Each edge has a cost $(x,y)$, whereby Player $i$ wants to minimize component $i$ (not zero-sum)

# The Shortest Path Game: Complexity and Algorithms[⋆]

Andreas Darmann[1], Ulrich Pferschy[2], and Joachim Schauer[2]

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

[2] University of Graz, Department of Statistics and Operations Research, 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 of the game. On the other hand, we can give polynomial time algorithms for directed acyclic graphs and for cactus graphs 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.

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

## 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

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

Not zero
sum

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 $t$.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.

To avoid that the players get stuck at some point, 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 $t$.

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

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, 2004, 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.[1]

The outcome, if both players follow this strategy, is a unique path from $s$ to $t$ in $G$ corresponding to the unique subgame perfect equilibrium (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.

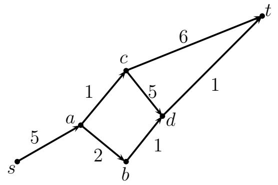
Clearly, such a *spe*-path path can be computed in exponential time by exploring the full game tree. It it is the main goal of this paper to study the complexity

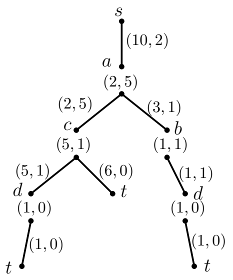
[1] In order to guarantee a unique solution of such a game and thus a specific subgame perfect Nash equilibrium, we have to define a tie-breaking rule. We will use the "optimistic case", where in case of indifference a player chooses the option with lowest possible cost for the other player. If both 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.

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 in polynomial time.

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 subpath of the *spe*-path on all positions.

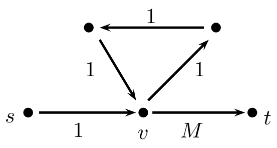
Consider the following example: The *spe*-path is determined by backward induction and represented by ordered pairs of cost values $(x, y)$ meaning that the decider in a given vertex has to pay a total value of $x$ whereas the follower has to pay a value of $y$.





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. 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, 2004, 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 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.

## 1.1 Related Literature

A closely related game is known as GEOGRAPHY (see Schaefer (1978)). 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 (1978) already showed PSPACE-completeness of GEOGRAPHY. Lichtenstein and Sipser (1980) 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 (1993) gave polynomial time algorithms for GEOGRAPHY and VERTEX GEOGRAPHY when played on directed acyclic graphs. In Fraenkel et al. (1993) 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 (1993) 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. (2013) 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. (2014).

### 1.2 Our Contribution

We introduce the concept of *spe*-path resulting from backward induction in a game tree with full 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. Using an advanced auxiliary data structure, its running time can be bounded by $O(n \log n)$.

## 2 Spe-Paths for Shortest Path Game on Directed Graphs

The following decision problem basically asks whether the unique subgame perfect equilibrium remains below given cost bounds $C_A$ and $C_B$.

Shortest Path Game DEC:
Given a weighted graph $G$ and two positive values $C_A$, $C_B$, does the *spe*-path in the above game tree yield costs $c(A) \leq C_A$ and $c(B) \leq C_B$ ?

### 2.1 PSPACE-completeness

The PSPACE-completeness of Shortest Path Game on general graphs can be shown by constructing an instance of Shortest Path Game, 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 DEC *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 (1980)). For a given instance of Vertex Geography we construct an instance of Shortest Path Game, such that the *spe*-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 DEC as follows: First we assign a cost $\varepsilon$ 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 $\varepsilon$ 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 $\varepsilon$, then $B$ will go to $z$ paying $M$, and $A$ from $z$ to $t$ at cost $\varepsilon$. $\square$

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.

## 2.2 Directed Acyclic Graphs

If the underlying graph $G$ is acyclic we can devise 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 $v$:

$d_1(v)$: minimal cost to go from $v$ to $t$ for the player deciding at $v$.
$d_2(v)$: minimal cost to go from $v$ to $t$ for the player **not** deciding at $v$.

The two arrays are initialized as follows:
$d_1(t) = d_2(t) = 0,\ \ d_1(v) = d_2(v) = \infty$ for all $v \in V, v \neq t$.

**Algorithm 1.** Optimal strategies of the shortest path game on acyclic graphs

```
1: repeat
2:     find $v \in V$ with $d_1(v) = \infty$ such that $d_1(u) \neq \infty$ for all $u \in S(v)$
3:     let $u' := \arg\min\{c(v,u) + d_2(u) \mid u \in S(v)\}$
4:     $d_1(v) := c(v,u') + d_2(u')$
5:     $d_2(v) := d_1(u')$
6: until $d_1(s) \neq \infty$
```

Starting from the destination vertex $t$ and moving backwards in the graph we iteratively compute the values of $d_i(v)$. Note that each such entry is computed only once and never updated later. In each iteration of the **repeat**-loop one entry $d_1(v)$ is reduced from $\infty$ 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 $d_1(v) = \infty$, then also $d_1(s) \neq \infty$ and we would have stopped the algorithm before. If for all vertices $v \in V$ with $d_1(v) = \infty$ there exists a vertex $u \in S(v)$ with $d_1(u) = \infty$, then we could apply the same argument to $u$. Thus, also $u$ has a successor $u_s$ with $d_1(u_s) = \infty$. 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. Taking a closer look at the running time details we can state with same additional effort:

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

## 3 Spe-Paths for Shortest Path Game on Undirected Graphs

We will provide a reduction from Quantified 3-Sat which is known to be PSPACE-complete (Stockmeyer and Meyer (1973)).

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

> GIVEN: Set $X = \{x_1, \ldots, x_n\}$ of variables and a quantified Boolean formula $F = (\exists x_1)(\forall x_2)(\exists x_3) \ldots (\forall x_n)\, \phi(x_1, \ldots, x_n)$ where $\phi$ 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 $\phi$, 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 (1987)): 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 $\phi$ true.

### 3.1 PSPACE-completeness

The following result shows a notable difference between Shortest Path Game and Geography, since Fraenkel et al. (1993) showed that Geography is polynomially solvable on undirected bipartite graphs (while PSPACE-complete on general undirected graphs).

**Theorem 3.** Shortest Path Game DEC *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 the figure below):

- 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 the figure below we assume $C_1 = (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ and $C_2 = (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_n)$.

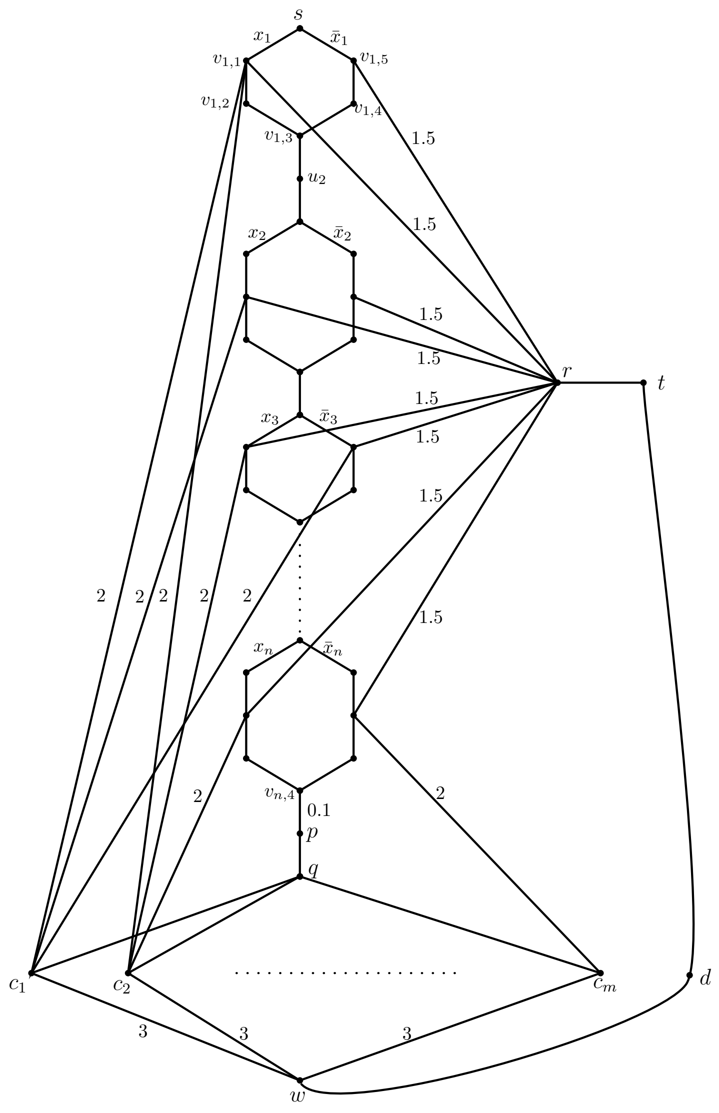
Finally, we define the edge costs.[2] We start with the edges emanating from vertex $w$: The cost of edge $\{w, d\}$ is $c\{w, d\} = 0$, all other edges emanating from $w$ have cost 3. The edge $\{r, t\}$ has cost $c(\{r, t\}) = 0$, while each other edge emanating from $r$ has cost 1.5. The edge $\{v_{n,4}, p\}$ 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 $\mathcal{S}$ the players $A, B$ try to find a path from $s$ to $t$ in the graph $G$, each one aiming at minimizing her own total cost.

**Claim.** $\mathcal{Q}$ is a "yes"-instance of Quantified 3-Sat $\Leftrightarrow$ Instance $\mathcal{S}$ of Shortest Path Game ends with $A$ having to carry zero cost.

**Proof of Claim.** omitted.

[2] 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 $\varepsilon > 0$.



## 4 Shortest Path Game on Undirected Cactus Graphs

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.

A *cactus graph* 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. Considering Shortest Path Game, it is easy to see that the union of all simple paths from $s$ to $t$ constitutes a unique sequence of edges (which are bridges of

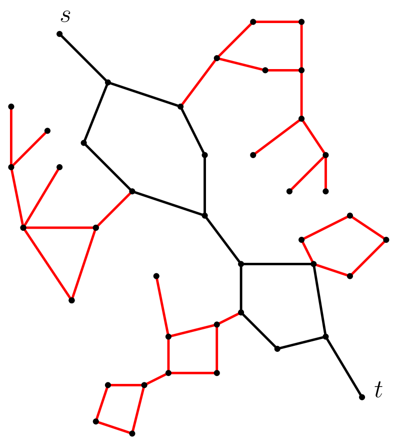


**Fig. 1.** Graph $G$ with connection strip $G'$ (in black) and branches $G \setminus G'$ (in red)

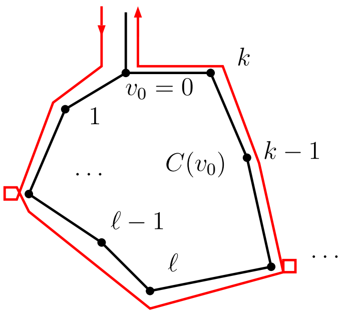
the graph) and simple cycles which defines a subgraph $G'$, i.e. the *connection strip* between $s$ and $t$. All other vertices of the graph are "dead end streets", i.e. edges and cycles branching off from $G'$ (see Figure 1). In the *spe*-path vertices in $G \setminus G'$ could be included only to change 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 $v \in G'$ 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).

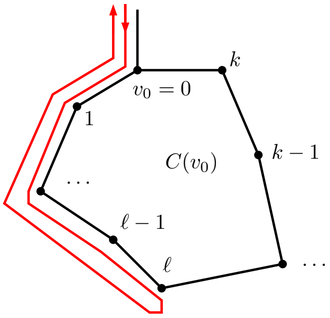
Our algorithm will first compute these swap costs by recursively traversing the components of $G \setminus G'$ in Section 4.1. Then, in the second step, the *spe*-path in $G'$ is computed by moving backwards from $t$ towards $s$ in Section 4.2. Because of the space restriction, we will only describe the contraction of branches in algorithmic detail to give the reader a flavor of the required computation. The main part of the algorithm is sketched only briefly. Its details are quite involved and follow in some sense the technique described in Section 4.1.

## 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) (see left figure below). 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 $\ell$ and go back to 0 (see right figure below). 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 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 but 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 use a generic notation for dynamic programming arrays, where $d_P^{\pm}(i)$ denotes the cost of a certain path starting in vertex $i$ and ending in a fixed specified vertex, with 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. 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. $\pm = -$. We use cost $\top$ if a path is infeasible.

Following this system we use:
$tc_P^{\pm}(i)$ : 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.

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

The decision tree in any vertex $i$ is depicted in Figure 2.

1. move on along the cycle to vertex $i+1$:
   $rc_d^+(i) := c(i,i+1) + rc_f^-(i+1)$, $rc_f^+(i) = rc_d^-(i+1)$
   $rc_d^-(k) := c(i,i+1) + rc_f^+(i+1)$, $rc_f^-(i) = rc_d^+(i+1)$

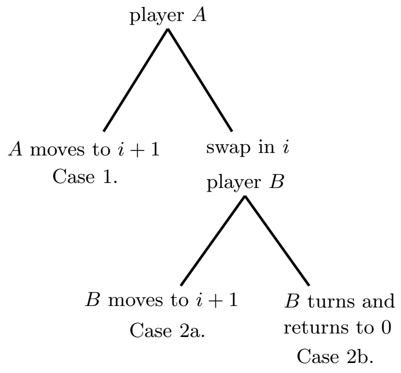


**Fig. 2.** Decision tree for player $A$ deciding in vertex $i$

$tc_d^+(i) := c(i, i+1) + tc_f^-(i+1)$, $tc_f^+(i) = tc_d^-(i+1)$
$tc_d^-(k) := c(i, i+1) + tc_f^+(i+1)$, $tc_f^-(i) = tc_d^+(i+1)$

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 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. Hence, the decision maker in vertex $i$ chooses the minimum between case 1. and case 2. 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. We get:

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

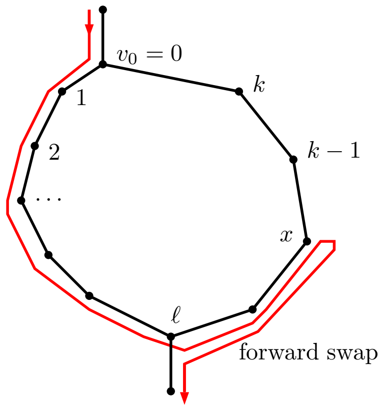
## 4.2 Main Part of the Algorithm

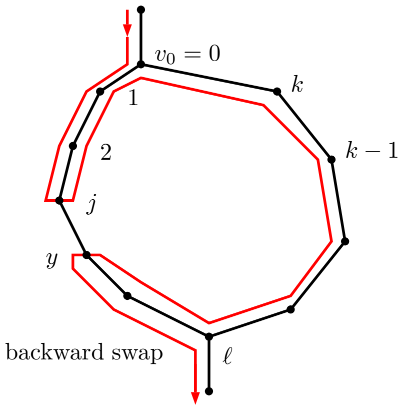
In the main part of the algorithm we determine the *spe*-path from $s$ to $t$ along the connection strip $G'$ after contracting the reminder of the graph into swap options in $G'$. We traverse the connection strip backwards starting in $t$ and moving up towards $s$. In the following we will focus on the computation of a *spe*-path 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" $v_\ell$ through which all paths connecting the cycle with $t$ have to leave the cycle.

If the decider in 0 takes the edge $(0,1)$ there are two main possibilities for the path from 0 to $\ell$ (the alternative case starting with edge $(0,k)$ is completely symmetric and the decider will finally take the better of the two options).

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

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





To determine the *spe*-path in $G'$ we move backwards through the cycle starting with the decision in $v_\ell$ and going back up to $v_0$. In each vertex $j$ we consider the possibilities of moving on to $j+1$ or taking a swap and anticipating the further moves of the other player. To do so, the cost of potential paths from $j$ to $v_\ell$ after a swap have to be computed beforehand in auxiliary arrays in the spirit of Section 4.1. In Case (ii) this turns out to be fairly complicated since the costs of the path from the current vertex $j$ back up to $v_0$ and on to $k$, $k-1$, etc., downto $v_\ell$ with a possible backward swap have to be taken into account. Note that the turning point $y$ of a backward swap is limited by $j$. Thus, we have to compute the cost of a backward swap for all bounds $j$.

In the initialization of all paths leading to $v_\ell$ we also have to take the *spe*-path from $v_\ell$ to $t$ into account which was computed before. The correct combination of solutions of two adjacent cycles contains another subtle detail, since the joining vertex $v_\ell$ of the upper cycle can be traversed at most twice and thus only certain combinations are feasible.

While the main procedure can be done in linear time over all vertices of a cycle, the auxiliary array to compute the backward swap would be quadratic in a straightforward implementation. However, we can do better by utilizing an advanced data structure and thus reach the following statement:

**Theorem 4.** *The* spe*-path of* SHORTEST PATH GAME *on undirected cactus graphs can be computed in $O(n \log n)$ time.*

## References

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

Darmann, A., Klamler, C., Pferschy, U.: Sharing the cost of a path. Technical report (2013), `http://ssrn.com/abstract=2287875` (submitted)

Darmann, A., Nicosia, G., Pferschy, U., Schauer, J.: The subset sum game. European Journal on Operational Research 233, 539–549 (2014)

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

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

Fraenkel, A.S., Simonson, S.: Geography. Theoretical Computer Science 110(1), 197–214 (1993)

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

Osborne, M.J.: An Introduction to Game Theory. Oxford Univ. Press (2004)

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

Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Proc. of the 5th Symp. on Theory of Computing, STOC 1973, pp. 1–9 (1973)