to switch roles (by entering an odd cycle in $G \setminus G'$ rooted in v) at cost of $sw_d(v)$ for himself (the decider) and $sw_f(v)$ for the other player (the follower).

If there is more more than one cycle rooted in some vertex v, each of them is contracted separately. But since the *spe*-path can utilize at most one swap in v, we simply pick the swap option with the smallest cost for the decider.

Our algorithm will first compute these swap costs by recursively traversing the components of $G \setminus G'$ in Section 4.1. Then, in the second step, the *spe*-path in G' is computed by moving backwards from t towards s in Section 4.2. In each iteration of this second step a cycle is considered, where the part of the *spe*-path from the “exit” of the cycle towards t is already known (resp. only a small number of possibilities remain). By evaluating several dynamic programming arrays, the best options for moving from the “entry” of this cycle to the exit are computed. These iterations are continued until the starting vertex s is reached.

4.1. Contraction of the branches

Consider a cycle $C(v_0)$ which is connected to the remaining graph only via v_0 and all other vertices of $C(v_0)$ have degree 2, i.e. all other edges and cycles incident

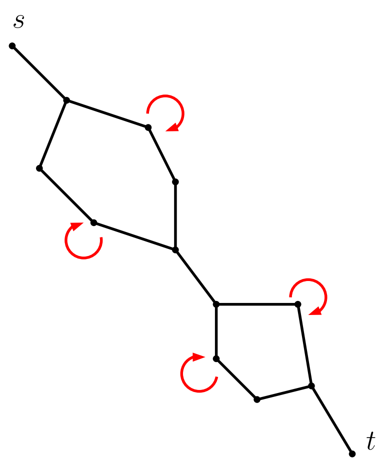


Figure 3: Graph G with connection strip G': branches $G \setminus G'$ are contracted into swaps.

to these vertices were contracted into swap options before. For simplicity of notation we refer to vertices by their index number and assume that $C(v_0) = C(0)$ consists of a sequence of vertices $0, 1, 2, \ldots, k-1, k, 0$.

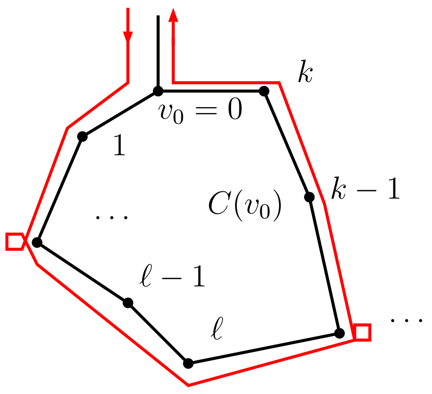


Figure 4: Cycle $C(v_0)$ with a possible path (in red) going round the cycle (with possible swaps on the way).

There are four possibilities how to use the cycle for a swap: The players could enter the cycle by the edge $(0,1)$ and go around the full length of the cycle (possibly using additional swaps in vertices of the cycle) as depicted in Figure 4. Or after edge $(0,1)$ the players could move up to some vertex $\ell \in \{1, 2, \ldots, k\}$, turn around by utilizing a swap option in ℓ and go back to 0 the way they came as depicted in Figure 5. Note that in the latter case, the players can not use any additional swaps in vertices $1, \ldots, \ell - 1$ (resp. $k, k-1, \ldots, \ell + 1$) since in that

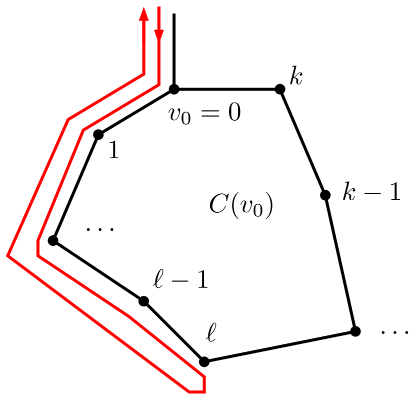


Figure 5: Cycle $C(v_0)$ with a possible path (in red) using a swap in vertex ℓ.

case the swap vertex would be visited three times in violation of (R2). Thus, we have to distinguish in each vertex whether such a turn around is still possible or whether it is ruled out by a swap in a previously visited vertex of the cycle.

These two configurations can also be used in a laterally reversed way moving on the cycle in the different direction starting with the edge $(0, k)$ which yields four cases in total.

Let $D \in \{A, B\}$ be the decision maker in 0. We introduce the following generic notation for dynamic programming arrays:

> $d_P^{\pm}(i)$: denotes the cost of a certain path starting in vertex i and ending in a fixed specified vertex.

We use the subscript $P \in \{d, f\}$, where $P = d$ signifies that the cost occurs for the player deciding in i and $P = f$ refers to the cost of the follower, i.e. the other player not deciding in i. Superscript $\pm \in \{+, -\}$ shows that the decider in i is equal to D if $\pm = +$, or whether the other player decides in i, i.e. $\pm = -$. For simplicity, we also extend the cost range and use cost $\top$ if a path is infeasible. When taking the minimum of values, $\top$ stands for an arbitrarily large value. Adding the cost of a path to an array entry $\top$ yields again $\top$.

Following this system we define:
$tc_P^{\pm}(i)$: minimal cost to move from i back to 0, if a turn around is still possible.
$rc_P^{\pm}(i)$: minimal cost to move from i back to 0, if no turn around is possible and the path has to go around the cycle, i.e. visit vertices $i + 1, i + 2, \dots, k, 0$, with possible swaps on the way. If one player decides to turn around at some vertex i, the cost of the path back towards vertex 0 is completely determined

since no choices remain open. The corresponding costs are independent from D and will be recorded as $\text{path}_P(i)$ in analogy to above.

Now we can state the appropriate update recursion for the case where D chooses $(0,1)$ as a first edge. We go backwards along this path and settle the minimal costs for vertices $k, k-1, \dots, 1$. The case where D moves into the other direction of the cycle is completely analogous and the final swap costs $sw(v_0)$ are given by the cheaper alternative.

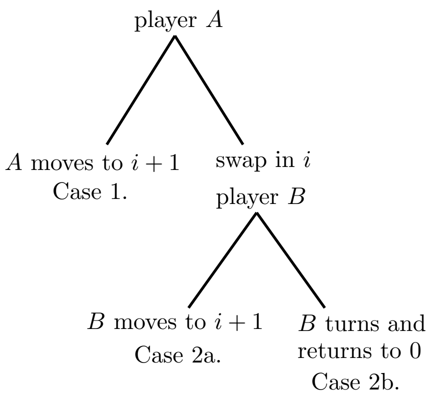


Figure 6: Decision tree for player A deciding in vertex i.

In any vertex i the decision process has at most three outcomes as illustrated in Figure 6.

1. move on along the cycle to vertex $i+1$:
   $rc_d^+(i) := c(i,i+1) + rc_f^-(i+1)$, $rc_f^+(i) = rc_d^-(i+1)$
   $rc_d^-(i) := c(i,i+1) + rc_f^+(i+1)$, $rc_f^-(i) = rc_d^+(i+1)$
   $tc_d^+(i) := c(i,i+1) + tc_f^-(i+1)$, $tc_f^+(i) = tc_d^-(i+1)$
   $tc_d^-(i) := c(i,i+1) + tc_f^+(i+1)$, $tc_f^-(i) = tc_d^+(i+1)$
2. make a swap (if available): Then the other player has (at most) two possibilities and chooses the one with the lower cost between 2a. and 2b. (or the only feasible choice), which automatically implies the cost for the decider in i.

2a. move on to vertex $i+1$:
   $rc_f^+(i) = sw_f(i) + c(i,i+1) + rc_f^+(i+1)$, $rc_d^+(i) := sw_d(i) + rc_d^+(i+1)$
   $rc_f^-(i) = sw_f(i) + c(i,i+1) + rc_f^-(i+1)$, $rc_d^-(i) := sw_d(i) + rc_d^-(i+1)$
   $tc_f^+(i) := sw_f(i) + c(i,i+1) + rc_f^+(i+1)$, $tc_d^+(i) := sw_d(i) + rc_d^+(i+1)$
   $tc_f^-(i) := sw_f(i) + c(i,i+1) + rc_f^-(i+1)$, $tc_d^-(i) := sw_d(i) + rc_d^-(i+1)$.

2b. turn around (if possible): Since the decider at the end of the return path in vertex 0 must be different from D, the feasibility of a turn around depends on the number of edges between 0 and i.
   If i is even:
   $tc_d^+(i) := sw_d(i) + \text{path}_f(i)$, $tc_f^+(i) := sw_f(i) + \text{path}_d(i)$
   $tc_d^-(i) := \top$, $tc_f^-(i) := \top$

If i is odd:
$tc_d^+(i) := \top$, $tc_f^+(i) := \top$
$tc_d^-(i) := sw_d(i) + \text{path}_f(i)$, $tc_f^-(i) := sw_f(i) + \text{path}_d(i)$

Now the decider in vertex i can anticipate the potential decision of the other player in case 2., since the other player will choose the better outcome between cases 2a. and 2b. (if 2b. is feasible). Hence, the decision maker in vertex i chooses the minimum between case 1. and case 2. (if a swap is possible) independently for all four dynamic programming entries. This immediately implies the cost for the other player.

It remains to discuss the initialization of the arrays for $i = k$, i.e. the last vertex in the cycle and thus the first vertex considered in the recursion. To avoid the repetition of all three cases implied by a possible swap at vertex k, we add two artificial vertices $k+1$ and $k+2$ and three artificial edges $(k, k+1)$, $(k+1, k+2)$ and $(k+2, 0)$ replacing the previous edge $(k, 0)$. We set $c(k, k+1) = c(k, 0)$ and the other two artificial edges have cost 0. It is easy to see that this extension of the cycle does not change anything. Now we can start the recursive computation at vertex $k + 2$ which has no swap option. (If vertex k does not have a swap option we can skip this extension and perform the following initialization for k replacing $k + 2$.) We get:

$rc_d^+(k+2) := c(k+2, 0)$, $rc_f^+(k+2) = 0$
$rc_d^-(k+2) := \top$, $rc_f^-(k+2) = \top$
$tc_d^+(k+2) := c(k+2, 0)$, $tc_f^+(k+2) = 0$
$tc_d^-(k+2) := \top$, $tc_f^-(k+2) = \top$

4.2. Main part of the algorithm

Now we proceed to the main part of the algorithm to determine the *spe*-path for moving from s to t along the connection strip G' after the reminder of the graph was contracted into swap options in vertices of G'. We traverse the connection strip backwards starting in t and moving upwards in direction of s. Recall that the connection strip consists of a sequence of cycles and edges. In the following we will focus on the computation of an optimal subpath for one cycle of this sequence. Each such cycle has two designated vertices which all paths from s to t have to traverse, an "upper vertex" v_0 through which every path starting in s enters the cycle, and a "lower vertex" v_ℓ through which all paths connecting the cycle with t have to leave the cycle. As before we refer to the vertices of the cycle simply by their index numbers and denote the cycle as $0, 1, \ldots, \ell, \ell+1, \ldots, k, 0$.

If we assume that the decision maker decides in 0 to take the edge $(0, 1)$ there are two main possibilities for the path from 0 to ℓ and onwards to the next cycle or edge. The other situation, where the decision maker starts with the edge $(0, k)$ is completely symmetric and the decider will finally take the better of the two options.

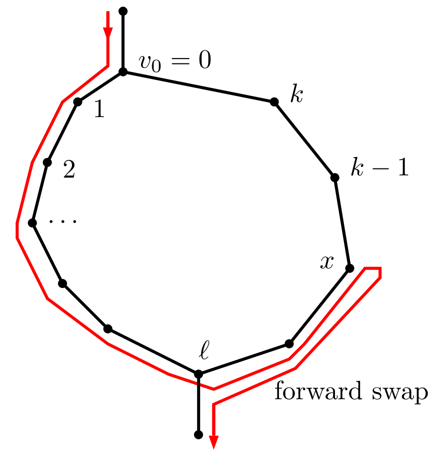


Figure 7: Case (i): Forward swap in x.

Case (i): The two players may move along the vertices $0, 1, \ldots, \ell$, possibly with a few swaps on the way. After reaching ℓ, they may either exit the cycle or continue to $\ell + 1, \ldots, x$, make a *forward swap* in x and return back via $x - 1, x - 2, \ldots$ back to ℓ and finally exit the cycle (see Figure 7). As a special variant of this situation, the players may also never swap in some vertex x but go back to 0 thus traversing the full cycle and then taking the path $0, 1, \ldots, \ell$ a second time as depicted in Figure 8.

Case (ii): As a second, more complicated possibility, the two players may also move along vertices $0, 1, \ldots, j$, $j < \ell$, and then utilize a swap in j and return to 0. Then they are forced to move on from 0 to $k, k-1, \ldots, \ell$. After reaching ℓ they may either exit the cycle directly or they may also continue to $\ell - 1, \ell - 2, \ldots, y$ with $y > j$, make a *backward swap* in y and return via $y + 1, \ldots, \ell$ where they finally exit the cycle (see Figure 9).

Of course, all the resulting subpaths have to be feasible, in the sense that they actually lead to the desired swap between decider and follower and contain no even cycles. Moreover, all subpaths that are dead end streets used only for reaching a swap at some vertex are traversed twice and thus are not allowed to contain any swap except at their endpoints. In the above notation this holds for the subpaths $\ell + 1, \ldots, x$, the full cycle, $0, \ldots, j$ and $\ell, \ldots, y$.

From our backward processing through the connection strip we can assume that we already know the *spe*-path from v_ℓ to t (without using any other vertex from the current cycle). If there is a bridge leading from v_ℓ towards t, it is easy to take the costs of the *spe*-path from v_ℓ via this edge onwards to t into account. However, the situation gets more complicated if two cycles meet directly in one vertex, i.e. if the lower vertex v_ℓ of the current vertex is also the upper vertex of the following cycle in the connection strip towards t. In such a case we cannot

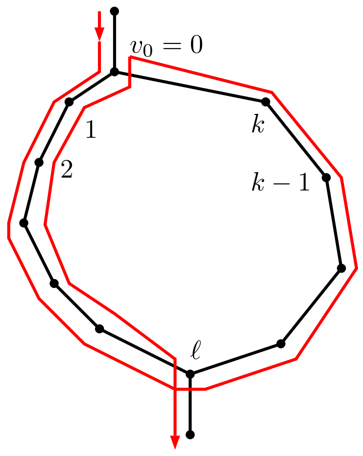


Figure 8: Case (i), special case: Traverse the full cycle.

combine the solutions of the two adjacent cycles in an arbitrary way since each player can decide in vertex v_ℓ at most once.

For the correct combination of solutions of two adjacent cycles it will be necessary to distinguish whether the path through the current cycle takes only one decision in v_ℓ and then leaves the cycle (Case (i) without forward swap in x or Case (ii) without backward swap in y) or whether the first decision in v_ℓ leads to another vertex in the cycle and thus a second decision takes place in v_ℓ with the only possible result of leaving the cycle. In the former case, we do not have any restriction on the solution of the following cycle in the connection strip (closer to t) since the path is also allowed to move back to v_ℓ (which is the upper vertex of that cycle) as it is done in Case (ii). We denote by ed_P^2 (exit costs) the costs of the *spe*-path from v_ℓ to t allowing to visit v_ℓ a second time. In the latter case, we only consider paths that never move back to v_ℓ in the following cycle (i.e. only Case (i) applies) and denote the cost of the *spe*-path from v_ℓ to t under this restriction by ed_P^1.

We now process the current cycle to determine the *spe*-path starting in v_0. Let $D \in \{A, B\}$ again be the decision maker in 0. Keeping the notational system introduced above we will introduce the following dynamic programming arrays:

$td_P^\pm(i)$: minimal cost to move from i, $i = 1, \dots, \ell - 1$, to ℓ and exit the cycle, if a turn around (according to Case (i)) is still possible.
$rd_P^\pm(i)$: minimal cost to move from i, $i = 1, \dots, \ell - 1$, to ℓ and exit the cycle, if no turn around is possible.
$ad_P^\pm(i)$: ("alternative path") minimal cost for moving from i, $\ell \le i \le k$, via $i-1, i-2, \dots$ to ℓ (Case (ii)). We will later add a parameter j to the definition of this array since it may be combined with a backward swap (see below).
$fs_P^\pm(i)$: ("forward swap") minimal cost for moving from i, $\ell \le i \le k$, without

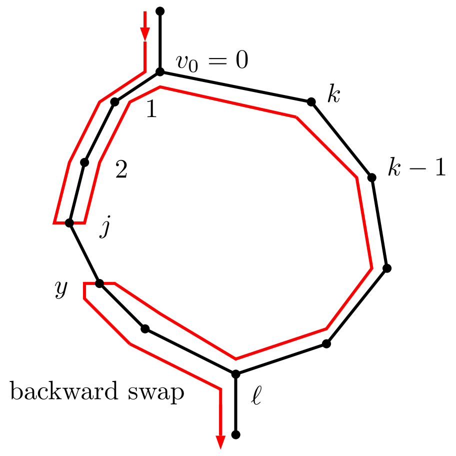


Figure 9: Case (ii): Backward swap in y.

swaps to some vertex x, $i \le x \le k$, make a swap in x and return to ℓ (Case (i)). The parity $\pm = +$ indicates that the decider in i is the same as the decider in ℓ.
$bs^{\pm}_P(i, j)$: ("backward swap") minimal cost for moving from i, $j < i \le \ell$, without swaps to some vertex y, $j < y \le i$, make a swap in y and return to ℓ (Case (ii)). Note that parameter j indicates a lower bound on the position of the swap vertex y. Since we do not know at this point at which vertex j the path starting in 0 was stopped by a swap, we have to compute the values of this array for all values of j, $j = 1, \ldots, \ell - 1$. The parity $\pm = +$ indicates that the decider in i is the same as the decider in ℓ.

We first show how to compute the entries of the auxiliary dynamic programming arrays starting with the forward swap moving backwards from a starting point $i = k' - 1$ (see below) down to $i = \ell$. Let $\mathrm{fpath}_P(i)$ denote the cost of the path from i back towards ℓ. We have to take the minimum between the following two cases (if the second case exists):

1. move on along the cycle to vertex $i + 1$:
   $fs^+_d(i) := c(i, i+1) + fs^-_f(i+1)$, $fs^+_f(i) := fs^-_d(i+1)$
   $fs^-_d(i) := c(i, i+1) + fs^+_f(i+1)$, $fs^-_f(i) := fs^+_d(i+1)$
2. make a swap in i (if available and if $i > \ell$): Then the other player has to return to ℓ to avoid a violation of (R2). The feasibility of such a turn depends on the number of edges between ℓ and i.
   If $i - \ell$ is even:
   $fs^+_d(i) := sw_d(i) + \mathrm{fpath}_f(i)$, $fs^+_f(i) := sw_f(i) + \mathrm{fpath}_d(i)$
   $fs^-_d(i) := \top$, $fs^-_f(i) := \top$
   If $i - \ell$ is odd:
   $fs^+_d(i) := \top$, $fs^+_f(i) := \top$
   $fs^-_d(i) := sw_d(i) + \mathrm{fpath}_f(i)$, $fs^-_f(i) := sw_f(i) + \mathrm{fpath}_d(i)$

As an initialization we start at a vertex with a swap option having the largest index $k' \leq k$ and insert the values of the above Case 2. for $i = k'$.

Considering the backward swap we let $\text{bpath}_P(i)$ denote the cost of the path from i towards ℓ. Now we have to perform the following computation for all values of $j = 1, \dots, \ell - 1$. Again we have to take the minimum between the following two cases (if the second case exists) while we move up from $i = j' + 1$ (defined below) to $i = \ell$.

1. move on along the cycle to vertex $i - 1$:
   $bs_d^+(i,j) := c(i-1,i) + bs_f^-(i-1,j)$, $bs_f^+(i,j) := bs_d^-(i-1,j)$
   $bs_d^-(i,j) := c(i-1,i) + bs_f^+(i-1,j)$, $bs_f^-(i,j) := bs_d^+(i-1,j)$
2. make a swap in i (if available and if $i < \ell$): Then the other player has to return to ℓ to avoid a violation of (R2). The feasibility of such a turn depends on the number of edges between i and ℓ.
   If $\ell - i$ is even:
   $bs_d^+(i,j) := sw_d(i) + \text{bpath}_f(i)$, $bs_f^+(i,j) := sw_f(i) + \text{bpath}_d(i)$
   $bs_d^-(i,j) := \top$, $bs_f^-(i,j) := \top$
   If $\ell - i$ is odd:
   $bs_d^+(i,j) := \top$, $bs_f^+(i,j) := \top$
   $bs_d^-(i,j) := sw_d(i) + \text{bpath}_f(i)$, $bs_f^-(i,j) := sw_f(i) + \text{bpath}_d(i)$

As an initialization we start at a vertex with a swap having the smallest index $j' > j$ and insert the values of the above Case 2. for $i = j'$.

Next we determine the path from 0 via $k, k-1, \dots$ to ℓ, when no turn arounds are possible any more (Case (ii)). We consider the decision at vertex i and compute the entries of the dynamic programming array from $i = \ell + 1$ on until $i = k$. In each vertex, the decision maker has to take the minimum between the following two cases (if the second case exists):

1. move on along the cycle to vertex $i - 1$:
   $ad_d^+(i) := c(i-1,i) + ad_f^-(i-1)$, $ad_f^+(i) := ad_d^-(i-1)$
   $ad_d^-(i) := c(i-1,i) + ad_f^+(i-1)$, $ad_f^-(i) := ad_d^+(i-1)$
2. make a swap in i (if available): Then the other player has to continue moving along the edge $(i-1,i)$ towards ℓ to avoid a violation of (R2).
   $ad_d^+(i) := sw_d(i) + ad_d^+(i-1)$, $ad_f^+(i) := sw_f(i) + c(i-1,i) + ad_f^+(i-1)$
   $ad_d^-(i) := sw_d(i) + ad_d^-(i-1)$, $ad_f^-(i) := sw_f(i) + c(i-1,i) + ad_f^-(i-1)$

Finally, we extend the definition to $i = 0$ and compute $ad_P^\pm(0)$ according to case 1. with k replacing $i - 1$.

Note that all entries of $ad_P^\pm$ are extended to include a parameter j in analogy to $bs_P^\pm$ which has no influence on the definition of the above recursion, although the array entries for different values of j may differ due to the different initialization.

As an initialization we have to consider the decision in ℓ for Case (ii). The decider can either move on directly to the next cycle resp. edge in the connection strip or make a swap, either by a swap option available in ℓ from the previous contraction of $G \setminus G'$ or by entering the backward swap. Because of (R2) at most one of the two swap variants can be part of the solution. Thus we have:

$$\begin{aligned} ad_d^+(\ell, j) &:= \min\{ed_d^2, sw_d + ed_f^1, bs_d^+(\ell, j) + ed_f^1\}, \\ ad_d^-(\ell, j) &:= \min\{ed_d^2, sw_d + ed_f^1, bs_d^+(\ell, j) + ed_f^1\}. \end{aligned} \tag{1}$$

The values for the follower are immediately implied by the selection of the minimum.

Now we can turn to the main arrays and determine their values for all values of i by moving backwards from $i = \ell - 1$ down to $i = 1$. We let $\mathrm{rpath}_P(i)$ denote the cost of the return path from i towards 0. In any vertex i the decision process has at most three outcomes, from which the cheapest is chosen. (Note that cases 1. and 2. are completely analogous to the earlier problem of contracting a cycle.)

1. move on along the cycle to vertex $i + 1$:
   $rd_d^+(i) := c(i, i+1) + rd_f^-(i+1)$, $rd_f^+(i) = rd_d^-(i+1)$
   $rd_d^-(k) := c(i, i+1) + rd_f^+(i+1)$, $rd_f^-(i) = rd_d^+(i+1)$
   $td_d^+(i) := c(i, i+1) + td_f^-(i+1)$, $td_f^+(i) = td_d^-(i+1)$
   $td_d^-(k) := c(i, i+1) + td_f^+(i+1)$, $td_f^-(i) = td_d^+(i+1)$
2. make a swap (if available): Then the other player has (at most) two possibilities and chooses the one with the lower cost between 2a. and 2b. (or the only feasible choice), which automatically implies the cost for the decider in i.

2a. move on to vertex $i + 1$:
   $rd_f^+(i) = sw_f(i) + c(i, i+1) + rd_f^+(i+1)$, $rd_d^+(i) := sw_d(i) + rd_d^+(i+1)$
   $rd_f^-(i) = sw_f(i) + c(i, i+1) + rd_f^-(i+1)$, $rd_d^-(i) := sw_d(i) + rd_d^-(i+1)$
   $td_f^+(i) := sw_f(i) + c(i, i+1) + rd_f^+(i+1)$, $td_d^+(i) := sw_d(i) + rd_d^+(i+1)$
   $td_f^-(i) := sw_f(i) + c(i, i+1) + rd_f^-(i+1)$, $td_d^-(i) := sw_d(i) + rd_d^-(i+1)$.

2b. turn around (if possible): Then the players have to go directly back to 0 at cost $\mathrm{rpath}_P(i)$ and then move to ℓ at cost $ad_P^{\pm}(0, i)$ (Case (ii)). Recall that the second parameter of ad, i in our case, indicates a lower bound on the end vertex of a possible backward swap. Since the decider in vertex 0 must be different from D, the feasibility of a turn around in i depends on the number of edges between 0 and i. This yields:
   If i is even:
   $td_d^+(i) := sw_d(i) + \mathrm{rpath}_f(i) + ad_f^-(0, i)$,
   $td_f^+(i) := sw_f(i) + \mathrm{rpath}_d(i) + ad_d^-(0, i)$,
   $td_d^-(i) := \top$, $td_f^-(i) := \top$
   If i is odd:
   $td_d^+(i) := \top$, $td_f^+(i) := \top$
   $td_d^-(i) := sw_d(i) + \mathrm{rpath}_f(i) + ad_d^-(0, i)$,
   $td_f^-(i) := sw_f(i) + \mathrm{rpath}_d(i) + ad_f^-(0, i)$

For the initialization we have to consider the decision in ℓ for Case (i). Note that for Case (ii) a decision was made in some vertex j to turn around. For this decision the cost of the full path from j back to 0 and then to the next cycle and onwards to t had to be taken into account. These costs were included in the initialization of $ad_P^{\pm}$ in (1). For Case (i) we proceed in a similar way: The decider can either move on directly to the next cycle resp. edge in the connection strip or make a swap, either by a swap option available in ℓ from the previous contraction of $G \setminus G'$ or by entering the forward swap. Because of (R2) at most one of the two swap variants can be part of the solution. After a swap the other player has to move on to the next cycle resp. edge. Thus we have:
$rd_d^+(\ell) := \min\{ed_d^2, sw_d + ed_f^1,\, fs_d^+(\ell) + ed_f^1\}$,
$rd_d^-(\ell) := \min\{ed_d^2, sw_d + ed_f^1,\, fs_d^+(\ell) + ed_f^1\}$
The values for the follower are immediately implied by the selection of the minimum.

In the case that a turn around is still possible, i.e. no swap is performed in any vertex between 0 and ℓ, there is also the special case of reaching a swap by traversing the full cycle and repeating the path from 0 to ℓ. Therefore we denote by $fc_P^{\pm}(i)$ (full cycle) the minimal cost to move from i, $i = \ell, \dots, k$, to 0 with possible swaps on the way and then via $1, 2, \dots$ to ℓ without swaps. The parity $\pm$ is aligned with the decider in ℓ. The computation of $fc_P^{\pm}$ is very similar to $fs_P^{\pm}$, but with the addition of the fixed path $0, 1, \dots, \ell$ instead of the return path. Thus we refrain from giving the details. The initialization for $i = k$ has to observe the parity condition to guarantee that the going round the full cycle actually changes the role of the decider in ℓ. We get:
$td_d^+(\ell) := \min\{ed_d^2, sw_d + ed_f^1, fs_d^+(\ell) + ed_f^1, fc_d^+ + ed_f^1\}$,
$td_d^-(\ell) := \min\{ed_d^2, sw_d + ed_f^1, fs_d^+(\ell) + ed_f^1, fc_d^+ + ed_f^1\}$

At the end of these computations we can determine the exit costs for the cycle or edge preceding the current cycle, i.e. the costs of the *spe*-path from v_0 to t. It is easy to see how these exits costs can be propagated along an edge leading from vertex v_0 of the current cycle towards s. The general case of the *spe*-path for the current cycle yields $ed_d^1 := td_d^+(0)$ and $ed_f^1 := td_f^+(0)$. If the path for the current cycle is restricted and thus leaves more freedom of choice for the preceding cycle, we have to compute ed_P^2 by repeating all computations for the current cycle with the restriction that v_0 can be visited only once. This means that Case (ii) and the special case of Case (i), which traverses the full cycle, are simply removed from consideration and the reduced problem is computed as before. We refrain from giving the simple details of this process.

Recall that we also have to consider the mirrored situation where D picks $(0, k)$ as the first edge in the cycle and run all the computations once more with exactly opposite configurations. At the end, we determine each of ed_d^1 and ed_d^2 as the minimum among among the two directions with the obvious consequences for the follower.

For a *directed cactus graph*, i.e. a directed graph which can be obtained from an undirected cactus graph by assigning a direction to each edge, the number of possibilities for the two players to explore a cycle is much more restricted. A cycle may be directed in a cyclic order and thus allow a swap option (in the contraction of branches) or a full cycle in the main part of the algorithm (Case (i), special case), both in a unique way. Or a cycle permits two paths from v_0 to v_ℓ in the main part of the algorithm, which can be settled by simply exploring both options for the decider in v_0. In this case, no swap is possible and such a cycle can be eliminated if it is in $G \setminus G'$. Finally, a cycle may leave only a unique way for its traversal thus being equivalent to a sequence of edges or contain directions that leave it blocked. In all cases, we can find the *spe*-path in polynomial time by a considerably simplified version of the above algorithm.

4.3. Running time

Let $n := |V|$. The overall execution of the algorithm is based on the tree structure inherent in every cactus graph by contracting its cycles. The algorithm first considers all subtrees of this tree lying outside the unique path from s to t and contracts recursively all leaves of these subtrees to their parent vertex. Then the main path is resolved moving from t to s. All together, each cycle is contracted into a constant number (2 or 4) of cost values (swap option or exit costs) containing all the information of previously considered parts of the graph. Thus, the overall running time is given by the sum of running times for the contraction of all cycles in G' and $G \setminus G'$.

Considering the computation for one cycle of the connection strip G', we can observe that each entry of the dynamic programming arrays can be computed in constant time. Entries of the arrays are computed once by moving along certain parts of the cycle which yields on overall linear running time (linear in the number of vertices of the cycle). This also applies to the arrays used in the branch contraction of $G \setminus G'$ performed in Section 4.1. There are two notable exceptions from this property, namely the backward swap $bs_P^{\pm}(i,j)$ and the alternative path $ad_P^{\pm}(i,j)$ which have a quadratic number of entries to be computed, each of them in constant time. Thus, we have the following statement.

Theorem 4. *The* spe-*path of* Shortest Path Game *on undirected cactus graphs can be computed in $O(n^2)$ time.*

Since backward swaps do not apply for directed graphs we can state immediately.

Corollary 5. *The* spe-*path of* Shortest Path Game *on directed cactus graphs can be computed in $O(n)$ time.*

4.4. More general classes of graphs and some observations

The main reason why Shortest Path Game is solvable in polynomial time on cactus graphs, which have treewidth 2, is the fact that optimal solutions of subgraphs can be used for deriving an optimal solution of the whole graph. In this section we give a simple example showing that this does not work anymore for outerplaner graphs, which still have treewidth 2 and are a superclass of cactus graphs: In Figure 10 the path corresponding to the optimal strategy is illustrated by giving edges chosen by player 1 the color red and its opponent blue. The *spe*-path has length $(2,1)$ and goes through vertices a and b. However, when restricting the problem to the subgraph $G \setminus \{s,t\}$ (framed by the dashed line in the third picture of Figure 10) a *spe*-path from a to b with player 2 starting in a consists of simply choosing edge (a,b) with length $(0,0)$. If however player 2 would choose this edge when playing the game on the whole graph, player 1 would choose the edge of length 1 forcing player 2 to use the edge of length M in order to reach t. Thus, the connection from a to b traversed in the global *spe*-path is different from the local subgame perfect equilibrium path from a to b.

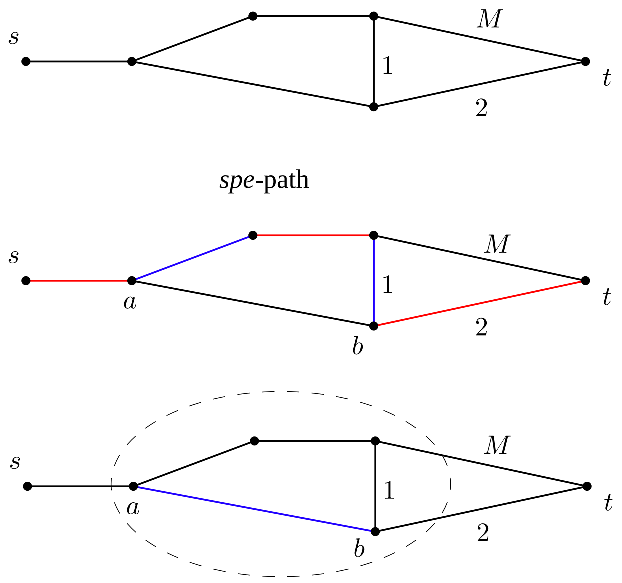


Figure 10: Example where the connection from a to b used in the global *spe*-path from s to t is disjoint form the local *spe*-path from a to b. Unlabeled edges have cost 0.

For narrowing the gap between the positive result for cactus graphs (having treewidth 2) and the negative result of general (and bipartite) graphs, it would be interesting to consider graphs of bounded treewidth, as it was done for Geography. However, a potential PSPACE-completeness result for Shortest Path Game on bounded treewidth graphs along the lines of the proof of Theorem 3 does not seem to be within reach. To be in line with that proof, we would need a restricted variant of Quantified 3-Sat satisfying the property of having *respectful* bounded treewidth (cf. Atserias and Oliva [1]) in order to achieve bounded treewidth of the graph used. However, Atserias and Oliva [1] note that bounded treewidth quantified boolean formula become polynomial time solvable when restricted to *respectful* bounded treewidth instances by a result of Chen and Dalmau [3]. Thus, a potential PSPACE-completeness result for Shortest Path Game on bounded treewidth graphs will likely require a different approach; the computational complexity of Shortest Path Game on bounded treewidth graphs remains an interesting open question.

Acknowledgements

We would like to thank Christian Klamler (University of Graz) for fruitful discussions and valuable comments.

Ulrich Pferschy and Joachim Schauer were supported by the Austrian Science Fund (FWF): [P 23829-N13]. Andreas Darmann was supported by the Austrian Science Fund (FWF): [P 23724-G11].

References

[1] Albert Atserias and Sergi Oliva. Bounded-width QBF is PSPACE-complete. *Journal of Computer and System Sciences*, 80(7):1415 – 1429, 2014.

[2] H. Bodlaender. Complexity of path-forming games. *Theoretical Computer Science*, 110(1):215–245, 1993.

[3] H. Chen and V. Dalmau. Decomposing quantified conjunctive (or disjunctive) formulas. In *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012*, pages 205–214, 2012.

[4] A. Darmann, C. Klamler, and U. Pferschy. Sharing the cost of a path. *to appear in: Studies in Microeconomics*, 2014. available at: `http://ssrn.com/abstract=2287875`.

[5] A. Darmann, G. Nicosia, U. Pferschy, and J. Schauer. The subset sum game. *European Journal on Operational Research*, 233:539–549, 2014.

[6] A. Darmann, U. Pferschy, and J. Schauer. The shortest path game: Complexity and algorithms. In *Proceedings of Theoretical Computer Science (TCS 2014), Rome*, volume 8705 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2014.

[7] A.S. Fraenkel and E. Goldschmidt. PSPACE-hardness of some combinatorial games. *Journal of Combinatorial Theory*, 46(1):21–38, 1987.

[8] A.S. Fraenkel and S. Simonson. Geography. *Theoretical Computer Science*, 110(1):197–214, 1993.

[9] A.S. Fraenkel, E.R. Scheinerman, and D. Ullman. Undirected edge geography. *Theoretical Computer Science*, 112(2):371–381, 1993.

[10] D. Lichtenstein and M. Sipser. Go is polynomial-space hard. *Journal of the ACM*, 27(2):393–401, 1980.

[11] M.J. Osborne. *An Introduction to Game Theory.* Oxford University Press, USA, 2004.

[12] T.J. Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978.

[13] L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th Symposium on Theory of Computing, STOC '73*, pages 1–9. ACM, 1973.