Faculty of Informatics
Masaryk University Brno

# Using Strategy Improvement to Stay Alive

by

Luboš Brim and Jakub Chaloupka

FI MU Report Series FIMU-RS-2010-03

Publications in the FI MU Report Series are in general accessible via WWW:

`http://www.fi.muni.cz/reports/`

Further information can be obtained by contacting:

Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic

# Using Strategy Improvement to Stay Alive[*]

Luboš Brim and Jakub Chaloupka
Faculty of Informatics, Masaryk University,
Botanická 68a, 60200 Brno, Czech Republic
{brim, xchalou1}@fi.muni.cz

March 31, 2010

**Abstract**

We design a novel algorithm for solving Mean-Payoff Games (MPGs). Besides solving an MPG in the usual sense, our algorithm computes more information about the game, information that is important with respect to applications. The weights of the edges of an MPG can be thought of as a gained/consumed energy – depending on the sign. For each vertex, our algorithm computes the minimum amount of initial energy that is sufficient for player Max to ensure that in a play starting from the vertex, the energy level never goes below zero. Our algorithm is not the first algorithm that computes the minimum sufficient initial energies, but according to our experimental study it is the fastest algorithm that computes them. The reason is that it utilizes the strategy improvement technique which is very efficient in practice.

## 1 Introduction

A *Mean-Payoff Game (MPG)* [11, 14, 18] is a two-player infinite game played on a finite weighted directed graph, the vertices of which are divided between the two players. A play starts by placing a token on some vertex and the players, named Max and Min, move the token along the edges of the graph ad infinitum. If the token is on Max's vertex, he chooses an outgoing edge and the token goes to the destination vertex of that edge. If the token is on Min's vertex, it is her turn to choose an outgoing edge. Roughly

[*]This work has been partially supported by the Grant Agency of the Czech Republic grants No. 201/09/1389, 102/09/H042.

speaking, Max wants to maximize the average weight of the traversed edges whereas Min wants to minimize it. It was proved in [11] that each vertex $v$ has a *value*, denoted by $\nu(v)$, which each player can secure by a positional strategy, i.e., strategy that always chooses the same outgoing edge in the same vertex. To solve an MPG is to find the values of all vertices, and, optionally, also strategies that secure the values.

In this paper we deal with MPGs with other than the standard average-weight goal. Player Max now wants the sum of the weights of the traversed edges, plus some initial value (initial "energy"), to be non-negative at each moment of the play. He also wants to know the minimal sufficient amount of initial energy that enables him to stay non-negative. For different starting vertices, the minimal sufficient initial energy may be different and for starting vertices with $\nu < 0$, it is impossible to stay non-negative with arbitrarily large amount of initial energy.

The problem of computation of the minimal sufficient initial energies has been studied under different names by Chakrabarti et al. [4], Lifshits and Pavlov [16], and Bouyer et al. [2]. In [4] it was called the problem of *pure energy interfaces*, in [16] it was called the problem of *potential computation*, and in [2] it was called *the lower-bound problem*. The paper [2] also contains the definition of a similar problem – *the lower-weak-upper-bound problem*. An instance of this problem contains, besides an MPG, also a bound $b$. The goal is the same, Max wants to know how much initial energy he needs to stay non-negative forever, but now the energy level is bounded from above by $b$ and during the play, all increases above this bound are immediately truncated.

Various resource scheduling problems for which the standard solution of an MPG is not useful can be formulated as the lower-bound or the lower-weak-upper-bound problems, which extends the applicability of MPGs. For example, an MPG can be used to model a robot in a hostile environment. The weights of edges represent changes in the remaining battery capacity of the robot – positive edges represent recharging, negative edges represent energy consuming actions. The bound $b$ is the maximum capacity of the battery. Player Max chooses the actions of the robot and player Min chooses the actions of the hostile environment. By solving the lower-weak-upper-bound problem, we find out if there is some strategy of the robot that allows him to survive in the hostile environment, i.e., its remaining battery capacity never goes below zero, and if there is such a strategy, we also get the minimum initial remaining battery capacity that allows him to survive.

The first algorithm solving the lower-bound problem was proposed by Chakrabarti et al. [4] and it is based on value iteration. The algorithm can also be easily modified to solve the lower-weak-upper-bound problem. The value iteration algorithm was later improved by Chaloupka and Brim [6], and independently by Doyen, Gentilini, and Raskin [10], extended version of [6, 10] was recently submitted [3]. Henceforward we will use the term "value iteration" (VI) to denote only the improved version from [6, 10]. The algorithms of Bouyer et al. [2] that solve the two problems are essentially the same as the original algorithm from [4]. However, [2] focuses mainly on other problems than the lower-bound and the lower-weak-upper-bound problems for MPGs. Different approach to solving the lower-bound problem was proposed by Lifshits and Pavlov [16], but their algorithm has exponential space complexity, and so it is not appropriate for practical use. VI seems to be the best known approach to solving the two problems.

In this paper, we design a novel algorithm based on the strategy improvement technique, suitable for practical solving of the lower-bound and the lower-weak-upper-bound problems for large MPGs. The use of the strategy improvement technique for solving MPGs goes back to the algorithm of Hoffman and Karp from 1966 [15]. Their algorithm can be used to solve only a restricted class of MPGs, but strategy improvement algorithms for solving MPGs in general exist as well [1, 17, 9]. However, all of them solve neither the lower-bound nor the lower-weak-upper-bound problem (cf. Section 4, first part, last paragraph), our algorithm is the first. Another contribution of this paper is a further improvement of VI.

The shortcoming of VI is that it takes enormous time on MPGs with at least one vertex with $\nu < 0$. Natural way to alleviate this problem is to find the vertices with $\nu < 0$ by some fast algorithm and run VI on the rest. Based on our previous experience with algorithms for solving MPGs [5], we selected two algorithms for computation of the set of vertices with $\nu < 0$. Namely, the algorithm of Björklund and Vorobyov [1] (BV), and the algorithm of Schewe [17] (SW). This gives us two algorithms: VI + BV and VI + SW. However, the preprocessing is not helpful on MPGs with all vertices with $\nu \geq 0$, and it is also not helpful for solving the lower-weak-upper-bound problem for small bound $b$. Therefore, we also study the algorithm VI without the preprocessing.

Our new algorithm based on the strategy improvement technique that we propose in this paper has the complexity $O(|V| \cdot (|V| \cdot \log |V| + |E|) \cdot W)$, where $W$ is the maximal absolute edge-weight. It is slightly worse than the complexity of VI, the same as the complexity of VI + BV, and better than the complexity of VI + SW. We call our algorithm

"Keep Alive Strategy Improvement" (KASI). It solves both the lower-bound and the lower-weak-upper-bound problem. Moreover, as each algorithm that solves the lower-bound problem also divides the vertices of an MPG into those with $\nu \geq 0$ and those with $\nu < 0$, which can be used to compute the exact $\nu$ values of all vertices, KASI can be thought of as an algorithm that also solves MPGs in the usual sense. As a by-product of the design of KASI, we improved the complexity of BV and proved that Min may not have positional strategy that is also optimal with respect to the lower-weak-upper-bound problem. Moreover, we describe a way to construct an optimal strategy for Min with respect to the lower-weak-upper-bound problem.

To evaluate and compare the algorithms VI, VI + BV, VI + SW, and KASI, we implemented them and carried out an experimental study. According to the study, KASI is the best algorithm.

## 2 Preliminaries

A *Mean-Payoff Game* (MPG) [11, 14, 18] is given by a triple $\Gamma = (G, V_{\text{Max}}, V_{\text{Min}})$, where $G = (V, E, w)$ is a finite weighted directed graph such that $V$ is a disjoint union of the sets $V_{\text{Max}}$ and $V_{\text{Min}}$, $w : E \to \mathbb{Z}$ is the weight function, and each $v \in V$ has out-degree at least one. The game is played by two opposing players, named Max and Min. A play starts by placing a token on some given vertex and the players then move the token along the edges of G ad infinitum. If the token is on vertex $v \in V_{\text{Max}}$, Max moves it. If the token is on vertex $v \in V_{\text{Min}}$, Min moves it. This way an infinite path $p = (v_0, v_1, v_2, \ldots)$ is formed. Max's aim is to maximize his gain: $\liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} w(v_i, v_{i+1})$, and Min's aim is to minimize her loss: $\limsup_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} w(v_i, v_{i+1})$. For each vertex $v \in V$, we define its *value*, denoted by $\nu(v)$, as the maximal gain that Max can ensure if the play starts at vertex $v$. It was proved that it is equal to the minimal loss that Min can ensure. Moreover, both players can ensure $\nu(v)$ by using positional strategies defined below [11].

A (general) *strategy* of Max is a function $\sigma : V^* \cdot V_{\text{Max}} \to V$ such that for each finite path $p = (v_0, \ldots, v_k)$ with $v_k \in V_{\text{Max}}$, it holds that $(v_k, \sigma(p)) \in E$. Recall that each vertex has out-degree at least one, and so the definition of a strategy is correct. The set of all strategies of Max in $\Gamma$ is denoted by $\Sigma^\Gamma$. We say that an infinite path $p = (v_0, v_1, v_2, \ldots)$ agrees with the strategy $\sigma \in \Sigma^\Gamma$ if for each $v_i \in V_{\text{Max}}$, $\sigma(v_0, \ldots, v_i) = v_{i+1}$. A strategy $\pi$ of Min is defined analogously. The set of all strategies of Min in $\Gamma$ is denoted by $\Pi^\Gamma$. Given an initial vertex $v \in V$, the *outcome* of two strategies $\sigma \in \Sigma^\Gamma$ and $\pi \in \Pi^\Gamma$ is the (unique) infinite path $\mathsf{outcome}^\Gamma(v, \sigma, \pi) = (v = v_0, v_1, v_2, \ldots)$ that agrees with both $\sigma$ and $\pi$.

The strategy $\sigma \in \Sigma^\Gamma$ is called a *positional strategy* if $\sigma(p) = \sigma(p')$ for all finite paths $p = (v_0, \ldots, v_k)$ and $p' = (v'_0, \ldots, v'_{k'})$ such that $v_k = v'_{k'} \in V_{\mathrm{Max}}$. For the sake of simplicity, we think of a positional strategy of Max as a function $\sigma : V_{\mathrm{Max}} \to V$ such that $(v, \sigma(v)) \in E$, for each $v \in V_{\mathrm{Max}}$. The set of all positional strategies of Max in $\Gamma$ is denoted by $\Sigma^\Gamma_M$. A positional strategy $\pi$ of Min is defined analogously. The set of all positional strategies of Min in $\Gamma$ is denoted by $\Pi^\Gamma_M$. We define $G_\sigma$, the *restriction of* $G$ *to* $\sigma$, as the graph $(V, E_\sigma, w_\sigma)$, where $E_\sigma = \{(u, v) \in E \mid u \in V_{\mathrm{Min}} \vee \sigma(u) = v\}$, and $w_\sigma = w \restriction E_\sigma$. That is, we get $G_\sigma$ from $G$ by deleting all the edges emanating from Max's vertices that do not follow $\sigma$. $G_\pi$ for a strategy $\pi$ of Min is defined analogously. For $\sigma \in \Sigma^\Gamma_M$, we also define $\Gamma_\sigma = (G_\sigma, V_{\mathrm{Max}}, V_{\mathrm{Min}})$, and for $\pi \in \Pi^\Gamma_M$, $\Gamma_\pi = (G_\pi, V_{\mathrm{Max}}, V_{\mathrm{Min}})$.

The *lower-bound problem* for an MPG $\Gamma = (G = (V, E, w), V_{\mathrm{Max}}, V_{\mathrm{Min}})$ is the problem of finding $\mathsf{lb}^\Gamma(v) \in \mathbb{N}_0 \cup \{\infty\}$ for each $v \in V$, such that:

$$\begin{aligned} \mathsf{lb}^\Gamma(v) = \min\{x \in \mathbb{N}_0 \mid \quad & (\exists \sigma \in \Sigma^\Gamma)(\forall \pi \in \Pi^\Gamma) \\ & (\quad \mathsf{outcome}^\Gamma(v, \sigma, \pi) = (v = v_0, v_1, v_2, \ldots) \wedge \\ & \quad (\forall n \in \mathbb{N})(x + \textstyle\sum_{i=0}^{n-1} w(v_i, v_{i+1}) \geq 0) \ ) \} \end{aligned}$$

where minimum of an empty set is $\infty$. That is, $\mathsf{lb}^\Gamma(v)$ is the minimal sufficient amount of initial energy that enables Max to keep the energy level non-negative forever, if the play starts from $v$. If $\mathsf{lb}^\Gamma(v) = \infty$, which means that $\nu(v) < 0$, then we say that Max loses from $v$, because arbitrarily large amount of initial energy is not sufficient. If $\mathsf{lb}^\Gamma(v) \in \mathbb{N}_0$, then Max wins from $v$.

The strategy $\sigma \in \Sigma^\Gamma$ is an *optimal strategy of Max with respect to the lower-bound problem*, if it ensures that for each $v \in V$ such that $\mathsf{lb}^\Gamma(v) \neq \infty$, $\mathsf{lb}^\Gamma(v)$ is a sufficient amount of initial energy. Formally:

$$\begin{aligned} \mathsf{lb}^\Gamma(v) \geq \min\{x \in \mathbb{N}_0 \mid \quad & (\forall \pi \in \Pi^\Gamma) \\ & (\quad \mathsf{outcome}^\Gamma(v, \sigma, \pi) = (v = v_0, v_1, v_2, \ldots) \wedge \\ & \quad (\forall n \in \mathbb{N})(x + \textstyle\sum_{i=0}^{n-1} w(v_i, v_{i+1}) \geq 0) \ ) \} \end{aligned}$$

The strategy $\pi \in \Pi^\Gamma$ is an *optimal strategy of Min with respect to the lower-bound problem*, if it ensures that for each $v \in V$ such that $\mathsf{lb}^\Gamma(v) \neq \infty$, Max needs at least $\mathsf{lb}^\Gamma(v)$ units of initial energy, and for each $v \in V$ such that $\mathsf{lb}^\Gamma(v) = \infty$, Max loses. Formally:

$$\mathsf{lb}^{\Gamma}(v) \leq \min\{x \in \mathbb{N}_0 \mid \quad (\exists \sigma \in \Sigma^{\Gamma}) \\ (\quad \mathsf{outcome}^{\Gamma}(v, \sigma, \pi) = (v = v_0, v_1, v_2, \ldots) \wedge \\ (\forall n \in \mathbb{N})(x + \textstyle\sum_{i=0}^{n-1} w(v_i, v_{i+1}) \geq 0)\ )\ \}$$

The *lower-weak-upper-bound problem* for an MPG $\Gamma = (G = (V, E, w), V_{\text{Max}}, V_{\text{Min}})$ and a bound $b \in \mathbb{N}_0$ is the problem of finding $\mathsf{lwub}_b^{\Gamma}(v) \in \mathbb{N}_0 \cup \{\infty\}$ for each $v \in V$, such that:

$$\mathsf{lwub}_b^{\Gamma}(v) = \min\{x \in \mathbb{N}_0 \mid \quad (\exists \sigma \in \Sigma^{\Gamma})(\forall \pi \in \Pi^{\Gamma}) \\ (\quad \mathsf{outcome}^{\Gamma}(v, \sigma, \pi) = (v = v_0, v_1, v_2, \ldots) \wedge \\ (\forall n \in \mathbb{N})(x + \textstyle\sum_{i=0}^{n-1} w(v_i, v_{i+1}) \geq 0) \wedge \\ (\forall n_1, n_2 \in \mathbb{N}_0)(n_1 < n_2 \Rightarrow \textstyle\sum_{i=n_1}^{n_2-1} w(v_i, v_{i+1}) \geq -b)\ )\ \}$$

where minimum of an empty set is $\infty$. That is, $\mathsf{lwub}_b^{\Gamma}(v)$ is the minimal sufficient amount of initial energy that enables Max to keep the energy level non-negative forever, if the play starts from $v$, under the additional condition that the energy level is truncated to $b$ whenever it exceeds the bound. The additional condition is equivalent to the condition that the play does not contain a segment of weight less than $-b$. If $\mathsf{lwub}_b^{\Gamma}(v) = \infty$, then we say that Max loses from $v$, because arbitrarily large amount of initial energy is not sufficient. If $\mathsf{lwub}_b^{\Gamma}(v) \in \mathbb{N}_0$, then Max wins from $v$. Optimal strategies for Max and Min with respect to the lower-weak-upper-bound problem are defined in the same way as for the lower-bound problem.

It was proved in [2] that both for the lower-bound problem and the lower-weak-upper-bound problem Max can restrict himself only to positional strategies, i.e., he always has a positional strategy that is also optimal. Therefore, we could use the set $\Sigma_M^{\Gamma}$ instead of the set $\Sigma^{\Gamma}$ in the definitions of both the lower-bound problem and the lower-weak-upper-bound problem.

In the rest of the paper, we will focus only on the lower-weak-upper-bound problem, because it includes the lower-bound problem as a special case. The reason is that for each $v \in V$ such that $\mathsf{lb}^{\Gamma}(v) < \infty$, it holds that $\mathsf{lb}^{\Gamma}(v) \leq (|V| - 1) \cdot W$, where $W$ is the maximal absolute edge-weight in G. It was proved in [2]. Therefore, if we choose $b = (|V| - 1) \cdot W$, then for each $v \in V$, $\mathsf{lb}^{\Gamma}(v) = \mathsf{lwub}_b^{\Gamma}(v)$.

Let $G = (V, E, w)$ be a weighted directed graph, let $p = (v_0, \ldots, v_k)$ be a path in G, and let $c = (u_0, \ldots, u_{r-1}, u_r = u_0)$ be a cycle in G. Then $w(p)$, the weight of $p$, $l(p)$, the number of edges in $p$, $w(c)$, the weight of $c$, and $l(c)$, the number of edges in c, are defined in the following way: $w(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$, $l(p) = k$, $w(c) = \sum_{i=0}^{r-1} w(u_i, u_{i+1})$, $l(c) = r$.

The set of all (finite) paths in G is denoted by $\mathsf{path}^G$, the set of all cycles in G is denoted by $\mathsf{cycle}^G$, and finally the set of all infinite paths in G is denoted by $\mathsf{path}^G_\infty$.

A suffix of a path $p = (v_0, \ldots, v_k) \in \mathsf{path}^G$ is a path $(v_i, \ldots, v_k)$, where $i \in \{0, \ldots, k\}$. The set of all suffixes of p is denoted by $\mathsf{suffix}(p)$. A prefix of p is a path $(v_0, \ldots, v_i)$, where $i \in \{0, \ldots, k\}$. The set of all prefixes of p is denoted by $\mathsf{prefix}(p)$. Finally, a segment of p is a path $(v_i, \ldots, v_j)$, where $i, j \in \{0, \ldots, k\}$ and $i \le j$. The set of all segments of p is denoted by $\mathsf{segment}(p)$. The definitions of segments and prefixes naturally extend to infinite paths. A suffix of a path $p = (v_0, v_1, v_2, \ldots) \in \mathsf{path}^G_\infty$ is a path $(v_i, v_{i+1}, v_{i+2}, \ldots)$, where $i \in \mathbb{N}_0$.

Let $c = (u_0, \ldots, u_{r-1}, u_0) \in \mathsf{cycle}^G$. If $u_j \in c$, then $u_{j+1}$ is the vertex following $u_j$ in c. In particular, $j + 1$ is taken modulo r. A segment of c is a path $(u_i, u_{i+1}, \ldots, u_j)$, where $i, j \in \{0, \ldots, r-1\}$. Please note that we do not require $i \le j$, because there is a path in c between any two vertices in c. The set of all segments of c is denoted by $\mathsf{segment}(c)$.

Let $\Gamma = (G = (V, E, w), V_{\mathrm{Min}}, V_{\mathrm{Max}})$ be an MPG and let $D \subseteq V$. Then $G(D)$ is the subgraph of G induced by the set D. Formally, $G(D) = (D, E \cap D \times D, w \restriction D \times D)$. We also define the restriction of $\Gamma$ induced by D. Since some vertices might have zero out-degree in $G(D)$, we define $\Gamma(D) = (G'(D), V_{\mathrm{Min}} \cap D, V_{\mathrm{Max}} \cap D)$, where $G'(D) = (D, (E \cap D \times D) \cup L, (w \restriction D \times D) \cup w_L)$, where $L = \{(v, v) \mid v \in D \wedge (\nexists u \in D)((v, u) \in E)\}$, and $w_L : L \to \mathbb{Z}$ such that for each $e \in L$, $w(e) = -1$. That is, we make the vertices with zero out-degree in $G(D)$ losing for Max in $\Gamma(D)$ with respect to the the lower-weak-upper-bound problem.

Let $G = (V, E, w)$ be a graph and let $B, A \subseteq V$. If we say that "p is a path from v to B" we mean a path with the last vertex and only the last vertex in B, formally: $p = (v = v_0, \ldots, v_k) \in \mathsf{path}^G$, where $v_0, \ldots, v_{k-1} \in V \setminus B \wedge v_k \in B$. We denote the set of all paths from v to B by $\mathsf{path}^G(v, B)$. Furthermore, a path from A to B is a path from v to B such that $v \in A$. We denote the set of all paths from A to B by $\mathsf{path}^G(A, B)$. The term "longest" in connection with paths always refers to the weights of the paths, not the numbers of edges.

Operations on vectors of the same dimension are element-wise. For example, if $d_0$ and $d_1$ are two vectors of dimension $|V|$, then $d_0 < d_1$ means that for each $v \in V$, $d_0(v) \le d_1(v)$, and for some $v \in V$, $d_0(v) < d_1(v)$.

For the whole paper let $\Gamma = (G = (V, E, w), V_{\mathrm{Max}}, V_{\mathrm{Min}})$ be an MPG and let $W$ be the maximal absolute edge-weight in $G$, i.e., $W = \max_{e \in E} |w(e)|$.

## 3 The Algorithm

High-level description of our Keep Alive Strategy Improvement algorithm (KASI) for the lower-weak-upper-bound problem is as follows. Let $(\Gamma, b)$ be an instance of the lower-weak-upper-bound problem. KASI maintains a vector $d \in (\mathbb{Z} \cup \{-\infty\})^V$ such that $-d \geq \mathbf{0}$ is always a lower estimate of $\mathsf{lwub}^{\Gamma}_{b}$, i.e., $-d \leq \mathsf{lwub}^{\Gamma}_{b}$. The vector $d$ is gradually decreased, and so $-d$ is increased, until $-d = \mathsf{lwub}^{\Gamma}_{b}$. The algorithm also maintains a set $D$ of vertices such that about the vertices in $V \setminus D$ it already knows that they have infinite $\mathsf{lwub}^{\Gamma}_{b}$ value. Initially, $d = \mathbf{0}$ and $D = V$. KASI starts with an arbitrary strategy $\pi \in \Pi^{\Gamma}_{M}$ and then iteratively improves it until no improvement is possible. In each iteration, the current strategy is first evaluated and then improved. The strategy evaluation examines the graph $G_{\pi}(D)$ and updates the vector $d$ so that for each $v \in D$, it holds

$$-d(v) = \mathsf{lwub}^{\Gamma_{\pi}(D)}_{b}(v)$$

That is, it solves the lower-weak-upper-bound problem in the restricted game $\Gamma_{\pi}(D)$, where Min has no choices. This explains why the restricted game was defined the way it was, because if a vertex from $D$ has outgoing edges only to $V \setminus D$, then it is losing for Max in $\Gamma$. The vertices with the $d$ value equal to $-\infty$ are removed from the set $D$. Since the strategy $\pi$ is either the first strategy or an improvement of the previous strategy, the vector $d$ is always decreased by the strategy evaluation and we get a better estimate of $\mathsf{lwub}^{\Gamma}_{b}$. To improve the current strategy the algorithm checks whether for some $(v, u) \in E$ such that $v \in V_{\mathrm{Min}}$ and $d(v) > -\infty$ it holds that $d(v) > d(u) + w(v, u)$. This is called a *strategy improvement condition*. Such an edge indicates that $-d(v)$ is not a sufficient initial energy at $v$, because traversing the edge $w(v, u)$ and continuing from $u$ costs at least $-w(v, u) - d(u)$ units of energy, which is greater than $-d(v)$ (Recall that $-d$ is a lower estimate of $\mathsf{lwub}^{\Gamma}_{b}$). If there are edges satisfying the condition, the strategy $\pi$ is improved in the following way. For each vertex $v \in V_{\mathrm{Min}}$ such that there is an edge $(v, u) \in E$ such that $d(v) > d(u) + w(v, u)$, $\pi(v)$ is switched to $u$. If $v$ has more than one such edge emanating from it, any of them is acceptable. Then, another iteration of KASI is started. If no such edge exists, the algorithm terminates, because it holds that each vertex $v \in V$ has $-d(v) = \mathsf{lwub}^{\Gamma}_{b}(v)$. Detailed description of the algorithm follows.

In Figure 1 is a pseudo-code of the strategy evaluation part of our algorithm. The input to the procedure consists of four parts. The first and the second part form the lower-weak-upper-bound problem instance that the main algorithm KASI is solving, the MPG $\Gamma$ and the bound $b \in \mathbb{N}_0$. The third part is the strategy $\pi \in \Pi_M^\Gamma$ that we want to evaluate and the fourth part of the input is a vector $d_{-1} \in (\mathbb{Z} \cup \{-\infty\})^V$. The vector $d_{-1}$ is such that $-d_{-1}$ is a lower estimate of $\mathsf{lwub}_b^\Gamma$, computed for the previous strategy, or, in case of the first iteration of KASI, set by initialization to a vector of zeros. Let $A = \{v \in V \mid d_{-1}(v) = 0\}$ and $D = \{v \in V \mid d_{-1}(v) > -\infty\}$, then the following conditions hold.

i. $(\forall c \in \mathsf{cycle}^{G_\pi(D \setminus A)})(w(c) < 0)$,

ii. $(\forall v \in D \setminus A)(d_{-1}(v) < 0 \wedge (\forall (v,u) \in E_\pi)(d_{-1}(v) \geq d_{-1}(u) + w(v,u)))$.

From these technical conditions it follows that $-d_{-1}$ is also a lower estimate of $\mathsf{lwub}_b^{\Gamma_\pi(D)}$ and the purpose of the strategy evaluation procedure is to decrease the vector $d_{-1}$ so that the resulting vector $d$ satisfies $-d = \mathsf{lwub}_b^{\Gamma_\pi(D)}$. To see why from (i.) and (ii.) it follows that $-d_{-1} \leq \mathsf{lwub}_b^{\Gamma_\pi(D)}$, consider a path $p = (v_0, \ldots, v_k)$ from $D \setminus A$ to $A$ in $G_\pi(D)$. From (ii.) it follows that for each $j \in \{0, \ldots, k-1\}$, it holds that $d_{-1}(v_j) \geq d_{-1}(v_{j+1}) + w(v_j, v_{j+1})$. If we sum the inequalities, we get $d_{-1}(v_0) \geq d_{-1}(v_k) + w(p)$. Since $v_k \in A$, $d_{-1}(v_k) = 0$ and the inequality becomes $d_{-1}(v_0) \geq w(p)$. Therefore, each infinite path in $G_\pi(D)$ starting from $v \in D$ and containing a vertex from $A$ has a prefix of weight less or equal to $d_{-1}(v_0)$. Furthermore, if the infinite path does not contain a vertex from $A$, weights of its prefixes cannot even be bounded from below, because by (i.), all cycles in $G_\pi(D \setminus A)$ are negative. All in all, $-d_{-1}$ is a lower estimate of $\mathsf{lwub}_b^{\Gamma_\pi(D)}$.

The conditions (i.) and (ii.) trivially hold in the first iteration of the main algorithm, for $d_{-1} = \mathbf{0}$. In each subsequent iteration, $d_{-1}$ is taken from the output of the previous iteration and an intuition why the conditions hold will be given below.

The output of the strategy evaluation procedure is a vector $d \in (\mathbb{Z} \cup \{-\infty\})^V$ such that for each $v \in D$, it holds that $-d(v) = \mathsf{lwub}_b^{\Gamma_\pi(D)}(v)$. Recall that $D = \{v \in V \mid d_{-1}(v) > -\infty\}$.

The strategy evaluation works only with the restricted graph $G_\pi(D)$ and it is based on the fact that if we have the set $B_z = \{v \in D \mid \mathsf{lwub}_b^{\Gamma_\pi(D)}(v) = 0\}$, i.e., the set of vertices where Max does not need any initial energy to win, then we can compute $\mathsf{lwub}_b^{\Gamma_\pi(D)}$ of the remaining vertices by computing longest paths to the set $B_z$. More precisely:

$$(\forall v \in D \setminus B)(\mathsf{lwub}_b^{\Gamma_\pi(D)}(v) = -\max\{w(p) \mid \; p \in \mathsf{path}^{G_\pi(D)}(v, B_z) \wedge \\ (\forall p_f \in \mathsf{suffix}(p))(w(p_f) \geq -b)\,\}$$

To get some idea about why this holds consider a play winning for Max. The energy level never drops below zero in the play, and so there must be a moment from which onwards the energy level never drops at all. Therefore, Max does not need any initial energy to win a play starting from the appropriate vertex (Please note that Min has no choices in $\Gamma_\pi(D)$), and so $B_z$ is not empty. For the vertices in $D \setminus B_z$, in order to win, Max has to get to some vertex in $B_z$ without exhausting all of his energy. So the minimal sufficient energy to win is the minimal energy that Max needs to get to some vertex in $B_z$. All paths from $D \setminus B_z$ to $B_z$ must be negative (otherwise $B_z$ would be larger), and so the minimal energy to get to $B_z$ is the absolute value of the weight of a longest path to $B_z$ such that the weight of each suffix of the path is greater or equal to $-b$. If no path to $B_z$ exists or all such paths have suffixes of weight less than $-b$, Max cannot win.

Initially, the procedure over-approximates the set $B_z$ by the set $B_0$ of vertices $v$ with $d_{-1}(v) = 0$ that have an edge $(v, u)$ such that $w(v, u) - d_{-1}(v) + d_{-1}(u) \geq 0$ emanating from them (line 2), and then iteratively removes vertices from the set until it arrives at the correct set $B_z$. The vector $-d_i$ is always a lower estimate of $\mathsf{lwub}_b^{\Gamma_\pi(D)}$, i.e., it always holds that $-d_i \leq \mathsf{lwub}_b^{\Gamma_\pi(D)}$. Therefore, only vertices $v$ with $d_i(v) = 0$ are candidates for the final set $B_z$. However, the vertices $v$ with $d_i(v) = 0$ such that for each edge $(v, u)$, it holds that $w(v, u) - d_i(v) + d_i(u) < 0$ are removed from the set of candidates. The reason is that since $d_i(v) = 0$, the inequality can be developed to $-w(v, u) - d_i(u) > 0$, and so if the edge $(v, u)$ is chosen in the first step, then more than zero units of initial energy are needed at $v$. During the execution of the procedure, $d_i$ decreases, and so $-d_i$ increases, until $-d_i = \mathsf{lwub}_b^{\Gamma_\pi(D)}$.

In each iteration, the procedure uses a variant of the Dijkstra's algorithm to compute longest paths from all vertices to $B_i$ on line 4. Since $B_i$ is an over-approximation of $B_z$, the absolute values of the weights of the longest paths are a lower estimate of $\mathsf{lwub}_b^{\Gamma_\pi(D)}$. The weights of the longest paths are assigned to $d_i$. In particular, for each $v \in B_i$, $d_i(v) = 0$. Dijkstra's algorithm requires all edge-weights be non-positive (Please note that we are computing longest paths). Since edge-weights are arbitrary integers, we apply potential transformation on them to make them non-positive. As vertex potentials we use $d_{i-1}$, which contains the longest path weights computed in the previous iteration, or, in case $i = 0$, is given as input. Transformed weight of an edge $(x, y)$ is $w(x, y) - d_{i-1}(x) +$

```
proc EvaluateStrategy($\Gamma, b, \pi, d_{-1}$)
  $i := 0$; $B_0 := \{v \in V \mid d_{-1}(v) = 0 \wedge \max_{(v,u)\in E_\pi}(w(v,u) - d_{-1}(v) + d_{-1}(u)) \geq 0\}$
  while $i = 0 \vee B_{i-1} \neq B_i$ do
    $d_i :=$ Dijkstra($G_\pi, b, B_i, d_{i-1}$)
    $i := i + 1$
    $B_i := B_{i-1} \setminus \{v \mid \max_{(v,u)\in E_\pi}(w(v,u) - d_{i-1}(v) + d_{i-1}(u)) < 0\}$
  od
  return $d_{i-1}$
end
```

Figure 1: Evaluation of strategy

$d_{i-1}(y)$, which is always non-positive for the relevant edges. In the first iteration of the main algorithm it follows from the condition (ii.), and in the subsequent iterations it follows from properties of longest path weights and the fact that only vertices with all outgoing edges negative with the potential transformation are removed from the candidate set.

The Dijkstra's algorithm is also modified so that it assigns $-\infty$ to each $v \in D$ such that each path from $v$ to $B_i$ has a suffix of weight less than $-b$. Therefore, the vertices from which $B_i$ is not reachable or is reachable only via paths with suffixes of weight less than $-b$ have $d_i$ equal to $-\infty$. Also, vertices from $V \setminus D$ have $d_i$ equal to $-\infty$. A detailed description of Dijkstra() is in Appendix Section 6.1.

On line 5, the variable $i$ is increased (thus the current longest path weights are now in $d_{i-1}$), and on line 6, we remove from $B_{i-1}$ each vertex $v$ that does not have an outgoing edge $(v, u)$ such that $w(v,u) - d_{i-1}(v) + d_{i-1}(u) \geq 0$. Another iteration is started only if $B_i \neq B_{i-1}$. If no vertex is removed on line 6, then $B_i = B_{i-1}$ and the algorithm finishes and returns $d_{i-1}$ as output. The following theorem establishes the correctness of the algorithm. An intuition why the theorem holds was given above. Its formal proof is in Appendix Section 6.2.

**Theorem 3.1** *Let $(\Gamma, b)$ be an instance of the lower-weak-upper-bound problem. Let further $\pi \in \Pi_M^\Gamma$ be a positional strategy of Min, and finally let $d_{-1} \in (\mathbb{Z} \cup \{-\infty\})^V$ be such that for $A = \{v \in V \mid d_{-1}(v) = 0\}$ and $D = \{v \in V \mid d_{-1}(v) > -\infty\}$, the conditions (i.) and (ii.) hold. Then for $d :=$ EvaluateStrategy$(\Gamma, b, \pi, d_{-1})$ it holds that for each $v \in D$, $d(v) = -lwub_b^{\Gamma_\pi(D)}(v)$.*

```
1  proc LowerWeakUpperBound$(\Gamma, b)$
2    $i := 0$; $\pi_0 :=$ Arbitrary strategy from $\Pi^{\Gamma}_{M}$
3    $d_{-1} := \mathbf{0}$
4    $improvement := true$
5    while $improvement$ do
6      $d_i :=$ EvaluateStrategy$(\Gamma, b, \pi_i, d_{i-1})$
7      $improvement := false$
8      $i := i + 1$
9      $\pi_i := \pi_{i-1}$
10     foreach $v \in V_{Min}$ do
11       if $d_{i-1}(v) > -\infty$ then
12         foreach $(v, u) \in E$ do
13           if $d_{i-1}(v) > d_{i-1}(u) + w(v, u)$ then
14             $\pi_i(v) := u$; $improvement := true$
15           fi
16         od
17       fi
18     od
19   od
20   return $-d_{i-1}$
21 end
```

Figure 2: Solving the lower-weak-upper-bound problem

The complexity of EvaluateStrategy() is $O(|V| \cdot (|V| \cdot \log |V| + |E|))$. The term $(|V| \cdot \log |V| + |E|)$ is for Dijkstra() and the number of iterations of the while loop on lines 3–7 is at most $|V|$, because $B_i \subseteq V$ loses at least one element in each iteration.

In Figure 2 is a pseudo-code of our strategy improvement algorithm for solving the lower-weak-upper-bound problem using EvaluateStrategy(). The input to the algorithm is a lower-weak-upper-bound problem instance $(\Gamma, b)$. The output of the algorithm is the vector $\mathsf{lwub}^{\Gamma}_{b}$. The pseudo-code corresponds to the high-level description of the algorithm given at the beginning of this section.

The algorithm proceeds in iterations. It starts by taking an arbitrary strategy from $\Pi^{\Gamma}_{M}$ on line 2, and initializing the vector $d_{-1}$ to vector of zeros on line 3. The vector $-d_i$

is always a lower estimate of $\mathsf{lwub}_b^\Gamma$, i.e., it always holds that $-d_i \leq \mathsf{lwub}_b^\Gamma$. During the execution of the algorithm, $d_i$ decreases, and so $-d_i$ increases, until $-d_i = \mathsf{lwub}_b^\Gamma$.

In each iteration of the main while-loop on lines 5–19, the current strategy $\pi_i \in \Pi_M^\Gamma$ is first evaluated and then improved, if possible. The strategy $\pi_i$ is evaluated by the procedure EVALUATESTRATEGY() on line 6. By Theorem 3.1, for the strategy evaluation, the vector $d_i$, it holds that for each $v \in D_{i-1}$, $-d_i(v) = \mathsf{lwub}_b^{\Gamma_{\pi_i}(D_{i-1})}(v)$, where $D_{i-1} = \{v \in V \mid d_{i-1}(v) > -\infty\}$. In other words, for each $v \in D_{i-1}$, $-d_i(v)$ is the minimal amount of initial energy that Max needs to keep the energy level non-negative in a play starting from $v$ in the MPG $\Gamma(D_{i-1})$, if Min uses the strategy $\pi_i$. On line 8, the variable $i$ is increased, so the current strategy evaluation in now the vector $d_{i-1}$.

The vector $d_{i-1}$ is used to improve Min's strategy on lines 10–18. All Min's vertices are examined to see whether for some $v \in V_{\mathrm{Min}}$, she can choose an outgoing edge that makes $-d_{i-1}(v)$ insufficient amount of initial energy for Max at the vertex $v$. That is why $-d_i$, computed in the subsequent iteration, is greater. Min's strategy is updated for all vertices for which the strategy improvement condition is satisfied. If some update takes place, another iteration is started. Otherwise, we have the input lower-weak-upper-bound problem instance solved and so the algorithm finishes. It holds that for each $v \in V$, $-d_{i-1}(v) = \mathsf{lwub}_b^\Gamma(v)$, hence the command on line 20. The whole algorithm KASI is illustrated on Example 3.2. The following lemmas and theorem establish the correctness of the algorithm.

**Example 3.2** *In Figure 3 is an example of a run of our algorithm KASI on a simple MPG. The MPG is in Figure 3 (a). Circles are Max's vertices and the square is a Min's vertex. Let's denote the MPG by* $\Gamma$, *let* $b = 15$ *and consider a lower-weak-upper-bound problem given by* $(\Gamma, b)$. *Min has only two positional strategies, namely,* $\pi^1$ *and* $\pi^2$, *where* $\pi^1(v_3) = v_1$ *and* $\pi^2(v3) = v_4$. *Let* $\pi = \pi^2$ *be the first selected strategy. For simplicity, we will use the symbols* $\pi$, $d$, $B$, *and* $D$ *without indices, although in pseudo-codes these symbols have indices, and the set* $D$ *of vertices with finite* $d$ *value is not even explicitly used. Also, if we speak about a weight of an edge, we mean the weight with the potential transformation by* $d$. *Initially,* $d = \mathbf{0}$ *and* $D = \{v_1, v_2, v_3, v_4\}$.

*There are three vertices in* $G_\pi(D)$ *with non-negative edges emanating from them, namely,* $v_1, v_2, v_3$, *and so* EVALUATESTRATEGY() *takes* $\{v_1, v_2, v_3\}$ *as the first set* $B$. *After the vector* $d$ *is updated so that it contains longest path weights to* $B$ *(Figure 3 (b)), all vertices in* $B$ *still have non-negative edges, and so the strategy evaluation finishes and the strategy improvement phase is started. The strategy improvement condition is satisfied for the edge* $(v_3, v_1)$ *and so* $\pi$ *is*

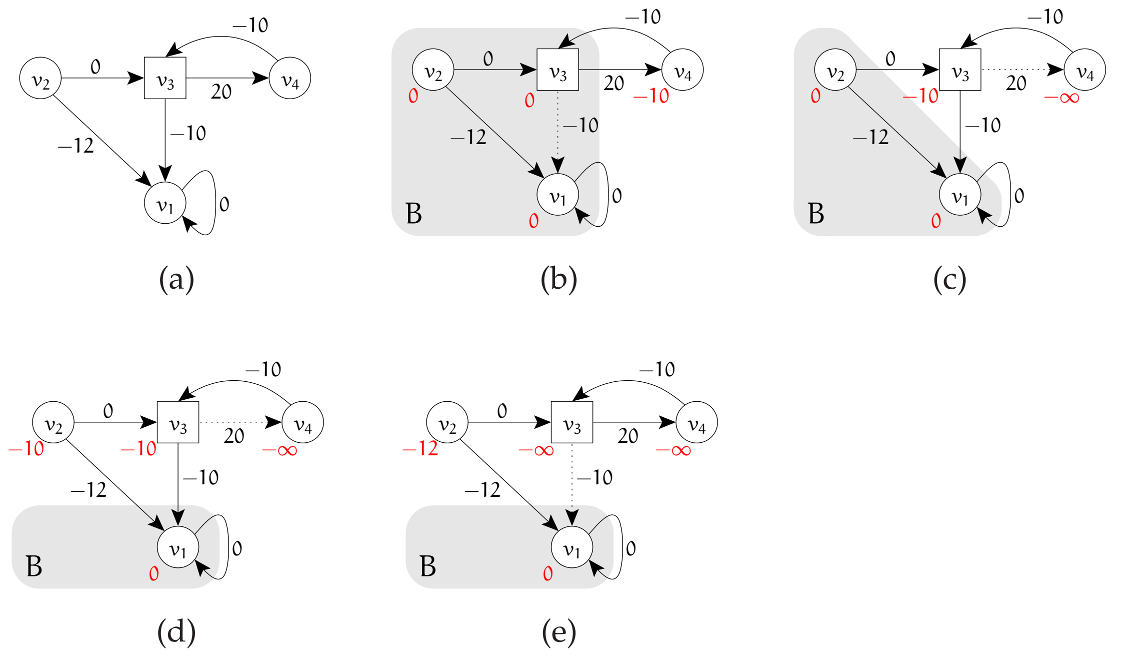

Figure 3: Example of a Run of KASI

*improved so that $\pi = \pi^1$. This completes the first iteration of KASI and another one is started to evaluate and possibly improve the new strategy $\pi$.*

*Now the vertex $v_3$ does not have a non-negative edge emanating from it, so it is removed from the set $B$ and the longest path weights are recomputed (Figure 3 (c)). Please note that the only path from $v_4$ to $B$ has a suffix of weight less than $-b$, and so $d(v_4) = -\infty$ and $v_4$ is removed from the set $D$. The update to $d$ causes that $v_2$ does not have a non-negative edge, thus it is also removed from the set $B$ and the vector $d$ is recomputed again (Figure 3 (d)). This finishes the strategy evaluation and strategy improvement follows. The strategy improvement condition is satisfied for the edge $(v_3, v_4)$, and so the strategy $\pi^2$ is selected as the current strategy $\pi$ again. However, this is not the same situation as at the beginning, because the set $D$ is now smaller. Evaluation of the strategy $\pi$ results in the $d$ vector as depicted in Figure 3 (e). The vertex $v_3$ has $d(v_3) = -\infty$, because $v_3$ cannot reach the set $B$, which also results in removal of $v_3$ from $D$. No further improvement of $\pi$ is possible, and so $\mathsf{lwub}_b^\Gamma = -d = (0, 12, \infty, \infty)$.*

**Lemma 3.3** *Every time line 6 of* LowerWeakUpperBound() *is reached, $\Gamma$, $b$, $\pi_i$, and $d_{i-1}$ satisfy the assumptions of Theorem 3.1. Every time line 7 of* LowerWeakUpperBound() *is reached and $i > 0$, it holds that $d_i < d_{i-1}$.*

A formal proof of Lemma 3.3 is in Appendix Section 6.2. The proof uses the following facts. The first one we have already used: If $p$ is a path from $v$ to $u$ such that for each edge $(x, y)$ in the path it holds that $d(x) \geq d(y) + w(x, y)$, then $d(v) \geq d(u) + w(p)$, and if for some edge the inequality is strict, then $d(v) > d(u) + w(p)$. The second fact is similar: If $c$ is a cycle such that for each edge $(x, y)$ in the cycle it holds that $d(x) \geq d(y) + w(x, y)$, then $0 \geq w(c)$, and if for some edge the inequality is strict, then the cycle is strictly negative. Using these facts we can now give an intuition why the lemma holds.

The assumptions of Theorem 3.1, conditions (i.) and (ii.), are trivially satisfied in the first iteration of LOWERWEAKUPPERBOUND(), as was already mentioned. During the execution of EVALUATESTRATEGY(), conditions (i.) and (ii.) remain satisfied, for the following reasons. The $d$ values of vertices from $D$ are weights of longest paths to $B$, and so each edge $(x, y)$ emanating from a vertex from $D \setminus B$ satisfies $d(x) \geq d(y) + w(x, y)$. Only vertices with all outgoing edges negative with the potential transformation are removed from the set $B$, i.e., only the vertices with each outgoing edge $(x, y)$ satisfying $d(x) > d(y) + w(x, y)$. Using the facts from the previous paragraph, we can conclude that all newly formed cycles in $G_\pi(D \setminus B)$ are negative and the weights of longest paths to $B$ cannot increase. So to complete the intuition, it remains to show why the conditions still hold after the strategy improvement and why the strategy improvement results in decrease of the $d$ vector. This follows from the fact that the new edges introduced by the strategy improvement are negative with the potential transformation.

**Lemma 3.4** *The procedure* LOWERWEAKUPPERBOUND() *always terminates.*

**Proof:** By Lemma 3.3, $d_i$ decreases in each iteration. For each $v \in V$, $d_i(v)$ is bounded from below by the term $-(|V| - 1) \cdot W$, because it is the weight of some path in $G$ with no repeated vertices (Except for the case when $d_i(v) = -\infty$, but this is obviously not a problem). Since $d_i$ is a vector of integers, infinite chain of improvements is not possible, and so termination is guaranteed. ■

The next theorem is the main theorem of this paper which establishes the correctness of LOWERWEAKUPPERBOUND(). Its proof is in Appendix Section 6.2. The key idea of the proof is to define strategies for both players with the following properties. Let $ds :=$ LOWERWEAKUPPERBOUND$(\Gamma, b)$. Max's strategy that we will define ensures that for each vertex $v \in V$, $ds(v)$ is a sufficient amount of initial energy no matter what his opponent does, and Min's strategy that we will define ensures that Max cannot do with smaller amount of initial energy. In particular, for vertices with $ds(v) = \infty$, the strategy ensures that Max will eventually go negative or traverse a path segment of weight less than $-b$ with arbitrarily large amount of initial energy. From the existence of such strategies it follows that for each $v \in V$, $ds(v) = \mathsf{lwub}_b^\Gamma(v)$, and both strategies are optimal with respect to the lower-weak-upper-bound problem.

The optimal strategy of Max is constructed from the final longest path forest computed by EVALUATESTRATEGY() and the non-negative (with potential transformation) edges emanating from the final set B. The optimal strategy of Min is more complicated.

There is a theorem in [2] which claims that Min can restrict herself to positional strategies. Unfortunately, this is not true. Unlike Max, Min sometimes needs memory. Example 3.2 is a proof of this fact, because none of the two positional strategies of Min guarantees that Max loses from the vertex $v_3$. However, Min can play optimally using the sequence of positional strategies computed by our algorithm. In Example 3.2, to guarantee that Max loses from $v_3$, Min first sends the play from $v_3$ to $v_4$ and when it returns back to $v_3$, she sends the play to $v_1$. As a result, a path of weight $-20$ is traversed and since $b = 15$, Max loses.

In general, let $\pi_0, \pi_1, \ldots$ be the sequence of positional strategies computed by the algorithm. Min uses the sequence in the following way: if the play starts from a vertex with finite final $d$ value and never leaves the set of vertices with finite final $d$ value, then Min uses the last strategy in the sequence, and it is the best she can do, as stated by Theorem 3.1. If the play starts or gets to a vertex with infinite final $d$ value, she uses the strategy that caused that the $d$ value of that vertex became $-\infty$, but only until the play gets to a vertex that was made infinite by some strategy with lower index. At that moment Min switches to the appropriate strategy. In particular, Min never switches to a strategy with higher index.

**Theorem 3.5** *Let* $ds :=$ LOWERWEAKUPPERBOUND$(\Gamma, b)$*, then for each* $v \in V$*,* $ds(v) = \mathsf{lwub}_b^\Gamma(v)$*.*

The algorithm has a pseudopolynomial time complexity: $O(|V|^2 \cdot (|V| \cdot \log |V| + |E|) \cdot W)$. It takes $O(|V|^2 \cdot W)$ iterations until the while-loop on lines 5–19 terminates. The reason is that for each $v \in V$, if $d(v) > -\infty$, then $d(v) \geq -(|V| - 1) \cdot W$, because $d(v)$ is the weight of some path with no repeated vertices, and so the $d$ vector can be improved at most $O(|V|^2 \cdot W)$ times. Each iteration, if considered separately, takes $O(|V| \cdot (|V| \cdot \log |V| + |E|))$, so one would say that the overall complexity should be $O(|V|^3 \cdot (|V| \cdot \log |V| + |E|) \cdot W)$. However, the number of elements of the set $B_i$ in EVALUATESTRATEGY() never increases, even between two distinct calls of the evaluation procedure, hence the amortized complexity of one iteration is only $O(|V| \cdot \log |V| + |E|)$. Therefore, the overall complexity is better by a factor of $|V|$.

The algorithm can even be improved so that its complexity is $O(|V| \cdot (|V| \cdot \log |V| + |E|) \cdot W)$. This is accomplished by efficient computation of vertices which which will update their $d$ value in the next iteration so that computational time is not wasted on vertices whose $d$ value is not going to change. Interestingly enough, the same technique can be used to improve the complexity of the algorithm of Björklund and Vorobyov so that the complexities of the two algorithms are the same. Detailed description of the technique is in Appendix Section 6.3.

## 4 Experimental Evaluation

Our experimental study compares four algorithms for solving the lower-bound and the lower-weak-upper-bound problems. The first is value iteration [6, 10] (VI). The second and the third are combinations of VI with other algorithm. Finally, the fourth algorithm is our algorithm KASI. We will now briefly describe the algorithms based on VI.

Let $(\Gamma, b)$ be an instance of the lower-weak-upper-bound problem. VI starts with $d_0(v) = 0$, for each $v \in V$, and then computes $d_1, d_2, \ldots$ according to the following rules.

$$d_{i+1}(v) = \begin{cases} x = \min_{(v,u)\in E} \max(0, d_i(u) - w(v,u)) & \text{if } v \in V_{Max} \wedge x \le b \\ x = \max_{(v,u)\in E} \max(0, d_i(u) - w(v,u)) & \text{if } v \in V_{Min} \wedge x \le b \\ \infty & \text{otherwise} \end{cases}$$

It is easy to see that for each $v \in V$ and $k \in \mathbb{N}_0$, $d_k(v)$ is the minimum amount of Max's initial energy that enables him to keep the sum of traversed edges, plus $d_k(v)$, greater or equal to zero in a $k$-step play. The computation continues until two consecutive $d$ vectors are equal. The last $d$ vector is then the desired vector $\mathsf{lwub}_b^\Gamma$. If $b = (|V| - 1) \cdot W$, the algorithm solves the lower-bound problem. The complexity of the straightforward implementation of the algorithm is $O(|V|^2 \cdot |E| \cdot W)$, which was improved in [6, 10] to $O(|V| \cdot |E| \cdot W)$, which is slightly better than the complexity of KASI.

The shortcoming of VI is that it takes enormous time before the vertices with infinite $\mathsf{lb}^\Gamma$ and $\mathsf{lwub}_b^\Gamma$ value are identified. That's why we first compute the vertices with $\nu < 0$ by some fast MPG solving algorithm and then apply VI on the rest of the graph. For the lower-bound problem, the vertices with $\nu < 0$ are exactly the vertices with infinite $\mathsf{lb}^\Gamma$ value. For the lower-weak-upper-bound problem, the vertices with $\nu < 0$ might be a strict subset of the vertices with infinite $\mathsf{lwub}_b^\Gamma$ value, but still the preprocessing sometimes saves a lot of time in practice. It is obvious that on MPGs with all vertices with $\nu \geq 0$ the preprocessing does not help at all. It is also not helpful for the lower-weak-upper-bound problem for small bound $b$.

According to our experiments, which were partly published in [5], the fastest algorithms in practice for dividing the vertices of an MPG into those with $\nu \geq 0$ and $\nu < 0$ are the algorithm of Björklund and Vorobyov [1] (BV) and the algorithm of Schewe [17] (SW). The fact that they are the fastest does not directly follow from [5], because that paper focuses on parallel algorithms and computation of the exact $\nu$ values.

The original algorithm BV is a sub-exponential randomized algorithm. To prove that the algorithm is sub-exponential, some restrictions had to be imposed. If these restrictions are not obeyed, BV runs faster. Therefore, we decided not to obey the restrictions and use only the "deterministic part" of the algorithm. We used only the modified BV algorithm in our experimental study. We even improved the complexity of the deterministic algorithm from $O(|V|^2 \cdot |E| \cdot W)$ to $O(|V| \cdot (|V| \cdot \log |V| + |E|) \cdot W)$ using the same technique as for the improvement of the complexity of KASI which is described in Appendix Section 6.3. Since the results of the improved BV were significantly better on all input instances included in our experimental study, all results of BV in this paper are the results of the improved BV.

The complexity of SW is $O(|V|^2 \cdot (|V| \cdot \log |V| + |E|) \cdot W)$. It might seem that this is in contradiction with the title of Schewe's paper [17], because if some algorithm is optimal, one would expect that there are no algorithms with better complexity. However, the term "optimal" in the title of the paper refers to the strategy improvement technique. SW is also a strategy improvement algorithm, and the strategy improvement steps in SW are optimal in a certain sense.

We note that any algorithm that divides the vertices of an MPG into those with $\nu \geq 0$ and those $\nu < 0$ can be used to solve the lower-bound and the lower-weak-upper-bound problem with the help of binary search, but it requires introduction of auxiliary edges and vertices into the input MPG and repeated application of the algorithm. According to our experiments, BV and SW run no faster than KASI. Therefore, solving the two problems by repeated application of BV and SW would lead to higher runtimes than the runtimes of KASI. If we use the reduction technique from [2], then BV/SW has to be executed $\Theta(|V| \cdot \log(|V| \cdot W))$ times to solve the lower-bound problem, and $\Theta(|V| \cdot \log b)$ times to solve the lower-weak-upper-bound. That's why we compared KASI only with the algorithm VI and the two combined algorithms: VI + BV and VI + SW. Both the complexity of BV and the complexity of SW exceed the complexity of VI, and so the complexities of the combined algorithms are the complexities of BV and SW.

## 4.1 Input MPGs

We experimented with completely random MPGs as well as more structured synthetic MPGs and MPGs modeling simple reactive systems. The synthetic MPGs were generated by two generators, namely SPRAND [8] and TOR [7], downloadable from [13]. The outputs of these generators are only directed weighted graphs, and so we had to divide vertices between Max and Min ourselves. We divided them uniformly at random. The MPGs modeling simple reactive systems we created ourselves.

SPRAND was used to generate the "randx" MPG family. Each of these MPGs contains $|E| = x \cdot |V|$ edges and consist of a random Hamiltonian cycle and $|E| - |V|$ additional random edges, with weights chosen uniformly at random from $[1, 10000]$. To make these inputs harder for the algorithms, in each of them, we subtracted a constant from each edge-weight so that the $\nu$ value of each vertex is close to $0$.

TOR was used for generation of the families "sqnc", "lnc", and "pnc". The sqnc and lnc families are 2-dimensional grids with wrap-around, while the pnc family contains layered networks embedded on a torus. We also created subfamilies of each of the three families by adding cycles to the graphs. For more information on these inputs we refer you to [12] or [5]. Like for the SPRAND generated inputs, we adjusted each TOR generated MPG so that the $\nu$ value of each vertex is close to $0$.

As for the MPGs modeling simple reactive systems, we created three parameterized models. The first is called "collect" and models a robot on a ground with obstacles which has to collect items occurring at different locations according to certain rules. Moving and even idling consumes energy, and so the robot has to return to its docking station from time to time to recharge. By solving the lower-bound, or the lower-weak-upper-bound problem for the corresponding MPG, depending on whether there is some upper bound on the robot's energy, we find out from which initial configurations the robot has a strategy which ensures that it will never consume all of its energy outside the docking station, and we also get some strategy which ensures it. For each "good" initial configuration, we also find out the minimal sufficient amount of initial energy. We note that the energy is not a part of the states of the model. If it was, the problem would be much simpler. We could simply compute the set of states from which Min has a strategy to get the play to a state where the robot has zero energy and it is not in the docking station. However, making the energy part of the states would cause an enormous increase in the number of states and make the model unmanageable.

The second model is called "supply" and models a truck which delivers material to various locations the selection of which is beyond its control. The goal is to never run out of the material so that the truck is always able to satisfy each delivery request. We also want to know the minimal sufficient initial amount of the material.

The third model is called "taxi" and models a taxi which transports people at their request. Its operation costs money and the taxi also earns money. The goal is to never run out of money, and we also want to know the minimal sufficient initial amount of money.

To get MPGs of manageable size, the models are, of course, very simplified, but they are still much closer to real world problems than the synthetic MPGs.

## 4.2 Results

The experiments were carried out on a machine equipped with two dual-core Intel® Xeon® 2.00GHz processors and 16GB of RAM, running GNU/Linux kernel version 2.6.26. All algorithms were implemented in C++ and compiled with GCC 4.3.2 with the "-O2" option.

Table 1 gives the results of our experiments. The first column of the table contains names of the input MPGs. Numbers of vertices and edges, in thousands, are in brackets. The MPGs prefixed by "sqnc", "lnc", and "pnc" were generated by the TOR generator. They all contain $2^{18}$ vertices. The MPGs prefixed by "rand" were generated by the SPRAND generator. Both for the rand5 and rand10 family, we experimented with three sizes of graphs, namely, with $2^{18}$ vertices – no suffix, with $2^{19}$ vertices – suffix "b", and with $2^{20}$ vertices – suffix "h". Finally, the MPGs prefixed by "collect", "supply", and "taxi" are the models of simple reactive systems created by ourselves. For each model, we tried two different values of parameters.

Each MPG used in the experiments has eight columns in the table. Each column headed by a name of an algorithm contains execution times of that algorithm in seconds, excluding the time for reading input. The term "n/a" means more than 10 hours. The first four columns contain results for the lower-bound problem, the last four columns contain the results for the lower-weak-upper-bound problem, which contains a bound b as a part of the input. If the bound is too high, the algorithms essentially solve the

| | | lower-bound | | | | lower-weak-upper-bound | | | |
|---|---|---|---|---|---|---|---|---|---|
| MPG | | VI | VI + BV | VI + SW | KASI | VI | VI + BV | VI + SW | KASI |
| sqnc01 | (262k 524k) | n/a | 31.22 | 55.40 | 17.83 | 13.28 | 19.41 | 43.54 | 9.06 |
| sqnc02 | (262k 524k) | n/a | 13.30 | 20.14 | 11.01 | 2.88 | 10.70 | 17.52 | 3.57 |
| sqnc03 | (262k 525k) | n/a | 3.18 | 3.54 | 1.58 | 0.75 | 3.20 | 3.55 | 1.07 |
| sqnc04 | (262k 532k) | n/a | 9.34 | 11.49 | 8.48 | 1.65 | 8.55 | 10.70 | 2.53 |
| sqnc05 | (262k 786k) | n/a | 10.45 | 14.24 | 4.89 | 1.20 | 10.17 | 13.95 | 1.72 |
| lnc01 | (262k 524k) | 60.79 | 67.89 | 111.32 | 11.31 | 17.49 | 27.85 | 71.19 | 5.99 |
| lnc02 | (262k 524k) | 57.63 | 63.99 | 93.89 | 10.34 | 14.68 | 24.04 | 53.87 | 5.06 |
| lnc03 | (262k 525k) | n/a | 3.30 | 4.39 | 1.48 | 0.73 | 3.34 | 4.41 | 1.03 |
| lnc04 | (262k 528k) | n/a | 21.05 | 25.28 | 10.63 | 3.53 | 11.65 | 15.64 | 4.24 |
| lnc05 | (262k 786k) | n/a | 10.89 | 11.30 | 4.77 | 1.17 | 10.64 | 11.03 | 1.65 |
| pnc01 | (262k 2097k) | n/a | 24.27 | 16.08 | 3.80 | 1.41 | 24.31 | 16.08 | 1.98 |
| pnc02 | (262k 2097k) | n/a | 25.49 | 15.37 | 3.80 | 1.43 | 25.55 | 15.38 | 1.98 |
| pnc03 | (262k 2098k) | n/a | 23.48 | 17.66 | 3.86 | 1.48 | 23.53 | 17.66 | 2.04 |
| pnc04 | (262k 2101k) | n/a | 26.36 | 25.24 | 3.91 | 1.49 | 26.34 | 25.23 | 2.05 |
| pnc05 | (262k 2359k) | n/a | 27.09 | 29.69 | 4.71 | 1.97 | 27.15 | 29.69 | 2.51 |
| rand5 | (262k 1310k) | n/a | 19.16 | 20.42 | 4.55 | 1.65 | 19.27 | 20.54 | 2.39 |
| rand5b | (524k 2621k) | n/a | 36.29 | 33.06 | 10.09 | 3.53 | 36.52 | 33.24 | 5.17 |
| rand5h | (1048k 5242k) | n/a | 86.55 | 59.01 | 21.35 | 7.45 | 87.16 | 59.48 | 11.07 |
| rand10 | (262k 2621k) | n/a | 39.30 | 36.96 | 5.60 | 2.37 | 39.39 | 37.00 | 3.68 |
| rand10b | (524k 5242k) | n/a | 105.69 | 43.43 | 14.54 | 5.07 | 105.97 | 43.45 | 7.98 |
| rand10h | (1048k 10485k) | n/a | 140.46 | 110.68 | 29.27 | 11.38 | 140.57 | 110.82 | 17.52 |
| collect1 | (636k 3309k) | 996.08 | 1027.12 | 1032.55 | 5.68 | 531.40 | 544.77 | 563.78 | 4.89 |
| collect2 | (636k 3309k) | 338.56 | 352.45 | 367.12 | 5.70 | 181.35 | 189.17 | 208.52 | 4.89 |
| supply1 | (363k 1014k) | 6956.23 | 16.03 | 109.72 | 1.79 | 7.72 | 8.87 | 102.97 | 1.85 |
| supply2 | (727k 2030k) | 28046.54 | 65.08 | 449.47 | 3.64 | 30.84 | 33.31 | 418.88 | 3.77 |
| taxi1 | (509k 979k) | 11.64 | 12.85 | 13.16 | 1.29 | 0.70 | 1.49 | 2.17 | 1.38 |
| taxi2 | (509k 979k) | 6.00 | 7.03 | 7.51 | 1.29 | 0.70 | 1.49 | 2.17 | 1.38 |

Table 1: Runtimes of the experiments (in seconds)

lower-bound problem, and so the runtimes are practically the same as for the lower-bound problem. If the bound is too low, all vertices in our inputs have infinite $\mathsf{lwub}_b^\Gamma$ value, and they become very easy to solve. We tried various values of $b$, and for this paper, we selected as $b$ the average $\mathsf{lb}^\Gamma$ value of the vertices with finite $\mathsf{lb}^\Gamma$ value divided by 2, which seems to be a reasonable amount so that the results provide insight. We note that smaller $b$ makes the computation of VI and KASI faster. However, the BV and SW parts of VI + BV and VI + SW always perform the same work, and so for $b \ll (|V|-1)\cdot W$, the combined algorithms are often slower than VI alone.

The table shows that the algorithm KASI was the fastest on all inputs for the lower-bound problem. For the lower-weak-upper-bound problem it was never slower than the fastest algorithm by more than a factor of 2, and for some inputs it was significantly faster. This was true for all values of $b$ that we tried. Therefore, the results clearly suggest that KASI is the best algorithm. In addition, there are several other interesting points.

VI is practically unusable for solving the lower-bound problem for MPGs with some vertices with $\nu < 0$. Except for lnc01–02, collect1–2, and taxi1–2, all input MPGs had vertices with $\nu < 0$. The preprocessing by BV and SW reduces the execution time by orders of magnitude for these MPGs. On the other hand, for the lower-weak-upper-bound problem for the bound we selected, VI is often very fast and the preprocessing slows the computation down in most cases. VI was even faster than KASI on a lot of inputs. However, the difference was never significant, and it was mostly caused by the initialization phase of the algorithms, which takes more time for the more complex algorithm KASI. Moreover, for some inputs, especially from the “collect” family, VI is very slow. VI makes a lot of iterations for the inputs from the collect family, because the robot can survive for quite long by idling, which consumes a very small amount of energy per time unit. However, it cannot survive by idling forever. The $i$-th iteration of VI computes the minimal sufficient initial energy to keep the energy level non-negative for $i$ time units, and so until the idling does not consume at least as much energy as the minimal sufficient initial energy to keep the energy level non-negative forever, new iterations have to be started. We believe that this is a typical situation for this kind of application. Other inputs for which VI took a lot of time are: sqnc01, lnc01–02, supply1–2.

Finally, we comment on scalability of the algorithms. As the experiments on the SPRAND generated inputs suggest, the runtimes of the algorithms increase no faster than the term $|V| \cdot |E|$, and so they are able to scale up to very large MPGs.

## 5 Conclusion

We proposed a novel algorithm for solving the lower-bound and the lower-weak-upper-bound problems for MPGs. Our algorithm, called Keep Alive Strategy Improvement (KASI), is based on the strategy improvement technique which is very efficient in practice. To demonstrate that the algorithm is able to solve the two problems for large MPGs, we carried out an experimental study. In the study we compared KASI with the value iteration algorithm (VI) from [6, 10], which we also improved by combining it with the algorithm of Björklund and Vorobyov [1] (BV) and the algorithm of Schewe (SW). KASI is the clear winner of the experimental study.

Two additional results of this paper are the improvement of the complexity of BV, and characterization of Min's optimal strategies w.r.t. the lower-weak-upper-bound problem.

## References

[1] H. Björklund and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Math.*, 155(2):210–229, 2007.

[2] P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *Proc. Formal Modeling and Analysis of Timed Systems*, pages 33–47. Springer, 2008.

[3] L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin. Faster algorithms for mean-payoff games. *submitted to FMSD*, 2010.

[4] A. Chakrabarti, L. de Alfaro, T. Henzinger, and M. Stoelinga. Resource interfaces. In *Proc. Embedded Software*, volume 2855 of *LNCS*, pages 117–133. Springer, 2003.

[5] J. Chaloupka. Parallel algorithms for mean-payoff games: An experimental evaluation. In *Proc. European Symposium on Algorithms*, volume 5757 of *LNCS*, pages 599–610. Springer, 2009.

[6] J. Chaloupka and L. Brim. Faster algorithm for mean-payoff games. In *Proceedings of the 5th Doctoral Wokrshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2009)*, pages 45–53. NOVPRESS, 2009.

[7] B. V. Cherkassky and A. V. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, 85:277–311, 1999.

[8] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.

[9] J. Cochet-Terrasson and S. Gaubert. A policy iteration algorithm for zero-sum stochastic games with mean payoff. *Comptes Rendus Mathematique*, 343:377–382, 2006.

[10] L. Doyen, R. Gentilini, and J.-F. Raskin. Faster pseudopolynomial algorithms for mean-payoff games. Technical Report 2009.120, Université Libre de Bruxelles (ULB), Bruxelles, Belgium, 2009.

[11] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

[12] L. Georgiadis, A. V. Goldberg, R. E. Tarjan, and R. F. Werneck. An experimental study of minimum mean cycle algorithms. In *Proc. Workshop on Algorithm Engineering and Experiments*, pages 1–13. SIAM, 2009.

[13] Andrew Goldberg's network optimization library. `http://www.avglab.com/andrew/soft.html`, October 2009.

[14] V. A. Gurvich, A. V. Karzanov, and L. G. Khachivan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Comput. Math. and Math. Phys.*, 28(5):85–91, 1988.

[15] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

[16] Y. Lifshits and D. Pavlov. Fast exponential deterministic algorithm for mean payoff games. *Zapiski Nauchnyh Seminarov POMI*, 340:61–75, 2006.

[17] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Proc. Computer Science Logic*, volume 5213 of *LNCS*, pages 369–384. Springer, 2008.

[18] U. Zwick and M. S. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1–2):343–359, 1996.

# 6 Appendix

## 6.1 Modified Dijkstra's Algorithm

```
proc DIJKSTRA($G_\pi = (V, E_\pi, w), b, X, d_p$)
  $S = \{v \in V \mid d_p(v) > -\infty\}$
  foreach $v \in S$ do $key(v) := -\infty$ od
  foreach $v \in X$ do
    $key(v) := 0$
    $q.enqueue(v)$
    $queued(v) := true$
  od
  [$q$ is a maximum priority queue of vertices, where the priority of vertex $v$ is $key(v)$]
  while $\neg q.empty()$ do
    $u := q.dequeue()$
    foreach $(v, u) \in E_\pi$ do
      if $v \in S \land v \notin X$ then
        $tmp := key(u) + w(v, u) - d_p(v) + d_p(u)$
        if $d_p(v) + tmp \geq -b \land tmp > key(v)$ then
          $key(v) := tmp$
          if $\neg queued(v)$ then $q.enqueue(v)$; $queued(v) := true$ fi
        fi
      fi
    od
  od
  foreach $v \in S$ do $d_o(v) := d_p(v) + key(v)$ od
  foreach $v \in V \setminus S$ do $d_o(v) = -\infty$ od
  return $d_o$
end
```

Figure 4: Modified Dijkstra's algorithm

In Figure 4 is the pseudocode of the modified Dijkstra's algorithm used in the strategy evaluation procedure EVALUATESTRATEGY(). The input to DIJKSTRA() consists of four parts. The first part is a directed graph $G_\pi$. The graph is denoted by $G_\pi$ to emphasize which graph the main algorithm KASI computes longest paths in. The second part of the input is a bound $b$, the third part is a set $X$, to which the algorithm computes longest paths, and finally the fourth part is a vector of integers $d_p$ used for potential transformation of edge-weights. The vector $d_p$ contains longest path weights computed in different setting. DIJKSTRA() computes the difference between $d_p$ and the longest path weights in the current setting and then adds the difference to $d_p$, thus obtaining the current longest path weights, which are returned as output.

DIJKSTRA() works only with the vertices in the set $S$ computed on line 2. The set $S$ contains vertices from $V$ with finite $d_p$ value. On line 3, the tentative distance $\mathit{key}(v)$ of each vertex $v \in S$ is initialized to $-\infty$. The algorithm computes longest paths to the set $X$, and so the initialization phase on lines 4–8 sets $\mathit{key}(v)$ of each vertex $v \in X$ to the (final) value $0$. Each vertex from $X$ is also put to the maximum priority queue $q$ and its presence in the queue is recorded in the vector $\mathit{queued}$. The input to DIJKSTRA() always guarantees that $X \subseteq S$. The priority of a vertex $v$ in the maximum priority queue $q$ is $\mathit{key}(v)$.

The main loop on lines 10–21 differs from the standard Dijkstra's algorithm only in the following. For each $v \in S$, the updates of $\mathit{key}(v)$ that do not lead to path weight greater or equal to $-b$ are ignored.

On line 22, the longest path weight $d_o(v)$ for each $v \in S$ is computed as $d_p(v)+\mathit{key}(v)$. Please note that $\mathit{key}(v)$ might be equal to $-\infty$, and so $d_o(v)$ is set to $-\infty$ too. These are the vertices for which there is no path to $X$ in $G(S)$ or each such path has a suffix of weight less than $-b$.

On line 23, for each vertex $v \in V \setminus S$ ($d_p(v) = -\infty$), $d_o(v)$ is simply set to $-\infty$. The vector $d_o$ is then returned as output on line 24.

## 6.2 Proofs

**Lemma 6.1** *Let $(\Gamma, b)$ be an instance of the lower-weak-upper-bound problem. Let further $\pi \in \Pi_b^\Gamma$, and finally let $d_{-1} \in (\mathbb{Z} \cup \{-\infty\})^V$ be such that for $A = \{v \in V \mid d_{-1}(v) = 0\}$ and $D = \{v \in V \mid d_{-1}(v) > -\infty\}$, the conditions (i.) and (ii.) hold. Then in* EVALUATESTRATEGY$(\Gamma, b, \pi, d_{-1})$*, the following is an invariant of the while-loop on lines 3–7. The invariant holds just after line 4 of the algorithm. That is after* DIJKSTRA() *was executed and before the variable $i$ is incremented. Let*

$$D_i = \{v \in D \mid d_i(v) > -\infty\}$$

*then the following holds.*

1. $(\forall v \in D)(d_i(v) = \max\{w(p) |\ p \in \mathit{path}^{G_\pi(D)}(v, B_i) \land$
   $(\forall p_f \in \mathit{suffix}(p))(w(p_f) \geq -b)\ \}$
2. $d_i \leq d_{i-1} \land ((\exists v \in D_{i-1})(\forall (v,u) \in E_\pi)(d_{i-1}(v) > d_{i-1}(u) + w(v,u)) \Rightarrow d_i < d_{i-1})$
3. $(\forall v \in B_i)(d_i(v) = 0)$
4. $(\forall v \in D_i \setminus B_i)(d_i(v) < 0)$
5. $(\forall v \in D_i \setminus B_i)(\forall (v,u) \in E_\pi)(d_i(v) \geq d_i(u) + w(v,u))$
6. $(\forall c \in \mathit{cycle}^{G(D_i \setminus B_i)})(w(c) < 0)$
7. $(\forall v \in D \setminus D_i)(\forall p \in \mathit{path}^{G_\pi(D)}(v, B_i))(\exists p_f \in \mathit{suffix}(p))(w(p_f) < -b)$
8. $(\forall c \in \mathit{cycle}^{G(D \setminus B_i)})(w(c) \geq 0 \Rightarrow (\exists p_s \in \mathit{segment}(c))(w(p_s) < -b))$
9. $(\forall v \in V \setminus D)(d_i(v) = -\infty)$

**Proof:** In the whole proof, all references to lines in pseudocode are references to lines in pseudocode of EVALUATESTRATEGY() in Figure 1. We will prove the lemma by induction on $i$. For technical convenience, we define $B_{-1} = \{v \in V \mid d_{-1}(v) = 0\}$ and start the induction from $-1$.

$i = -1$. We do not require that the property 1 holds for $i = -1$, and so we are not allowed to use it when we invoke induction hypothesis. Also, property 2 cannot be proved, because neither $d_{-2}$ nor $D_{-2}$ is defined. However, the remaining properties are satisfied. It holds that $D_{-1} = D$ and $B_{-1} = A$, and so the properties 3–6 follow directly from the assumptions of the lemma, the properties 7–8 hold trivially, and the property 9 follows from the definition of the set $D$.

$i > -1$. Let's suppose that the invariant holds for $i - 1$. We now show that it holds also for $i$. Recall that $D_{-1} = D$. To show the property 1, we have to prove that DIJKSTRA() on line 4 terminates and gives correct longest path weights. DIJKSTRA() works only with the graph $G_\pi(D_{i-1})$, where the $d_{i-1}$ value of each vertex is finite. For each vertex $v \in V \setminus D_{i-1}$, it simply assigns $-\infty$ to $d_i(v)$. This immediately implies that:

$$D_i \subseteq D_{i-1}$$

Therefore, to prove that DIJKSTRA() gives correct results, we need to show that for each vertex $v \in D \setminus D_{i-1}$, each path from $v$ to $B_i$ has a suffix of weight less than $-b$, and for each $v \in D_{i-1} \setminus B_i$, each edge $(v, u)$ emanating from $v$ is non-positive with the potential transformation, i.e., $w(v, u) - d_{i-1}(v) + d_{i-1}(u) \leq 0$.

If $v \in D_{i-1} \setminus B_{i-1}$, then by the induction hypothesis, part 5, we have $w(v, u) - d_{i-1}(v) + d_{i-1}(u) \leq 0$ for each edge $(v, u)$ emanating from $v$. Because of the way $B_i$ is computed (see line 2 and line 6 of the pseudocode) it holds that:

$$B_i \subseteq B_{i-1}$$

For $v \in B_{i-1} \setminus B_i$, we even have strict inequality $w(v, u) - d_{i-1}(v) + d_{i-1}(u) < 0$ for each edge $(v, u)$ emanating from $v$. For $i = 0$ this follows from way $B_0$ was computed on line 2 of the pseudocode. For $i > 0$ the inequality follows from the way $B_i$ was computed on line 6. To complete the proof of correct termination of DIJKSTRA(), we show that for $v \in D \setminus D_{i-1}$, each path from $v$ to $B_i$ contains a suffix of weight less than $-b$.

Let $v \in D \setminus D_{i-1}$. Let $p = (v = v_0, \ldots, v_k) \in \mathsf{path}^{G_\pi(D)}(v, B_i)$, and let $p_f = (v_f, \ldots, v_k)$ be the longest suffix of $p$ with all $d_{i-1}$ values of vertices finite. Let further $v_s$ be the first vertex in $p_f$ such that $v_s \in B_{i-1}$. Since $B_i \subseteq B_{i-1}$, there must be such a vertex. Let $p_s = (v_s, \ldots, v_k)$. Please note that $p_s$ must be entirely in $D_{i-1}$. Therefore, from what was proved above it follows that for each $j \in \{s, s+1, \ldots, k-1\}$, $w(v_j, v_{j+1}) - d_{i-1}(v_j) + d_{i-1}(v_{j+1}) \leq 0$. If we sum the inequalities, we get $w(p_s) - d_{i-1}(v_s) + d_{i-1}(v_k) \leq 0$, and since $v_k \in B_i \subseteq B_{i-1}$, $d_{i-1}(v_k) = 0$, and so $w(p_s) \leq d_{i-1}(v_s)$.

By the induction hypothesis, part 3–4, $d_{i-1}(v_s) \leq 0$, and so $w(p_s) \leq 0$. By the choice of $p_f$, $d_{i-1}(v_{f-1}) = -\infty$, which means that $v_{f-1} \in D \setminus D_{i-1}$, and so by the induction hypothesis, part 7, each path from $v_{f-1}$ to $B_{i-1}$ has a suffix of weight less than $-b$. In particular, the path $(v_{f-1}, v_f, \ldots, v_s)$ has a suffix of weight less than $-b$, and since $w(p_s) \leq 0$, so does does the path $(v_{f-1}, v_f, \ldots, v_s, \ldots, v_k)$. Therefore, since $p_f$ is a suffix of $p$, also $p$ has a suffix of weight less than $-b$. Together we have that DIJKSTRA() terminates and gives us the vector $d_i$ such that the property 1 holds.

Let's now prove the properties 2–9. The property 3 follows directly from the property 1. We now show the property 2. More specifically, we will show that for each $v \in V \setminus X$, $d_i(v) \leq d_{i-1}(v)$, and for each $v \in X$, $d_i(v) < d_{i-1}(v)$, where $X = \{v \in V \mid (\forall (v, u) \in E_\pi)(d_{i-1}(v) > d_{i-1}(u) + w(v, u))\}$.

It holds that $X \subseteq D_{i-1}$, because for each $v \in V \setminus D_{i-1}$, $d_{i-1}(v) = -\infty$, and so there cannot be an edge $(v, u)$ such that $d_{i-1}(v) > d_{i-1}(u) + w(v, u)$. Therefore, for $v \in V \setminus D_{i-1}$

we need to prove just the inequality $d_i(v) \le d_{i-1}(v)$. This is easy, because DIJKSTRA() assigns $-\infty$ to $d_i(v)$.

It also holds that $X \cap B_i = \emptyset$, because for each $v \in B_i$ there is an edge $(v, u)$ such that $w(v, u) - d_{i-1}(v) + d_{i-1}(u) \ge 0$. Therefore, for $v \in B_i$ we also need to prove just the inequality $d_i(v) \le d_{i-1}(v)$. By the property 3, $d_i(v) = 0$, and since $B_i \subseteq B_{i-1}$, $d_{i-1}(v) = 0$ holds by the induction hypothesis, part 3, and so $d_i(v) \le d_{i-1}(v)$. It remains to show that $d_i(v) \le d_{i-1}(v)$ holds for $v \in D_{i-1} \setminus X$, and for $v \in X$, it holds $d_i(v) < d_{i-1}(v)$.

Let $v \in D_{i-1} \setminus X$. If there is no path from $v$ to $B_i$ in $D_{i-1}$, then $d_i(v) = -\infty$, and so $d_i(v) \le d_{i-1}(v)$. If there is a path from $v$ to $B_i$ in $D_{i-1}$, then let $p = (v = v_0, \dots, v_k) \in \mathsf{path}^{G_\pi(D_{i-1})}(v, B_i)$. If $v_j \in p \cap X$, then $w(v_j, v_{j+1}) - d_{i-1}(v_j) + d_{i-1}(v_{j+1}) < 0$. If $v_j \in p \cap (D_{i-1} \setminus X)$, then $w(v_j, v_{j+1}) - d_{i-1}(v_j) + d_{i-1}(v_{j+1}) \le 0$. So together we have $w(v_j, v_{j+1}) - d_{i-1}(v_j) + d_{i-1}(v_{j+1}) \le 0$ for all $j \in \{0, \dots, k-1\}$. By summing the inequalities we get $w(p) - d_{i-1}(v) + d_{i-1}(v_k) \le 0$, and since $v_k \in B_i \subseteq B_{i-1}$, $d_{i-1}(v_k) = 0$, and so $w(p) \le d_{i-1}(v)$. Moreover, if some vertex in $p$ other than $v_k$ is from X, then we have strict inequality $w(p) < d_{i-1}(v)$. Therefore, the weight of a longest path from $v$ to $B_i$ cannot exceed $d_{i-1}(v)$, and so $d_i(v) \le d_{i-1}(v)$.

Let finally $v \in X$. Then the inequality $d_i(v) < d_{i-1}(v)$ follows from the previous paragraph, because each path from $v$ to $B_i$ contains a vertex from X which is not the last vertex in the path, namely the vertex $v$. This completes the proof of the property 2. We can now prove also the property 4.

It holds that $B_{i-1} \setminus B_i \subseteq X$, it follows from the way $B_i$ was computed. Therefore, for each $v \in B_{i-1} \setminus B_i$, $d_i(v) < d_{i-1}(v) = 0$. Since $d_i(v) \le d_{i-1}(v)$ holds for each $v \in V$, for $v \in D_i \setminus B_{i-1}$ we have $d_i(v) \le d_{i-1}(v) < 0$. The inequality $d_{i-1}(v) < 0$ holds by the induction hypothesis, part 4, because $D_i \subseteq D_{i-1}$. Together we have $d_i(v) < 0$ for each $v \in D_i \setminus B_i$, hence the property 4 is proved.

The property 5 follows from the property 1, because of the properties of weights of longest paths. We will now prove the property 6.

Let $c = (v_0, \dots, v_{k-1}, v_0) \in \mathsf{cycle}^{G(D_i \setminus B_i)}$. If c is in $D_i \setminus B_{i-1}$, then it is also in $D_{i-1} \setminus B_{i-1}$ and its negativeness follows from the induction hypothesis, part 6. Otherwise, it contains a vertex from $B_{i-1} \setminus B_i$. If $v_j \in c \cap (B_{i-1} \setminus B_i)$, then $w(v_j, v_{j+1}) - d_{i-1}(v_j) + d_{i-1}(v_{j+1}) < 0$. If $v_j \in c \cap (D_i \setminus B_{i-1})$, then $w(v_j, v_{j+1}) - d_{i-1}(v_j) + d_{i-1}(v_{j+1}) \le 0$. So together we have that $w(v_j, v_{j+1}) - d_{i-1}(v_j) + d_{i-1}(v_{j+1}) \le 0$ for all $j \in \{0, \dots, k-1\}$ and for at least one vertex the inequality is strict. By summing the inequalities for each

$v_j \in c$, we get that $w(c) < 0$. Therefore, the cycle c is negative and the property 6 is proved.

The property 7 follows from the property 1, because for each $v \in D \setminus D_i$, $d_i(v) = -\infty$. Let's now show the property 8.

Let $c = (v_0, \dots, v_{k-1}, v_0) \in \mathsf{cycle}^{G(D \setminus B_i)}$. If c is completely in $D \setminus B_{i-1}$, then the desired properties follow from the induction hypothesis, part 8, so let's assume that c contains a vertex from $B_{i-1} \setminus B_i$. Let's further assume that the cycle is in $D_{i-1}$, and so for each $v_j \in c$, $d_{i-1}(v_j)$ is finite. If $v_j \in B_{i-1} \setminus B_i$ then $w(v_j, v_{j+1}) - d_{i-1}(v_j) + d_{i-1}(v_{j+1}) < 0$. If $v_j \in D_{i-1} \setminus B_{i-1}$, then $w(v_j, v_{j+1}) - d_{i-1}(v_j) + d_{i-1}(v_{j+1}) \leq 0$. So together we have that $w(v_j, v_{j+1}) - d_{i-1}(v_j) + d_{i-1}(v_{j+1}) \leq 0$ for all $j \in \{0, \dots, k-1\}$ and for at least one vertex the inequality is strict. By summing the inequalities for each $v_j \in c$, we get that $w(c) < 0$.

Now let's assume that some $v_j \in c$ has $d_{i-1}(v_j) = -\infty$. It follows that $v_j \in D \setminus D_{i-1}$. The cycle c contains a vertex from $B_{i-1} \setminus B_i$, so let $v_s$ be the first such vertex following $v_j$ in c. Then by induction hypothesis, part 7, the path $p_s = (v_j, v_{j+1}, \dots, v_s) \in \mathsf{segment}(c)$ has a suffix of weight less than $-b$. The suffix is also a segment of c, and so c contains a segment of weight less than $-b$, which completes the proof of the property 8.

Finally, the property 9 follows from the fact that DIJKSTRA() works only with the graph $G_\pi(D_{i-1})$ and for each $v \in V \setminus D_{i-1}$, it simply assigns $-\infty$ to $d_i(v)$. Therefore, since $V \setminus D \subseteq V \setminus D_{i-1}$, it holds that for each $v \in V \setminus D$, $d_i(v) = -\infty$. ■

**Lemma 6.2** *Let* $(\Gamma, b)$ *be an instance of the lower-weak-upper-bound problem. Let further* $\pi \in \Pi_M^\Gamma$, *and finally let* $d_{-1} \in (\mathbb{Z} \cup \{-\infty\})^V$ *be such that for* $A = \{v \in V \mid d_{-1}(v) = 0\}$ *and* $D = \{v \in V \mid d_{-1}(v) > -\infty\}$, *the conditions (i.) and (ii.) hold. Then* EVALUATESTRATEGY$(\Gamma, b, \pi, d_{-1})$ *terminates and for* $d :=$ EVALUATESTRATEGY$(\Gamma, b, \pi, d_{-1})$ *it holds that for each* $v \in B = \{v \in V \mid d(v) = 0\}$, *there is* $(v, u) \in E_\pi$ *such that* $w(v, u) - d(v) + d(u) \geq 0$.

**Proof:** Let's first prove that EVALUATESTRATEGY$(\Gamma, b, \pi, d_{-1})$ terminates. In the while loop on lines 3–7 of EVALUATESTRATEGY(), no elements are added to the set $B_i$ and when no element is removed, the while-loop terminates, which terminates the whole strategy evaluation algorithm. The set $B_i$ can have at most $|V|$ elements, and so the only thing that can cause that EVALUATESTRATEGY() does not terminate is the procedure DIJKSTRA(). However, there is only one loop that can prevent DIJKSTRA() from terminating. Namely, the while-loop on lines 10–21. This loop terminates when the priority queue q is empty, and since each vertex can be put to q only once and in each iteration one vertex is removed from q, the termination of the loop is guaranteed. Therefore, DIJKSTRA() (and so also EVALUATESTRATEGY()) always terminates.

After the termination of the while loop on lines 3–7 of EVALUATESTRATEGY(), it holds that $B = B_i = B_{i-1}$, and $d = d_{i-1}$. The set $B_i$ was computed by removing all vertices $v$ with the following property from $B_{i-1}$. There is no edge $(v, u) \in E_\pi$ such that $w(v, u) - d_{i-1}(v) + d_{i-1}(u) \geq 0$. Therefore, since no vertex was removed in the last iteration and since $d = d_{i-1}$ and $B = B_i$, the claim of the lemma holds. ■

**Lemma 6.3** *Let* $G = (V, E, w)$ *be a graph and let* $b \in \mathbb{N}_0$ *such that*

$$(\forall c \in \textit{cycle}^G)(w(c) \geq 0 \Rightarrow (\exists p_s \in \textit{segment}(c))(w(p_s) < -b))$$

*Then each infinite path* $p = (v_0, v_1, v_2, \ldots) \in \textit{path}^G_\infty$ *satisfies at least one of the following properties.*

1. $(\forall x \in \mathbb{N}_0)(\exists n \in \mathbb{N})(\sum_{i=0}^{n-1} w(v_i, v_{i+1}) < x)$
2. $(\exists n_1, n_2 \in \mathbb{N}_0)(n_1 < n_2 \wedge \sum_{i=n_1}^{n_2-1} w(v_i, v_{i+1}) < -b)$

**Proof:** Let $G$ and $b$ satisfy the assumptions of the lemma and let $p = (v_0, v_1, v_2, \ldots) \in \textit{path}^G_\infty$.

It is possible to partition each prefix $p_f$ of $p$ into a set of cycles plus some path with at most $|V| - 1$ vertices. We can do it in the following way: Start from $v_0$ and go along $p_f$ until an already visited vertex is encountered, remove the found cycle from the path, leaving only the first vertex of the cycle, and start over. Continue this process until we are left with a path with no repeated vertex.

Let $W = \max_{e \in E} |w(e)|$, $x \in \mathbb{N}_0$, and let $p_f$ be a long enough prefix of $p$, namely $l(p_f) > (|V| - 1) + (W \cdot (|V| - 1) + x + 1) \cdot |V|$. Then there are at least $(W \cdot (|V| - 1) + x + 1)$ cycles in the partition of $p_f$. If all the cycles are negative, then $w(p_f) < x$. The reason is that the weight of the remaining path in the partition is at most $W \cdot (|V| - 1)$ and the total weight of $(W \cdot (|V| - 1) + x + 1)$ negative cycles is less than $(W \cdot (|V| - 1) + x)$. Therefore, if all the cycles are negative, then the property 1 is satisfied.

If there is some non-negative cycle $c$ in the partition, then, by assumptions of the lemma, it must have a segment of weight less than $-b$. However, this does not trivially imply that $p_f$ contains a segment of weight less than $-b$, because the cycle might be scattered across the path. More precisely, $c = (v_{i_0}, \ldots, v_{i_{k-1}}, v_{i_k})$ where $v_{i_0} = v_{i_k}$, and for each $j \in \{0, \ldots, k-1\}$, it holds that $i_j < i_{j+1} \wedge (i_{j+1} = i_j + 1 \vee v_{i_{j+1}-1} = v_{i_j})$. If for some path of the form $p_{i_j} = (v_{i_j}, v_{i_j+1}, \ldots, v_{i_{j+1}-1})$ it holds that $l(p_{i_j}) > 0$, then the path $p_{i_j}$ "breaks" the cycle $c$ in $p_f$. However, since $v_{i_{j+1}-1} = v_{i_j}$, the path $p_{i_j}$ itself is a cycle and by the assumptions of the lemma, $p_{i_j}$ is negative or it contains a segment of weight less than $-b$. In the latter case, the property 2 follows. If all the "breaking" paths of $c$ are negative, then since $c$ contains a segment of weight $-b$, so does the segment of $p_f$ starting from $v_{i_0}$ and ending at $v_{i_k}$, and so the property 2 is also satisfied. ■

**Proof:** [Proof of Theorem 3.1] In the whole proof, all references to lines in pseudocode are references to lines in pseudocode of EVALUATESTRATEGY() in Figure 1. The procedure EVALUATESTRATEGY$(\Gamma, b, \pi, d_{-1})$ terminates by Lemma 6.2. Let's now prove that $d(v) = -\mathsf{lwub}_b^{\Gamma_\pi(D)}(v)$ holds for each $v \in D$.

Let $d :=$ EVALUATESTRATEGY$(\Gamma, b, \pi, d_{-1})$, and let

$$\begin{aligned} x_v = -\min\{\quad & x \in \mathbb{N}_0 \mid \\ & (\exists p = (v = v_0, v_1, v_2, \ldots) \in \mathsf{path}_\infty^{G_\pi(D)}) \\ & (\quad (\forall n \in \mathbb{N})(x + \textstyle\sum_{j=0}^{n-1} w(v_j, v_{j+1}) \geq 0) \wedge \\ & \quad (\forall n_1, n_2 \in \mathbb{N}_0)(n_1 < n_2 \Rightarrow \textstyle\sum_{j=n_1}^{n_2-1} w(v_j, v_{j+1}) \geq -b)\ )\ \} \end{aligned}$$

We will first prove that $x_v = -\mathsf{lwub}_b^{\Gamma_\pi(D)}$ for each $v \in D$, and then we will prove that $d(v) = x_v$ for each $v \in D$. Recall that, by definition:

$$\begin{aligned} \mathsf{lwub}_b^{\Gamma_\pi(D)}(v) = \min\{x \in \mathbb{N}_0 \mid \quad & (\exists \sigma \in \Sigma^{\Gamma_\pi(D)})(\forall \pi \in \Pi^{\Gamma_\pi(D)}) \\ & (\quad \mathsf{outcome}^{\Gamma_\pi(D)}(v, \sigma, \pi) = (v = v_0, v_1, v_2, \ldots) \wedge \\ & \quad (\forall n \in \mathbb{N})(x + \textstyle\sum_{j=0}^{n-1} w(v_j, v_{j+1}) \geq 0) \wedge \\ & \quad (\forall n_1, n_2 \in \mathbb{N}_0)(n_1 < n_2 \Rightarrow \textstyle\sum_{j=n_1}^{n_2-1} w(v_j, v_{j+1}) \geq -b)\ )\ \} \end{aligned}$$

Also, recall that the game graph of $\Gamma_\pi(D)$ may be different from $G_\pi(D)$. Namely, self-loops of weight $-1$ are added to vertices with zero out-degree in $G_\pi(D)$. However, for each infinite path containing such a vertex it holds that the weights of its prefixes cannot be bounded from below. Moreover, there is only one strategy of Min in $\Gamma_\pi(D)$. Namely, the strategy

$$\pi'(v) = \begin{cases} \pi(v) & \text{if } \pi(v) \in D \\ v & \text{if } \pi(v) \notin D \end{cases}$$

Therefore, for each $p = (v_0, v_1, v_2, \ldots) \in \mathsf{path}_\infty^{G_\pi(D)}$, there is a strategy $\sigma \in \Sigma^{\Gamma_\pi(D)}$ such that $\mathsf{outcome}^{\Gamma_\pi(D)}(v_0, \sigma, \pi') = p$. Also, for each $u \in D$ and each $\sigma \in \Sigma^{\Gamma_\pi(D)}$ such that the infinite path $\mathsf{outcome}^{\Gamma_\pi(D)}(u, \sigma, \pi') = (u = u_0, u_1, u_2, \ldots)$ satisfies

$$(\exists x \in \mathbb{N}_0)(\forall n \in \mathbb{N})(x + \sum_{j=0}^{n-1} w(u_j, u_{j+1}) \geq 0)$$

and

$$(\forall n_1, n_2 \in \mathbb{N}_0)(n_1 < n_2 \Rightarrow \sum_{j=n_1}^{n_2-1} w(u_j, u_{j+1}) \geq -b)$$

it holds that $\mathsf{outcome}(u, \sigma, \pi') \in \mathsf{path}_\infty^{G_\pi(D)}$. Together, we have $x_v = -\mathsf{lwub}_b^{\Gamma_\pi(D)}(v)$ for each $v \in D$. Let's now prove that $d(v) = x_v$ for each $v \in D$.

Let $B = \{v \in D \mid d(v) = 0\}$ and $v \in D$. We will first prove that $d(v) \geq x_v$. Let $p = (v = v_0, v_1, v_2, \ldots) \in \mathsf{path}_\infty^{G_\pi(D)}$. If the path $p$ does not contain any vertex from $B$ then by Lemma 6.1, part 6 and part 8, and by Lemma 6.3, the weights of prefixes of $p$ cannot be bounded from below or $p$ has a segment of weight less than $-b$, and so $p$ does not contribute to $x_v$.

If $p$ contains a vertex from $B$, then let $v_f$ be the first vertex from $B$ in the path $p$, and let $p_f = (v = v_0, v_1, \ldots, v_f)$. Let further $D' = \{v \in D \mid d(v) > -\infty\}$ and let's consider two cases.

The first case is that there is some $j \in \{0, \ldots, f-1\}$ such that $v_j \notin D'$. Then by Lemma 6.1, part 7, the path $p$ contains a segment of weight less than $-b$, and so $p$ does not contribute to $x_v$.

The second case is that for all $j \in \{0, \ldots, f-1\}$, it holds that $d(v_j) > -\infty$. Then by Lemma 6.1, part 5, it holds that $w(p_f) - d(v) + d(v_f) \leq 0$, and by Lemma 6.1, part 3, the inequality simplifies to $w(p_f) \leq d(v)$.

So far, we have proved that for each $p = (v = v_0, v_1, v_2, \ldots) \in \mathsf{path}_\infty^{G_\pi(D)}$, at least one of the following properties hold.

1. $(\forall x \in \mathbb{N}_0)(\exists n \in \mathbb{N})(\sum_{j=0}^{n-1} w(v_j, v_{j+1}) < x)$
2. $(\exists n_1, n_2 \in \mathbb{N}_0)(n_1 < n_2 \wedge \sum_{j=n_1}^{n_2-1} w(v_j, v_{j+1}) < -b)$
3. $(\exists n \in \mathbb{N})(\sum_{j=0}^{n-1} w(v_j, v_{j+1}) \leq d(v))$

Therefore, for each $v \in D$, $d(v) \geq x_v$. Let's now prove that it also holds that $d(v) \leq x_v$, hence $d(v) = x_v$. If $v \notin D'$, then $d(v) = -\infty$, and so $d(v) \leq x_v$ holds trivially. For $v \in D'$, we will prove the inequality by finding an infinite path starting from $v$ such that it does not contain a segment of weight less than $-b$ and the weight of each prefix of the path is greater or equal to $d(v)$.

Let $\nu \in D'$. We will construct the desired path inductively. Let $p_k = (\nu = \nu_0, \dots, \nu_k) \in \mathsf{path}^{G_\pi(D)}$ be an already constructed path, we will lengthen it by one vertex using the following procedure:

- If $\nu_k \in D' \setminus B$, then, by Lemma 6.1, part 1, there is a vertex $u \in D'$ such that $(\nu_k, u) \in E_\pi$ and $d(\nu_k) = d(u) + w(\nu_k, u)$. We set $\nu_{k+1} = u$.
- If $\nu_k \in B$, then, by Lemma 6.2, there is a vertex $u \in D'$ such that $(\nu_k, u) \in E_\pi$ and $d(\nu_k) \leq d(u) + w(\nu_k, u)$. We set $\nu_{k+1} = u$.

This way we form an infinite path $p = (\nu = \nu_0, \nu_1, \nu_2, \dots)$ such that for each $j \in \mathbb{N}_0$, it holds that $d(\nu_j) \leq d(\nu_{j+1}) + w(\nu_j, \nu_{j+1})$. Therefore, for each prefix $p_k = (\nu = \nu_0, \dots, \nu_k)$ of $p$, it holds that $d(\nu) \leq d(\nu_k) + w(p_k)$, and since by Lemma 6.1, part 3 and part 4, $d(\nu_k) \leq 0$, we have $d(\nu) \leq w(p_k)$. It remains to show that $p$ does not contain a segment of weight less than $-b$.

Let $n_1, n_2 \in \mathbb{N}_0$ such that $n_1 < n_2$. By the same reasoning as in the previous paragraph, we get that $d(\nu_{n_1}) \leq d(\nu_{n_2}) + \sum_{j=n_1}^{n_2-1} w(\nu_j, \nu_{j+1})$, and so $d(\nu_{n_1}) - d(\nu_{n_2}) \leq \sum_{j=n_1}^{n_2-1} w(\nu_j, \nu_{j+1})$. By Lemma 6.1, part 1, 3, and 4, it holds that $-b \leq d(\nu_{n_1}), d(\nu_{n_2}) \leq 0$. Therefore, $d(\nu_{n_1}) - d(\nu_{n_2}) \geq -b$, and the theorem is proved. ■

**Proof:** [Proof of Lemma 3.3] In the whole proof, all references to lines in pseudocode are references to lines in pseudocode of LOWERWEAKUPPERBOUND() in Figure 2. We will proceed by induction on $i$.

If $i = 0$, then we have to prove only the first part of the lemma. Since $d_{-1} = (0)_{\nu \in V}$, it holds that both the set $A = \{\nu \in V \mid d_{-1}(\nu) = 0\}$ and the set $D = \{\nu \in V \mid d_{-1}(\nu) > -\infty\}$ are be equal to $V$, i.e., $A = D = V$. Therefore, each vertex from $D \setminus A$ and each cycle from $\mathsf{cycle}^{G_{\pi_0}(D)}$ satisfies whatever we want, because there is no such vertex and no such cycle, and so conditions (i.) and (ii.) hold. The assumptions of Theorem 3.1 is satisfied. Let now $i > 0$ and suppose the lemma holds for $i - 1$.

By the induction hypothesis, $\Gamma$, $b$, $\pi_{i-1}$, and $d_{i-2}$ satisfy the assumptions of Theorem 3.1 (and also Lemma 6.1). Therefore, by Lemma 6.1, the following holds for $d_{i-1}$ returned by EVALUATESTRATEGY(). Let $A' = \{\nu \in V \mid d_{i-1}(\nu) = 0\}$, and $D' = \{\nu \in V \mid d_{i-1}(\nu) > 0\}$. Then for each $c \in \mathsf{cycle}^{G_{\pi_{i-1}}(D' \setminus A')}$, $w(c) < 0$, by Lemma 6.1, part 6. Also, for each $\nu \in D' \setminus A'$, $d_{i-1}(\nu) < 0$, by Lemma 6.1, part 4, and for each edge $(\nu, u) \in E_{\pi_{i-1}}$ emanating from $\nu$, $d_{i-1}(\nu) \geq d_{i-1}(u) + w(\nu, u)$, by Lemma 6.1, part 5. We will now show that the above holds also for the graph $G_{\pi_i}(D' \setminus A')$, which will prove the first part of the lemma.

The strategy $\pi_{i-1}$ is modified on lines 10–18 to form the new strategy $\pi_i$. $\pi_i$ may differ from $\pi_{i-1}$ only on vertices from $D' \cap V_{\mathrm{Min}}$. For each $v \in D' \cap V_{\mathrm{Min}}$ such that $\pi_i(v) \neq \pi_{i-1}(v)$, it holds that $d_{i-1}(v) > d_{i-1}(\pi_i(v)) + w(v, \pi_i(v))$. It will be important later that another iteration of the main while-loop on lines 5–19 is started only if there is some vertex with the above property.

It is obvious that the vertices for which Min's strategy was improved do not violate property (ii.) from the assumptions of Theorem 3.1. To see that property (i.) is also not violated consider a newly formed cycle $c = (v_0, \ldots, v_{k-1}, v_0) \in \mathsf{cycle}^{G_{\pi_i}(D' \setminus A')}$. For each $j \in \{0, \ldots, k-1\}$, it holds that $d_{i-1}(v_j) \geq d_{i-1}(v_{j+1}) + w(v_j, v_{j+1})$, and since the cycle was not in $G_{\pi_{i-1}}(D' \setminus A')$, for at least one j, the inequality is strict. Therefore, by summing the inequalities, we get $0 > w(c)$ and the property (i.) from the assumptions of Theorem 3.1 holds. Together, we have that $\Gamma$, b, $\pi_i$, and $d_{i-1}$ satisfy the assumptions of Theorem 3.1. Thus, the first part of the lemma is proved. Since the assumptions of Theorem 3.1 (and also Lemma 6.1) are satisfied, we can use Lemma 6.1 to prove the second part of the lemma.

As was already mentioned there is a vertex $v \in V$, such that there is $(v, u) \in E_{\pi_i}$ such that $d_{i-1}(v) > d_{i-1}(u) + w(v, u)$. We can take any vertex at which $\pi_i$ differs from $\pi_{i-1}$ as $v$, and $\pi_i$ differs from $\pi_{i-1}$, because otherwise the algorithm would have already terminated. Therefore, by repeated application on Lemma 6.1, part 2, it holds that the vector $d_i$ returned by EVALUATESTRATEGY() satisfies $d_i < d_{i-1}$. This completes the proof of the lemma. ■

**Proof:** [Proof of Theorem 3.5] In the whole proof, all references to lines in pseudocode are references to lines in pseudocode of LOWERWEAKUPPERBOUND() in Figure 2. By Lemma 3.4, the procedure LOWERWEAKUPPERBOUND($\Gamma$, b) terminates and gives us the vector ds. We will show that there is a positional strategy of Max $\sigma \in \Sigma_M^\Gamma$ and there is a strategy of Min $\pi \in \Pi^\Gamma$ such that the following holds for each $v \in V$.

$$\begin{aligned} ds(v) \geq \min\{x \in \mathbb{N}_0 \mid \quad & (\forall \pi' \in \Pi^\Gamma) \\ & (\ \mathsf{outcome}^\Gamma(v, \sigma, \pi') = (v = v_0, v_1, v_2, \ldots) \wedge \\ & \quad (\forall n \in \mathbb{N})(x + \textstyle\sum_{j=0}^{n-1} w(v_j, v_{j+1}) \geq 0) \wedge \\ & \quad (\forall n_1, n_2 \in \mathbb{N}_0)(n_1 < n_2 \Rightarrow \textstyle\sum_{j=n_1}^{n_2-1} w(v_j, v_{j+1}) \geq -b)\ )\ \} \end{aligned} \tag{1}$$

$$\begin{aligned} ds(v) \leq \min\{x \in \mathbb{N}_0 \mid \quad & (\exists \sigma' \in \Sigma^{\Gamma}) \\ & (\quad \mathsf{outcome}^{\Gamma}(v, \sigma', \pi) = (v = v_0, v_1, v_2, \ldots) \land \\ & \quad (\forall n \in \mathbb{N})(x + \textstyle\sum_{j=0}^{n-1} w(v_j, v_{j+1}) \geq 0) \land \\ & \quad (\forall n_1, n_2 \in \mathbb{N}_0)(n_1 < n_2 \Rightarrow \textstyle\sum_{j=n_1}^{n_2-1} w(v_j, v_{j+1}) \geq -b) \;)\;\} \end{aligned} \tag{2}$$

The inequality (1) says that if Max uses the strategy $\sigma$, then $ds(v)$ is a sufficient amount of initial energy for plays starting from $v$. The inequality (2) says that if Min uses the strategy $\pi$, then Max needs at least $ds(v)$ units of initial energy. By putting (1) and (2) together, we get that $ds(v) = \mathsf{lwub}_b^{\Gamma}(v)$. Let's first find the strategy $\sigma$.

Let's consider the situation just after termination of the main while-loop. In this situation it holds that $ds = -d_{i-1}$. Let $D_j = \{v \in V \mid d_j(v) > -\infty\}$, for each $j \in \{-1, 0, \ldots, i-1\}$. Please not that, by Lemma 3.3,

$$D_{-1} \supseteq D_0 \supseteq \cdots \supseteq D_{i-1}$$

Let further $B = \{v \in V \mid d_{i-1}(v) = 0\}$ and let $\sigma \in \Sigma_M^{\Gamma}$ be the following strategy of Max. For $v \in V_{\mathrm{Max}} \cap (V \setminus D_{i-1})$, let $\sigma(v)$ be an arbitrary successor of $v$. For $v \in (V_{\mathrm{Max}} \setminus B) \cap D_{i-1}$, let $\sigma(v) = u$ such that $(v, u) \in E$ and $d_{i-1}(v) = w(v, u) + d_{i-1}(u)$. Such a vertex $u$ exists by Lemma 6.1, part 1. And finally, for $v \in B \cap V_{\mathrm{Max}}$, let $\sigma(v) = u$ such that $(v, u) \in E$ and $w(v, u) - d_{i-1}(v) + d_{i-1}(u) \geq 0$. Such a vertex $u$ exists by Lemma 6.2.

Since the main while-loop on lines 5–19 terminated, there is no vertex in $V_{\mathrm{Min}} \cap D_{i-1}$ that satisfies the strategy improvement condition, hence for each $v \in V_{\mathrm{Min}} \cap D_{i-1}$, it holds that for each $(v, u) \in E$, $w(v, u) - d_{i-1}(v) + d_{i-1}(u) \geq 0$. This implies that in the graph $G_{\sigma}$, there is no edge from $D_{i-1}$ (vertices with finite $d_{i-1}$ value) to $V \setminus D_{i-1}$ (vertices with infinite $d_{i-1}$ value). Therefore, for each $v \in D_{i-1}$ and $\pi' \in \Pi^{\Gamma}$, $\mathsf{outcome}^{\Gamma}(v, \sigma, \pi') = (v = v_0, v_1, v_2, \ldots)$ satisfies the following. Let's denote the outcome by $p$.

$$(\forall j \in \mathbb{N}_0)(d_{i-1}(v_j) \leq d_{i-1}(v_{j+1}) + w(v_j, v_{j+1}))$$

It follows that for each prefix $p_k = (v = v_0, \ldots, v_k)$ of $p$, it holds that $d_{i-1}(v) \leq d_{i-1}(v_k) + w(p_k)$, and since by Lemma 6.1, part 3 and part 4, $d_{i-1}(v_k) \leq 0$, we have $d_{i-1}(v) \leq w(p_k)$. To prove (1), it remains to show that $p$ does not contain a segment of weight less than $-b$.

Let $n_1, n_2 \in \mathbb{N}_0$ such that $n_1 < n_2$. By the same reasoning as in the previous paragraph, we get that $d_{i-1}(v_{n_1}) \leq d_{i-1}(v_{n_2}) + \sum_{j=n_1}^{n_2-1} w(v_j, v_{j+1})$, and so $d_{i-1}(v_{n_1}) -$

$d_{i-1}(v_{n_2}) \leq \sum_{j=n_1}^{n_2-1} w(v_j, v_{j+1})$. By Lemma 6.1, part 1, 3, and 4, it holds that $-b \leq d_{i-1}(v_{n_1}), d_{i-1}(v_{n_2}) \leq 0$. Therefore, $d_{i-1}(v_{n_1}) - d_{i-1}(v_{n_2}) \geq -b$, and the inequality (1) is proved. Let's now find the strategy $\pi$.

For each path $p = (v_0, \ldots, v_k) \in \mathsf{path}^G$ such that $v_k \in V_{Min}$, $\pi(p) = \pi_j(v_k)$, where $j = \min(i-1, \min\{x \in \{0, \ldots, i-1\} \mid (\exists y \in \{0, \ldots, k\})(d_x(v_y) = -\infty)\})$. That is, the strategy $\pi$ makes the same choice as the first positional strategy from the sequence $(\pi_0, \ldots, \pi_{i-1})$ that is responsible for making one of the vertices in $p$ losing for Max. If there is no vertex in $p$ with infinite $d_{i-1}$ value, then $\pi$ makes the same choice as the strategy $\pi_{i-1}$.

Please note that, by Theorem 3.1, $ds = \mathsf{lwub}_b^{\Gamma_{\pi_{i-1}}(D_{i-2})}$. That is, if Min uses the strategy $\pi$, the play starts from $v \in D_{i-2}$, and stays in $D_{i-2}$ (and so $\pi$ follows the strategy $\pi_{i-1}$), then Max needs at least $ds(v)$ units of initial energy. Therefore, if we show that each play that leaves $D_{i-2}$ is losing for Max, then the inequality (2), and hence the theorem, will be proved.

Let $v \in V$ and $\sigma' \in \Sigma^\Gamma$ such that the outcome $\mathsf{outcome}^\Gamma(v, \sigma, \pi') = (v = v_0, v_1, v_2, \ldots)$ contains a vertex from $V \setminus D_{i-2}$. Let's denote the outcome by $p$ and let's prove that it is losing for Max.

Let $k = \min\{x \in \{0, \ldots, i-1\} \mid (\exists y \in \mathbb{N}_0)(d_x(v_y) = -\infty)\}$. There is some vertex from $V \setminus D_{i-2}$ in $p$, and so we do not have to consider the case when the minimum is equal to $\infty$. It follows that $k < i-1$. From the definition of $k$, it also follows that the play never leaves $D_{k-1}$, and there is vertex $v_\infty$ from $D_{k-1} \setminus D_k$ in the path $p$. From the vertex $v_\infty$ onwards, Min uses the strategy $\pi_k$. It holds that $d_k(v_\infty) = -\infty$, and so, by Theorem 3.1, $\mathsf{lwub}_b^{\Gamma_{\pi_k}(D_{k-1})}(v_\infty) = \infty$. Therefore, the suffix of $p$ starting from $v_\infty$ is losing for Max, and so $p$ is losing for Max too. We have that if Min uses $\pi$ and the play leaves $D_{i-2}$ then Max loses. This completes the proof of the theorem. ■

## 6.3 Improvement of the Complexity of KASI

This section describes a way to improve the complexity of KASI from $O(|V|^2 \cdot (|V| \cdot \log|V| + |E|) \cdot W)$ to $O(|V| \cdot (|V| \cdot \log|V| + |E|) \cdot W)$. The pseudocode of the improved algorithm is in Figures 5–9. All these figures share the following global variables: vectors $d, d', key \in (\mathbb{Z} \cup \{-\infty\})^V$, vector $queued \in \{true, false\}^V$, vectors $count_M, count_B \in \mathbb{N}_0^V$, sets $D, B \subseteq V$, and strategy $\pi \in \Pi_M^\Gamma$.

The improved algorithm KASI (ImpKASI for short) performs the same basic steps as the original algorithm KASI, but the steps are implemented more efficiently. Instead of filling all the elements of vector $d_i$ in each iteration of both EVALUATESTRATEGY() and

LowerWeakUpperBound(), ImpKASI uses one global vector $d$ and updates only the elements that need to be updated. The set of vertices for which the $d$ value needs to be updated is computed efficiently by the procedure ImpMarking() in Figure 7. This way, in each iteration, the algorithm works only with vertices whose $d$ value is updated, which improves the overall complexity by a factor of $|V|$. We will now describe how the individual parts of ImpKASI work.

During the execution of ImpKASI, $B$ is an over-approximation of the set of vertices with zero $\mathsf{lwub}_b^{\Gamma}$, and the vector $d$ is such that $-d$ is a lower estimate of $\mathsf{lwub}_b^{\Gamma}$ ($-d \leq \mathsf{lwub}_b^{\Gamma}$). The set $D$ is a set of vertices such that about the vertices in $V \setminus D$ we already know that they have infinite $\mathsf{lwub}_b^{\Gamma}$, hence

$$D = \{v \in V \mid d(v) > -\infty\}$$

To be able to perform certain operations efficiently, we keep the vectors $\mathtt{count}_M$ and $\mathtt{cound}_B$ such that

$$(\forall v \in D \setminus B)(\mathtt{count}_M(v) = |\{(v,u) \in E_\pi \mid d(v) = d(u) + w(v,u)\}|)$$

$$(\forall v \in B)(\mathtt{count}_B(v) = |\{(v,u) \in E_\pi \mid w(v,u) - d(v) + d(u) \geq 0\}|)$$

The vector $d$ contains weights of longest paths from $D$ to $B$, and so if for some $v \in D \setminus B$, $\mathtt{count}_M(v)$ drops to zero, its $d$ value has to be updated. Similarly, if for some $v \in B$, $\mathtt{count}_B(v)$ drops to zero, $v$ has to be removed from $B$.

In Figures 6 and 5 is the pseudocode of the modified Dijkstra's algorithm used in ImpKASI. The input to ImpDijkstra() consists of three parts. The first part is a graph $G_\pi$, the second part is a bound $b$, and finally the third part is a set $M \subseteq V$ such that $M$ contains exactly the vertices the $d$ value of which need to be updated. ImpDijkstra() has no specific output, it just modifies global variables, in particular, the vector $d$.

For each vertex $v \in D \setminus M$, $d(v)$ is already the correct weight of longest path from $v$ to $B$. So, the job of ImpDijkstra() is to assign the correct weights also to the vertices in $M$.

The Dijkstra's algorithm is initialized by the procedure InitializeDijkstra() in Figure 5. For each $v \in M$, the procedure scans all outgoing edges $(v,u) \in E_\pi$ such that $u \in D \wedge u \notin M$, i.e., $u$ is already assigned the correct longest path weight, and finds out which of the edges yields the smallest decrease of $d(v)$. Decreases below $-b$ are ignored.

The smallest decrease is stored in $key(v)$ and the count of edges that yield the smallest decrease is stored in $count_M(v)$. The vertices that have outgoing edges which satisfy the constraints are put to the maximum priority queue $q$, where priority of a vertex $v$ is $key(v)$. The queue is then used in the main procedure of the Dijkstra's algorithm in Figure 6.

The main while-loop of IMPDIJKSTRA() on lines 3–19 is the same as in the not-improved DIJKSTRA(), with the only modification that the vector $count_M$ is updated as necessary. After the main loop terminates, the vector $d$ and the set $D$ are updated on lines 20–23.

Let's now describe how the procedure IMPMARKING() computes the set of vertices whose $d$ value need to be updated. The vector $d$ contains longest path weights from $D$ to $B$ in $G_\pi(D)$ and the updates of $d$ are induced either by removal of vertices from $B$ or by improvements of the strategy $\pi$. In both cases IMPMARKING() works the same.

The input to the procedure consists of two parts. The first part is a graph $G_\pi$ and the second part is a set $X \subseteq V$ which contains the vertices about which we already know that their $d$ value needs to be updated. These are either the vertices removed from $B$ or the vertices at which the strategy $\pi$ was improved. The procedure then computes the smallest set $M$ such that $X \subseteq M \subseteq D \setminus B$ and for each $v \in (D \setminus B) \setminus M$, there is an edge $(v, u) \in E_\pi$ such that $u \notin M$ and $d(v) = d(u) + w(v, u)$.

IMPMARKING() starts from the set $X$ and then explores the graph $G_\pi$ in the reversed direction edges. It iteratively repeats the following steps. It picks a vertex $u$ from the working set $Z$ and explores all of its predecessors. For each predecessor $v$ it checks whether $v$ is not already in $M$, $v$ is in $D$, and $v$ is not in $B$. If $v \in M$, then we already know that $d(v)$ needs to be updated. If $v \notin D$, then $d(v) = -\infty$, and so no update of $d(v)$ is possible. Finally, if $v \in B$, then $d(v) = 0$, which is the correct longest path weight from $v$ to $B$. So, if $v$ satisfies all the conditions, we decide whether to add it to $M$ or not.

To that end, we check whether there is some successor $(v, u') \in E_\pi$ such that $u' \notin M$ and $d(v) = d(u') + w(v, u')$. For the complexity of ImpKASI, it is important that only predecessors/successors of vertices whose $d$ value needs to be updated are explored. Therefore, we do not explore all the successors of $v$ to find out if there is a successor $u' \notin M$ such that $d(v) = d(u') + w(v, u')$. We use the vector $count_M$ instead. If $u$, the successor from which we got to $v$, contributes to $count_M(v)$ ($d(v) = d(u) + w(v, u)$), then we decrease $count_M(v)$, because we already know that $d(u)$ needs to be updated, and so $u$ will no longer contribute to $count_M(v)$. If $count_M(v)$ drops to zero, then we know that $d(v)$ needs to be updated too, hence we add $v$ to the set $M$ and also to the working set $Z$. After the main loop on lines 4–18 terminates, the set $M$ is returned as output.

If Figure 8 is the pseudocode of the strategy evaluation procedure of ImpKASI. The input to ImpEvaluateStrategy() consist of four parts. The first and the second part are an MPG $\Gamma$ and a bound $b$. The third part is a strategy $\pi$ that we want to evaluate, and finally the fourth part is a set $Z \subseteq V$ which contains the vertices at which the strategy $\pi$ was improved since the last strategy evaluation. The set $Z$ is given as a part of input to ImpMarking() in the first iteration of the strategy evaluation process. The output of ImpEvaluateStrategy() is the set of vertices whose $d$ value was updated during the whole strategy evaluation process.

ImpEvaluateStrategy() iteratively repeats the following steps. On line 5, it uses ImpMarking() to compute the vertices whose $d$ value need to be updated and returns them as the set $M$. On line 6, it uses ImpDijkstra() to update the vector $d$. The vertices whose $d$ value was updated are added to the set $M_\pi$. The purpose of lines 9–21 is to remove from $B$ the vertices $v$ that does not have an outgoing edge $(v, u)$ such that $w(v, u) - d(v) + d(u) \geq 0$. For complexity reasons, this operation avoids exploration of successors/predecessors of vertices outside the set $M$. This is made possible by the vector $count_B$.

It holds that $M \cap B = \emptyset$, so the vertices from $B$ that we need to remove must have a successor in $M$. Therefore, for each vertex $u \in M$, we explore all edges $(v, u) \in E_\pi$ such that $v \in B$. If an edge $(v, u)$ satisfies $w(v, u) - d'(v) + d'(u) \geq 0 \wedge w(v, u) - d(v) + d(u) < 0$, where $d'$ is the state of the vector $d$ before it was updated, then it means that the edge $(v, u)$ was one of the reasons why $v$ is in the set $B$, but it no longer is a reason for that. And so, we decrease $count_B(v)$, and if $count_B(v)$ drops to zero, we remove $v$ from $B$ and add it to $X$. The set $X$ is a basis for the set of vertices whose $d$ value will be updated in the next iteration. After the updating of the set $B$ is finished, we copy the updated part of the vector $d$ to the vector $d'$, so that $d' = d$, and start another iteration of the main while-loop. If no vertex is removed from $B$, and so no vertex is added to $X$, then the main loop terminates and ImpEvaluateStrategy() returns the set $M_\pi$.

In Figure 9 is the main procedure of ImpKASI. The initialization phase of the procedure is on lines 2–11. As the initial strategy $\pi$ of Min we take an arbitrary strategy from $\Pi_M^\Gamma$. The vector $d$ is initialized to vector of zeros and so is the vector $d'$. The initialization of the set $B$ is on lines 5–9. For each $v \in V$, the algorithm first counts the number of non-negative edges emanation from $v$ (since $d$ is a vector of zeros, $w(v,u) \geq 0$ also means that $w(v,u) - d(v) + d(u) \geq 0$), assigns the count to $\mathtt{count}_B(v)$, and if $\mathtt{count}_B(v) > 0$, adds $v$ to $B$. The set $D$ is initially equal to $V$, because all vertices have finite $d$ value. Finally, since all vertices from $D \setminus B$ ($= V \setminus B$) have only negative edges emanating from them, the set of vertices whose $d$ value need to be updated in the first iteration of the main while-loop on lines 12–29 is the largest possible, namely, it is equal to $D \setminus B$. Therefore, the set of vertices $Z$ that will be given as a part of input to the procedure IMPEVALUATESTRATEGY() is initialized to $D \setminus B$.

Each iteration of the main loop does the following steps. On line 13, it evaluates the current strategy $\pi$, which leads to updates to the vector $d$. The strategy evaluation procedure returns the set $M_\pi$ of vertices whose $d$ value was updated. The purpose of the rest of the iteration is to improve the strategy $\pi$.

For complexity reasons, the strategy improvement phase has to explore only successors/predecessors of vertices whose $d$ value was updated by IMPEVALUATESTRATEGY(). However, this does not complicate matters much. The strategy improvement condition can be satisfied only at vertices with some successor in $M_\pi$. The reason is the following. If $d(v) > d(u) + w(v,u)$ was not satisfied in the previous iteration, then it can be satisfied in the current iteration only if $d(u)$ was updated (Recall that all updates to the vector $d$ are decreases). Therefore, for each vertex $u \in M_\pi$, we check each edge $(v,u) \in E$ such that $v \in D \cap V_{\text{Min}}$ to see whether it satisfies the strategy improvement condition, and if it does, then we take appropriate steps. However, there is one important difference from the original not-improved version of KASI.

We cannot use arbitrary edge that satisfies the strategy improvement condition for the strategy improvement. For each $v \in V_{\text{Min}}$ such that edges satisfying the strategy improvement condition emanate from $v$, we must always take the best possible improvement, because if the $d$ value of the successor vertex that yields the best improvement is not updated again, the improvement may be omitted. We use the vector $d'$ to find out which successors yield the best improvement.

Before the strategy improvement phase, it holds that $d' = d$. During the strategy improvement phase, we record in $d'$ the best possible improvements. If $d'(v) > d(u) + w(v,u)$, it means that $u$ yields better improvement than the ones found so far, and so $d'(v)$ is updated to $d(u) + w(v,u)$, $\pi(v)$ is set to $u$, and finally $v$ is added to $Z$ and removed from $\mathrm{B}$. The removal from $\mathrm{B}$ is caused by the fact that $(v, \pi(v))$ is the only edge emanating from $v$ in $G_\pi$, and if $d(v) > d(\pi(v)) + w(v, \pi(v))$, then $v$ cannot be in $\mathrm{B}$.

After the strategy improvement phase ends, we have to undo the updates made to $d'$, because to function properly, ImpEvaluateStrategy() requires that $d = d'$. This is done on line 28. If some improvements were made, which means that $Z \neq \emptyset$, another iteration of the main loop is started, otherwise $-d = \mathsf{lwub}_b^\Gamma$, and so the procedure terminates and returns $-d$. The complexity analysis follows.

The algorithm works only with vertices which update their $d$ value, in each iteration, both in ImpEvaluateStrategy() and ImpLowerWeakUpperBound(). Each vertex can be improved at most $O(|V| \cdot W)$ times and each improvement of a vertex $v \in V$ costs us $O(\log|V| + \mathit{indegree}(v) + \mathit{outdegree}(v))$ time. The complexity $O(\log|V| + \mathit{indegree}(v) + \mathit{outdegree}(v))$ comes from the fact that the successors/predecessors of each vertex with updated $d$ value are explored only constant number of times, and the operations needed for maintenance of the maximal priority queue used in ImpDijkstra() cost $O(\log|V|)$ time per vertex. Therefore, the complexity of the algorithm is $O(|V| \cdot W \cdot \sum_{v \in V}(\log|V| + \mathit{indegree}(v) + \mathit{outdegree}(v))) = O(|V| \cdot (|V| \cdot \log|V| + |E|) \cdot W)$.

```
proc InitializeImpDijkstra($G_\pi = (V, E_\pi, w), b, M$)
  [$q$ is a maximum priority queue of vertices, where the priority of vertex $v$ is $key(v)$]
  foreach $v \in M$ do
    $queued(v) := false$
    $key(v) = -\infty$
    foreach $(v, u) \in E_\pi$ do
      if $u \in D \ \wedge\ u \notin M$ then
        $tmp := w(v, u) - d(v) + d(u)$
        if $d(v) + tmp \geq -b$ then
          if $tmp > key(v)$ then
            $key(v) := tmp$
            $count_M(v) := 1$
          elsif $tmp = key(v)$ then
            $count_M(v) := count_M(v) + 1$
          fi
        fi
      od
      if $key(v) \neq -\infty$ then
        $q.enqueue(v)$
        $queued(v) := true$
      fi
    od
  od
  return $q$
end
```

Figure 5: Initialization of Dijkstra

```
proc ImpDijkstra($G_\pi = (V, E_\pi, w), b, M$)
  InitializeImpDijkstra($G_\pi, b, M$)
  while $\neg q.empty()$ do
    $u := q.dequeue()$
    foreach $(v, u) \in E_\pi$ do
      if $v \in M$ then
        $tmp := key(u) + w(v, u) - d(v) + d(u)$
        if $d(v) + tmp \geq -b$ then
          if $tmp > key(v)$ then
            $key(v) := tmp$
            $count_M(v) := 1$
            if $\neg queued(v)$ then $q.enqueue(v);\ queued(v) := true$ fi
          elsif $tmp = key(v)$ then
            $count_M(v) := count_M(v) + 1$
          fi
        fi
      fi
    od
  od
  foreach $v \in M$ do
    $d(v) := d(v) + key(v)$
    if $d(v) = -\infty$ then $D := D \setminus \{v\}$ fi
  od
end
```

Figure 6: Modified Dijkstra's algorithm

```
proc ImpMarking($G_\pi = (V, E_\pi, w), X$)
  $M := X$
  $Z := X$
  while $Z \neq \emptyset$ do
    pick $u \in Z$
    $Z := Z \setminus \{u\}$
    foreach $(v, u) \in E_\pi$ do
      if $v \notin M \wedge v \in D \wedge v \notin B$ then
        if $d(v) = d(u) + w(v, u)$ then
          $\mathrm{count}_M(v) := \mathrm{count}_M(v) - 1$
          if $\mathrm{count}_M(v) = 0$ then
            $Z := Z \cup \{v\}$
            $M := M \cup \{v\}$
          fi
        fi
      fi
    od
  od
  return $M$
end
```

Figure 7: Marking of vertices for which d has to be improved

```
proc ImpEvaluateStrategy($\Gamma, b, \pi, Z$)
  $X := Z$
  $M_\pi := \emptyset$
  while $X \neq \emptyset$ do
    $M :=$ ImpMarking($G_\pi, X$)
    ImpDijkstra($G_\pi, b, M$)
    $M_\pi := M_\pi \cup M$
    $X := \emptyset$
    foreach $u \in M$ do
      foreach $(v, u) \in E_\pi$ do
        if $v \in B$ then
          if $w(v,u) - d'(v) + d'(u) \geq 0 \land w(v,u) - d(v) + d(u) < 0$ then
            $count_B(v) := count_B(v) - 1$
            if $count_B(v) = 0$ then
              $B := B \setminus \{v\}$
              $X := X \cup \{v\}$
            fi
          fi
        fi
      od
    od
    foreach $v \in M$ do $d'(v) := d(v)$ od
  od
  return $M_\pi$
end
```

Figure 8: Evaluation of strategy

```
proc ImpLowerWeakUpperBound($\Gamma, b$)
  $\pi :=$ Arbitrary strategy from $\Pi_M^\Gamma$
  $d := (0)_{v \in V}$
  $d' := d$
  $B := \emptyset$
  foreach $v \in V$ do
    $count_B(v) :=| \{(v,u) \in E_\pi \mid w(v,u) \geq 0\} |$
    if $count_B(v) > 0$ then $B := B \cup \{v\}$ fi
  od
  $D := V$
  $Z := D \setminus B$
  while $Z \neq \emptyset$ do
    $M_\pi :=$ ImpEvaluateStrategy($\Gamma, b, \pi, Z$)
    $Z := \emptyset$
    foreach $u \in M_\pi$ do
      foreach $(v,u) \in E$ do
        if $v \in V_{Min} \wedge v \in D$ then
          if $d'(v) > d(u) + w(v,u)$ then
            $d'(v) := d(u) + w(v,u)$
            $\pi(v) := u$
            $Z := Z \cup \{v\}$
            $count_B(v) := 0$
            $B := B \setminus \{v\}$
          fi
        fi
      od
    od
    foreach $v \in Z$ do $d'(v) := d(v)$ od
  od
  return $-d$
end
```

Figure 9: Solving the lower-weak-upper-bound problem