Discrete Applied Mathematics 217 (2017) 3–18

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

On the Shortest Path Game☆

Andreas Darmann [a], Ulrich Pferschy [b,*], Joachim Schauer [b]

[a] Institute of Public Economics, University of Graz, Universitaetsstrasse 15, 8010 Graz, Austria

[b] Department of Statistics and Operations Research, University of Graz, Universitaetsstrasse 15, 8010 Graz, Austria

ARTICLE INFO

Article history:
Received 19 September 2014
Received in revised form 3 June 2015
Accepted 5 August 2015
Available online 28 August 2015

Keywords:
Shortest path problem
Game theory
Computational complexity
Cactus graph

ABSTRACT

In this work we address a game theoretic variant of the shortest path problem, in which two decision makers (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 the decision problem associated with 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.

1. Introduction

We consider the following game on a graph: There is a directed graph $G = (V, A)$ given 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 A and B who have full knowledge of the graph. They 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.

☆ 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].

* Corresponding author.
E-mail addresses: andreas.darmann@uni-graz.at (A. Darmann), pferschy@uni-graz.at (U. Pferschy), joachim.schauer@uni-graz.at (J. Schauer).

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

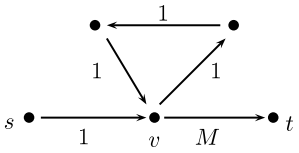
http://dx.doi.org/10.1016/j.dam.2015.08.003
0166-218X/

We will impose two restrictions on the setting in order to ensure that vertex t is in fact reachable and to guarantee finiteness of the game. 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. Clearly, this would imply cycles of even length. Such cycles, however, may cause the game to be infinite as illustrated by the following example.

Example 1. 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.



The general setting of the game is representable as a game in extensive form. However, 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 [14], ch. 16). Thus, we have to impose reasonable conditions to guarantee finiteness of the game.

A straightforward idea would be to restrict the game to *simple paths* and request that each vertex may be visited at most once. While this restriction would lead to a simpler setting of the game, it would also rule out very reasonable strategic behavior, such as going through a cycle in the graph. Indeed, a cycle of even length cannot be seen as a reasonable choice for any player since it necessarily increases the total costs for both players. However, in Example 1 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 which permits a rather general setting of the game:

(R2) The players cannot 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. This aspect of allowing odd cycles will be taken explicitly into account in the presented algorithms. 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 player.

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 for both players remain below given bounds (see Section 2). Clearly, such an upper bound permits a clear answer to problems such as Example 1. 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. Anticipating this outcome, the unfortunate player will 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.

As an illustrative example of the game consider the following scenario. Two scientists A and B meet at a conference and lay out the plan for a joint paper. This requires the completion of a number of tasks. Each task requires a certain working time (identical for both A and B) to be completed. Moreover, there is a precedence structure on the set of tasks, e.g. the notation and definitions have to be finished before the formal statement of results, which in turn should precede the writing of the proofs, etc. Obviously, this implies a partial ordering on the set of tasks. We can represent any state of the paper by a vertex of a directed graph with an arc representing a feasible task whose completion leads to a new state (= vertex).

To avoid confusion and a mix-up of file versions the two scientists decide to alternate in working on the paper such that each of them finishes a task and then passes the file to the other co-author. Since none of them wants to patronize the other they have no fixed plan on the division of tasks, but each can pick any outstanding task as long as it is a feasible step to a new state of the paper. They continue in this way until the paper is finished, i.e. the final state is reached. Since both scientists are extremely busy each of them tries to minimize the time devoted to the paper taking into account the anticipated rational optimal decisions by the other.

More generally, the considered game applies to situations in which a system needs to be transformed from state s to state t by traversing several transitional states in alternately performed steps.

1.1. Solution concept

In classical game theory Shortest Path Game is 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.

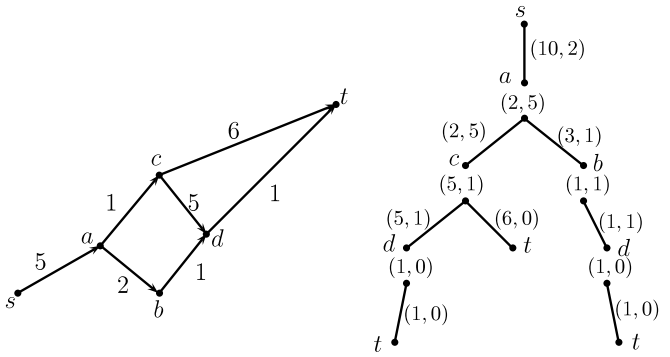
The standard procedure to determine equilibria in a game tree is *backward induction* (see Osborne [14], ch. 5). This means that for each node in the game tree, whose child nodes are all leaves, the associated player can reach a decision by simply choosing the best of all child nodes w.r.t. their allocated 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.

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 2. 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.



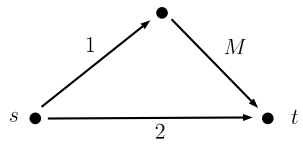
Example 2. Consider the above graph and the associated game tree. The *spe*-path is determined by backward induction and represented by ordered pairs of cost values (x, y) meaning that the decider in a given vertex has to pay a total value of x whereas the opponent has to pay a value of y.

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.

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.

Comparing the outcome of our game, i.e. the total cost of the *spe*-path, with the cost of the shortest path obtained by a cooperative strategy we can consider the *Price of Anarchy* (cf. [13]) defined by the ratio between these two values. However, it is easy to see that the Price of Anarchy can become arbitrarily large, e.g. in the following example, where the ratio is $\frac{1+M}{2}$.

[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. $1, \ldots, n$, we choose the path to a vertex with lowest vertex number.



1.2. Related literature

A closely related game is known as Geography (see Schaefer [15]). 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 [15] already showed PSPACE-completeness of Geography. Lichtenstein and Sipser [12] 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 [11] gave polynomial time algorithms for Geography and Vertex Geography when played on directed acyclic graphs. In Fraenkel et al. [10] 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 Vertex Geography is linear time solvable on both directed and undirected graphs of bounded treewidth. For 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. [5] 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. [6].

1.3. 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 follows from topological sort (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. [7].

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 [12]). 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.

Starting from a copy of G we create a new graph H for Shortest Path Game as follows: First we assign a cost 1 to every arc $e \in A$. Let M be a large number, e.g. $M := |A| + 1$. Then we introduce a new vertex t which we color red, and an arc of weight M from each green vertex to t. Next, introduce a green vertex z and an arc of weight M from each red vertex to z. Finally, introduce an arc of cost 1 from z to t.

The constants C_A and C_B are set to $C_A = 2$ 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 ≤ 2 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 1, then B will go to z paying M, and A from z to t at cost 1. □

Note that the result of Theorem 1 also follows from Theorem 3 by 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 [12] 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 any directed acyclic graph we can determine in $O(|A|)$ time a *topological sort*, i.e. a linear ordering of the vertices, such that all arcs point "from left to right" (see e.g. [4, ch. 22.4]). W.l.o.g. we can assume that s and t are the first and last vertex in this ordering.

For each vertex $v \in V$ we define the following two dynamic programming arrays:

$p_d(v)$: 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.

For each vertex v let $S(v) := \{u \mid (v, u) \in A\}$ be the set of successors of v. After initializing $p_d(t) = p_f(t) = 0$, the algorithm goes through the remaining vertices in reverse order of the topological sort and computes for every vertex v:

$$u' := \arg\min\{c(v, u) + p_f(u) \mid u \in S(v)\}$$
$$p_d(v) := c(v, u') + p_f(u'), \qquad p_f(v) := p_d(u')$$

The correctness of the computation follows immediately from the structure of the topological sort since all vertices in $S(v)$ are processed before v. Without elaborating on a formal proof we state:

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 [16]).

Definition (Quantified 3-Sat)**.** :

GIVEN: Set $X = \{x_1, \ldots, x_n\}$ of variables and a quantified Boolean formula

$$F = (\exists x_1)(\forall x_2)(\exists x_3) \cdots (\forall x_n)\phi(x_1, \ldots, x_n)$$

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 $C_1, \ldots, C_m$ denote the clauses that make up ϕ, i.e., $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \ldots C_m$, 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 [9]): There are two players (the existential- and the universal-player) moving alternately, starting with the existential-player. The ith 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. [10] 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 Fig. 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 Fig. 1 we assume $C_1 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ and $C_2 = (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_n)$.

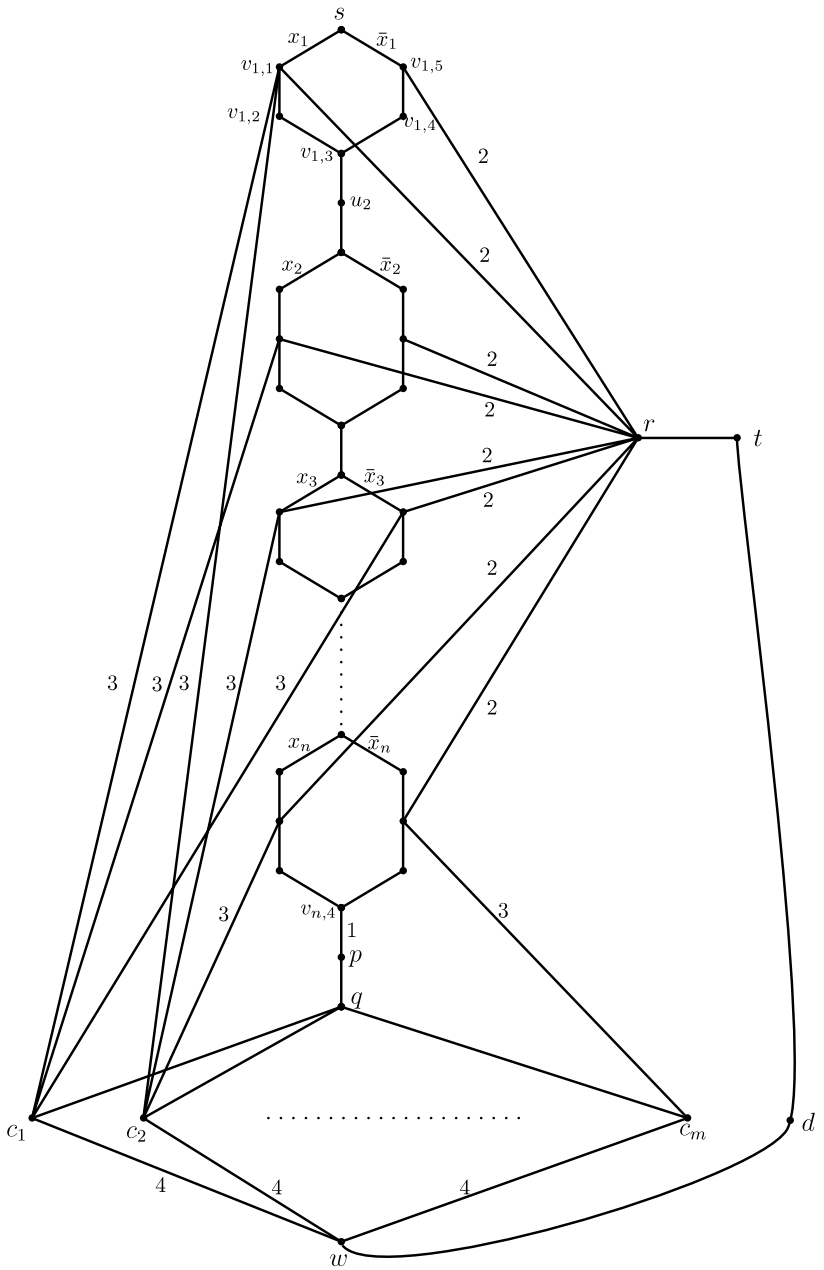


Fig. 1. Undirected graph G of instance $\mathcal{I}$ constructed for an instance $\mathcal{Q}$ of Quantified 3-Sat with $C_1 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ and $C_2 = (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_n)$.

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 from w have cost 4. The edge $\{r, t\}$ has cost $c(\{r, t\}) = 0$, while each other edge emanating from r has cost 2. The edge $\{v_{n,4}, p\}$ has cost $c(\{v_{n,4}, p\}) = 1$. 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 3. 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 $\mathcal{I}$ we set $C_A = 0$ and $C_B = 2$, 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 $\Leftrightarrow$ *$\mathcal{I}$ 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.

[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 $\varepsilon > 0$.

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 2, or (ii) edge $\{v_{1,1}, v_{1,2}\}$ with zero cost, or (iii) an edge of cost 3 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* $c(B) = 2$.

As a consequence of the outcome in Case I, B will not choose (iii), because B would end with $c(B) \geq 3$ 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 3, 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 2. Clearly, A then chooses edge (r, t), and again *the game ends with* $c(A) = 0$ *and* $c(B) = 2$.

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* $c(B) = 2$ 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 3 (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* $c(B) = 2$.

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 $c(B) = 2$.

(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 $c(B) = 2$. Assume B moves along $\{v_{n,4}, p\}$ with cost 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 4 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* $c(A) = 4$ *and* $c(B) = 1$.

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 3. 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 2. Clearly, B chooses the latter edge, since in vertex r player A will move along $\{r, t\}$. *Thus, the game ends with* $c(A) = 3$ *and* $c(B) = 1 + 2 = 3$.

Summing up the two cases we have:

If $\mathcal{Q}$ is a "yes"-instance then it is better for B to move along an edge towards r before reaching p, resulting in an outcome with $c(A) = 0$ and $c(B) = 2$.

If $\mathcal{Q}$ is a "no"-instance there is a clause which is not satisfied by ϕ and B will move along $\{v_{n,4}, p\}$ resulting in $c(A) = 4$ and $c(B) = 1$. □

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.

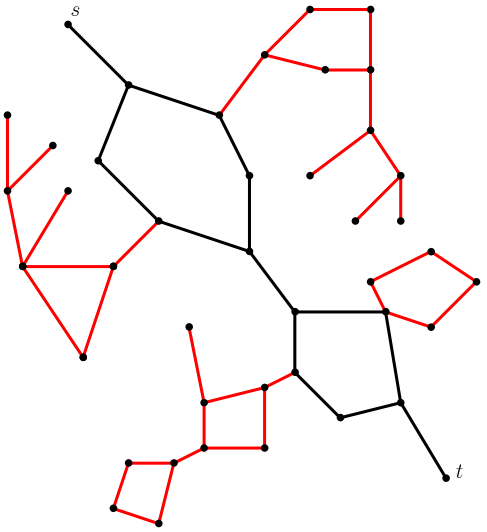


Fig. 2. Graph G with connection strip G' (in black) and branches $G \setminus G'$ (in red). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

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 players 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 graphs, 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 defines 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 Fig. 2). In the *spe*-path vertices in $G \setminus G'$ could be included only to change the role of the decision 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 Fig. 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 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. Then the best options for moving from the "entry" of this cycle to the exit are computed. These iterations are continued in a bottom-up way until the starting vertex s is reached.

The required partial solutions will be determined by a number of dynamic programming arrays which store the best options for traversing certain segments of the graph both for the decider and the follower in each vertex. The resulting

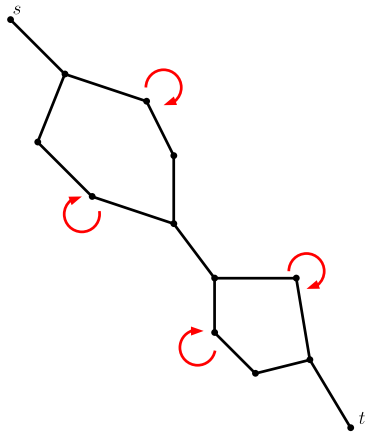


Fig. 3. Graph G with connection strip G': branches $G \setminus G'$ are contracted into swaps.

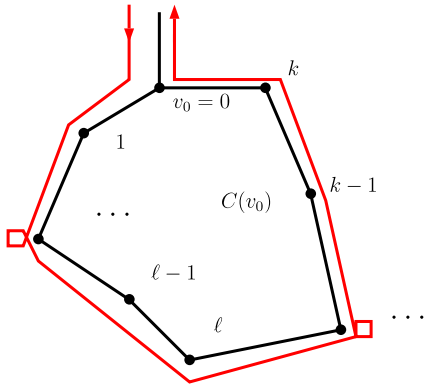


Fig. 4. Cycle $C(v_0)$ with a possible path (in red) going round the cycle (with possible swaps on the way). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

algorithm involves tedious technicalities. To provide the reader with a comprehensive guideline without elaborating too many computational details which are relevant only for any actual implementation we will describe the more compact first part (contraction of branches in Section 4.1) and the subtle difficulties of one configuration of the second step (Section 4.2) in detail but defer the more detailed description of all dynamic programming arrays and operations to an accompanying technical report [8].

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 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 $0, 1, 2, \ldots, k-1, k, 0$.

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 Fig. 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 Fig. 5. In the latter case, (R2) implies that no additional swaps in vertices $1, \ldots, \ell - 1$ (resp. $k, k-1, \ldots, \ell+1$) are allowed. 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. Starting with the edge $(0, k)$ we get another two configurations by mirroring. The final swap costs $sw(v_0)$ are given by the cheaper alternative.

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 $P = d$ ($P = f$) signifies that the cost occurs for the player deciding (following) 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 $\top$.

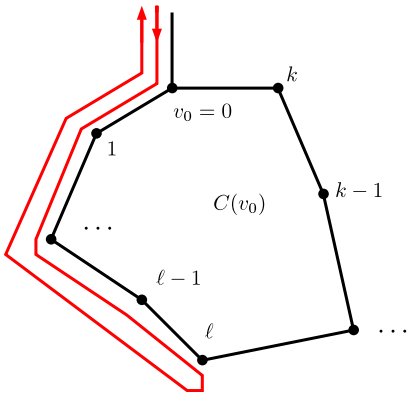


Fig. 5. Cycle $C(v_0)$ with a possible path (in red) using a swap in vertex ℓ. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

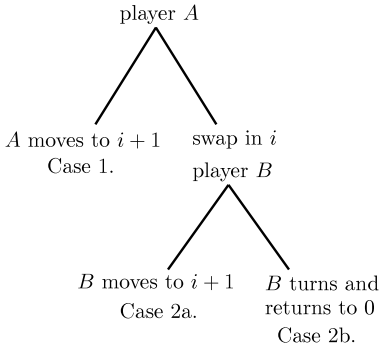


Fig. 6. Decision tree for player A deciding in vertex i.

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 $i+1, i+2, \ldots, k, 0$, 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. We state the appropriate update recursion for the case where D chooses $(0, 1)$ as a first edge (the symmetric case starting in the other direction is completely analogous). Moving backwards along the resulting path we settle the minimal costs for vertices $k, k-1, \ldots, 1$.

In any vertex i the decision process has at most three outcomes as illustrated in Fig. 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:
   $tc_d^+(i) := sw_d(i) + \text{path}_f(i), tc_f^+(i) := sw_f(i) + \text{path}_d(i)$

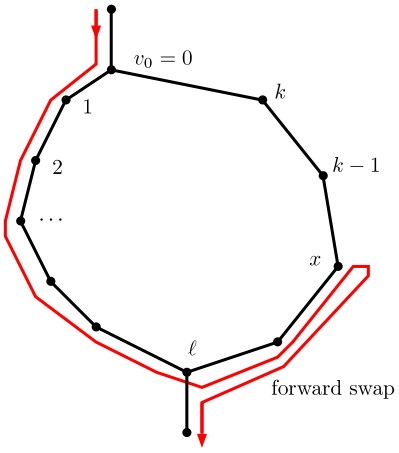


Fig. 7. Case (i): Forward swap in x.

$tc_d^-(i) := \top, tc_f^-(i) := \top$
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.

For the initialization of the arrays for $i = k$, we will for simplicity assume that vertex k has no swap option (see [8] for the general case) and set:

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

4.2. Main part of the algorithm

In the main part of the algorithm we 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'. Traversing the connection strip in a bottom up way starting in t and moving upwards in direction of s, we focus on the computation of an optimal subpath for one cycle. 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 $0, 1, \ldots, \ell, \ell+1, \ldots, k, 0$.

Assuming that it is decided in 0 to take the edge $(0, 1)$ there are two main possibilities for the path from 0 to ℓ and onwards. Starting with edge $(0, k)$ we would get a completely symmetric mirrored situation, and the decider will finally take the better of the two options.

Case (i): The two players 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 Fig. 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 Fig. 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 Fig. 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, i.e. no vertex can be traversed more than twice.

Computing the details of the two cases requires fairly involved dynamic programming steps. Since the general technique was already illustrated in detail in Section 4.1 we will only go into the details of the more interesting Case (ii) and point to our technical report [8] for all further details.

We process the current cycle to determine the *spe*-path starting in v_0. Keeping the notational system introduced above we will introduce the following dynamic programming arrays (considering only Case (ii)). The parity $\pm = +$ indicates that the decider in i is the same as the decider in ℓ.

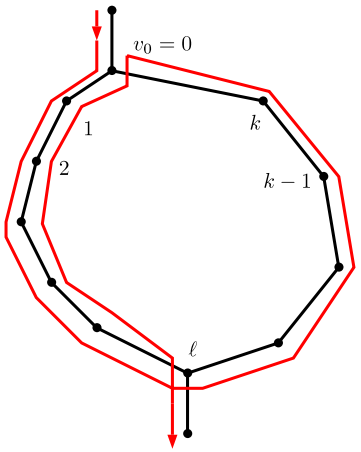


Fig. 8. Case (i), special case: Traverse the full cycle.

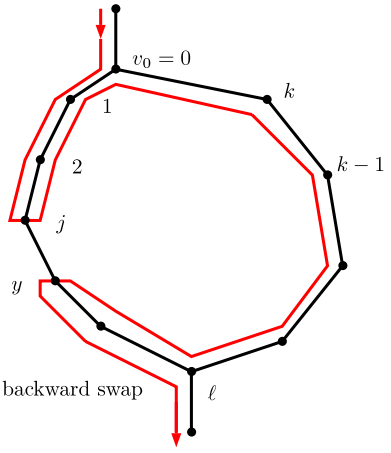


Fig. 9. Case (ii): Backward swap in y.

$td_P^{\pm}(i)$: minimal cost to move from i, $i = 1, \ldots, \ell - 1$, to ℓ and exit the cycle, if a turn around (according to Case (i)) is still possible.

$ad_P^{\pm}(i)$: ("alternative path") minimal cost for moving from i, $\ell \leq i \leq k$, via $i-1, i-2, \ldots$ 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).

$bs_P^{\pm}(i, j)$: ("backward swap") minimal cost for moving from $i, j < i \leq \ell$, without swaps to some vertex $y, j < y \leq i$, make a swap in y and return to ℓ. 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 $j, j = 1, \ldots, \ell - 1$.

We first compute the cost of a backward swap. Let $\text{bpath}_P(i)$ denote the cost of the path from i towards ℓ. For all values of $j = 1, \ldots, \ell - 1$ we perform the following computations and 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 Case 2 above for $i = j'$.

Next we determine the path from 0 via k, $k-1, \ldots$ to ℓ, when no turn arounds are possible any more. We consider the decision in vertex i, $i = \ell+1, \ldots, k$, where 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. Let ed_P (exit costs) denote the costs of the *spe*-path from v_ℓ to t known from previous computations. 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. The values for the follower are immediately implied by the selection of the minimum.

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

Now we can determine the value of the main arrays for all values of i by moving backwards from $i = \ell - 1$ down to $i = 1$. We let $\text{rpath}_P(i)$ denote the cost of the return path from i towards 0. Let $D \in \{A, B\}$ again be the decision maker in 0. In any vertex i the decision process has at most three outcomes, from which the cheapest is chosen. The omitted details are analogous to the problem of contracting a cycle.

1. move on along the cycle to vertex $i+1$.
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.

2a. move on to vertex $i+1$.

2b. turn around (if possible): Then the players have to go directly back to 0 at cost $\text{rpath}_P(i)$ and then move to ℓ at cost $ad_P^{\pm}(0, i)$. 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:
   $td_d^+(i) := sw_d(i) + \text{rpath}_f(i) + ad_f^-(0, i),$
   $td_f^+(i) := sw_f(i) + \text{rpath}_d(i) + ad_d^-(0, i),$
   $td_d^-(i) := \top, td_f^-(i) := \top$
   If i is odd:
   $td_d^+(i) := \top, td_f^+(i) := \top$
   $td_d^-(i) := sw_d(i) + \text{rpath}_f(i) + ad_d^-(0, i),$
   $td_f^-(i) := sw_f(i) + \text{rpath}_d(i) + ad_f^-(0, i).$

At the end of these computations we can determine the costs of the *spe*-path from v_0 to t. The general case of the *spe*-path for the current cycle yields $ed_d := td_d^+(0)$ and $ed_f := td_f^+(0)$.

Note that the combination of the solution for the current cycle with the previously computed *spe*-path from v_ℓ to t requires special care. In particular, we have to consider the case that v_ℓ is also the upper vertex of the next cycle (in direction of t). In such a situation, only certain configurations for passing through the "upper" and the "lower" cycle are feasible to avoid that v_ℓ is traversed more than twice. This means that for each cycle we have to consider two cases, namely that the path is traversing v_ℓ once or twice. For simplicity of presentation we stick to the latter (more complicated) case and ignore the case distinction, which can be found in [8].

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.

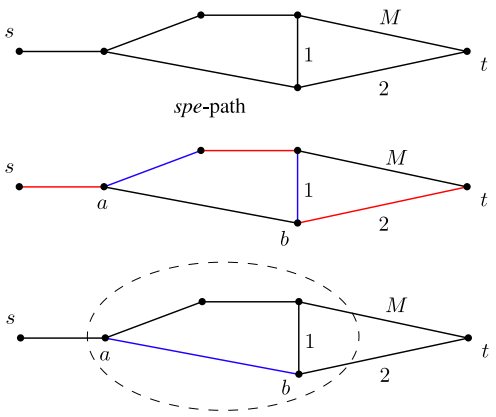


Fig. 10. Example where the connection from a to b used in the global *spe*-path from s to t is disjoint form the local *spe*-path from a to b. Unlabeled edges have cost 0. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

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 outerplanar graphs, which still have treewidth 2 and are a superclass of cactus graphs: In Fig. 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 Fig. 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.

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.

Acknowledgments

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

We would also like to thank the anonymous referees for their comments which helped a lot to improve the presentation of the paper.

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, Sergi Oliva, Bounded-width QBF is PSPACE-complete, J. Comput. System Sci. 80 (7) (2014) 1415–1429.
[2] H. Bodlaender, Complexity of path-forming games, Theoret. Comput. Sci. 110 (1) (1993) 215–245.
[3] H. Chen, V. Dalmau, Decomposing quantified conjunctive (or disjunctive) formulas, in: Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, 2012, pp. 205–214.
[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, third ed., MIT Press, 2009.
[5] A. Darmann, C. Klamler, U. Pferschy, Sharing the cost of a path, Stud. Microecon. 3 (2015) 1–12.
[6] A. Darmann, G. Nicosia, U. Pferschy, J. Schauer, The subset sum game, Eur. J. Oper. Res. 233 (2014) 539–549.
[7] A. Darmann, U. Pferschy, J. Schauer, The shortest path game: Complexity and algorithms, in: Proceedings of Theoretical Computer Science, TCS 2014, Rome, in: Lecture Notes in Computer Science, vol. 8705, Springer, 2014, pp. 39–53.
[8] A. Darmann, U. Pferschy, J. Schauer, On the Shortest Path Game: Extended Version. Technical Report, University of Graz, Department of Statistics and Operations Research, 2015, available as: arXiv:1506.00462.
[9] A.S. Fraenkel, E. Goldschmidt, PSPACE-hardness of some combinatorial games, J. Combin. Theory 46 (1) (1987) 21–38.
[10] A.S. Fraenkel, E.R. Scheinerman, D. Ullman, Undirected edge geography, Theoret. Comput. Sci. 112 (2) (1993) 371–381.
[11] A.S. Fraenkel, S. Simonson, Geography, Theoret. Comput. Sci. 110 (1) (1993) 197–214.
[12] D. Lichtenstein, M. Sipser, Go is polynomial-space hard, J. ACM 27 (2) (1980) 393–401.
[13] N. Nisan, T. Roughgarden, E. Tardos, V.V. Vazirani (Eds.), Algorithmic Game Theory, Cambridge University Press, 2007.
[14] M.J. Osborne, An Introduction to Game Theory, Oxford University Press, USA, 2004.
[15] T.J. Schaefer, On the complexity of some two-person perfect-information games, J. Comput. System Sci. 16 (2) (1978) 185–225.
[16] L.J. Stockmeyer, A.R. Meyer, Word problems requiring exponential time, in: Proceedings of the 5th Symposium on Theory of Computing, STOC'73, ACM, 1973, pp. 1–9.