ON THE COMPLEXITY OF THE POLICY ITERATION ALGORITHM

by

Mary Melekopoglou and Anne Condon

Computer Sciences Technical Report #941

June 1990

On the Complexity of the Policy Iteration Algorithm for Stochastic Games

Mary Melekopoglou
Anne Condon[1]

Computer Sciences Department
University of Wisconsin - Madison
1210 West Dayton Street
Madison, WI 53706

ABSTRACT

We consider a natural class of algorithms for simple stochastic games. It has been proved that the problem of deciding which player has the greatest chance of winning the game is in the class NP $\cap$ co-NP. It is not known whether the problem is in P. We examine a number of local search algorithms, called policy iteration algorithms, which solve this problem, and prove that these algorithms require exponential time in the worst case.

1. Introduction

Simple stochastic games model simple games of chance played by two players (these games are described in Section 1.1). The problem of deciding which player has the greatest chance of winning such a game has been proved to be in the class NP $\cap$ co-NP, but it is an open problem if it is in P. A natural approach to solve the problem is the policy iteration algorithm (described in Section 1.3). In this paper, we prove that the algorithm takes exponential time in the worst case. We also describe several ways in which the algorithm can be improved, and for all of them we construct counterexamples that require exponential time.

1.1. Description of the Simple Stochastic Game (SSG) problem

The model of stochastic games was introduced by Shapley in 1953, [11]. We describe a simplified version of Shapley's original model here. A **simple stochastic game (SSG)** is a directed graph $G = (V, E)$ with the following properties. The vertex set V is the union of disjoint sets $V_{\max}$, $V_{\min}$, V_{average}, called **max, min** and **average** vertices, together with two special vertices, called the **0-sink** and the **1-**

1 Supported by National Science Foundation grant number DCR-8802736.

sink. One vertex of V is called the **start** vertex. Each vertex of V has two outgoing edges (not necessary distinct), except the sink vertices, which have no outgoing edges.

The graph models a game between two players, 0 and 1. In the game a token is initially placed on the start vertex, and at each step of the game the token is moved from a vertex to one of its neighbors, according to the following rules: At a min (max) vertex, player 0 (1) chooses the neighbor to which the token is moved. At an average vertex, the token is moved to each of its neighbors with probability 1/2.

The game ends when the token reaches a sink vertex; player 1 wins if it reaches the 1-sink vertex and player 0 wins otherwise, that is, if the token reaches the 0-sink vertex or if the game never halts. The reason for the names max and min vertices is that at the max (min) vertices, player 1 (0) chooses its move so as to maximize (minimize) the probability of eventually reaching the 1-sink vertex.

A **strategy** for player 0 is (defined to be) a rule that defines what move the player takes whenever the token is at a min vertex; formally a strategy for player 0 is a set of edges of E, each originating from a min vertex, such that for each min vertex there is exactly one edge in the strategy originating from it. The definition of a strategy for player 1 is similar (min should be replaced with max). Note that with this definition, a move from a max or min vertex is not probabilistic and is the same every time that vertex is reached. For this reason, the strategies are called pure stationary strategies.

We say a SSG **halts with probability 1** if for all pairs of strategies of the players, the game ends (that is, the token reaches a sink vertex) with probability 1.

The **value** of a vertex i, $V_{S_0S_1}(i)$, with respect to a strategy S_0 for player 0 and a strategy S_1 for player 1 is (defined to be) the probability that player 1 wins the game, if the start vertex is i and the players use strategies S_0 and S_1. The values of all the vertices of the graph, with respect to a pair of strategies, can be found in the following way. The value of the 0-sink is 0, the value of the 1-sink is 1, and the value of any vertex from which there is no path to a sink is 0. For any other vertex, we can write an equation that gives its value as follows: the value of an average vertex is the average of the values of its two children, the value of a min (max) vertex is equal to the value of the child that player 0 (1) moves to according to S_0 (S_1). The set of these linear equations can be solved in polynomial time and it has a unique solution (proof in [1]).

The **value of the game** is (defined to be) $\max_{S_1} \min_{S_0} V_{S_0S_1}(start)$, or the probability that player 1 wins if it reveals its best strategy, S_1, to player 0 at the start of the game, and player 0 plays its best strategy, S_0, against S_1. The **SSG value problem** is: Given an SSG, is its value $> 1/2$? A related problem is to find the best strategies of the players.

1.2. Previous Work

The properties of stochastic games have been studied by a number of researchers since the introduction of the model by Shapley (see Peters and Vrieze, [10], or Condon, [1], for a survey of this work). Proofs of all of the results in this section can be found in [1].

Shapley showed that for any SSG, there is a pair of strategies $S_{0,OPT}$, $S_{1,OPT}$, such that for all i,

$$V_{S_{0,OPT}S_{1,OPT}}(i) = \max_{S_1} \min_{S_0} V_{S_0S_1}(i) \text{ (which in fact, equals } \min_{S_0} \max_{S_1} V_{S_0S_1}(i)).$$

A pair of strategies satisfying this property is called an **optimal** pair of strategies.

Shapley's results also imply the following important property of SSG's. Suppose we define a pair of strategies to be **locally optimal** if for all i, if i is a max vertex, the value of vertex i is the maximum of the values of its children, and, if i is is a min vertex, the value of vertex i is the minimum of the values of its children. Then for SSG's that halt with probability 1, any locally optimal pair of strategies is an optimal pair of strategies. Condon, [1], showed that the SSG value problem can be reduced in polynomial time to the SSG value problem for SSG's that halt with probability 1; hence without loss of generality we only consider SSG's that halt with probability 1 here. Thus, to solve the SSG value problem, it is sufficient to find a locally optimal pair of strategies.

The SSG value problem has many interesting complexity-theoretic properties. It is a rare example of a combinatorial problem in the class NP $\cap$ co-NP, but not known to be in P, [1]. The linear programming problem was previously an example of such a problem, but with the discovery of the ellipsoid algorithm (Khachiyan, [7]), this problem was shown to be in P. Also, if the instances of the SSG problem are restricted to have only two types of vertices (max and min, max and average or min and average), the problem can be solved in polynomial time. The proof of this for max and average, or min and average, vertices is via a reduction to the linear programming problem (Derman, [2]).

There are other similarities between the linear programming problem and the SSG value problem. In particular, there is a local search algorithm that solves each problem - the simplex algorithm in the case of linear programming and the policy iteration algorithm for the SSG value problem (where instances of the problem are SSG's that halt with probability 1.) Roughly, the **policy iteration algorithm** initially chooses an arbitrary pair of strategies, and repeatedly selects a better pair of strategies until a locally optimal pair of strategies is found. It was first described for graphs with max and average vertices by Howard, [4] and later generalized to graphs with max, min and average vertices, [10]. (The name of the algorithm is derived from the fact that in some of the literature, strategies are called policies). However, many variations of the simplex algorithm have been proved to require exponential time in the worst case (Klee and Minty, [8], Jeroslow, [5]). In this paper, we show that the policy iteration algorithm also requires exponential time in the worst case.

1.3. The Policy Iteration Algorithm

In this section, we describe the policy iteration algorithm in detail, and introduce some notation used in the paper. In the remaining sections, we prove that the policy iteration algorithm requires exponential time in the worst case.

In order to describe the algorithm, we need the following definition. We say a min (max) vertex is **switchable** with respect to a fixed pair of strategies, if its value is not the minimum (maximum) of the values of its children. Note that it can be determined in polynomial time if a vertex is switchable, since the values of the vertices with respect to any pair of strategies can be computed in polynomial time.

We first describe the policy iteration algorithm in the case that the graph has only min and average vertices, and then briefly outline how it can be generalized to handle max vertices as well. The algorithm starts from an initial strategy for player 0. Then it repeatedly selects a switchable min vertex, and **switches** it. That is, it changes the strategy of player 0, so that the edge corresponding to the selected vertex is replaced by the other edge of the vertex. The algorithm halts when there is no switchable vertex, in which case by definition, a locally optimal strategy has been found for player 0.

More than one vertex may be switchable at any iteration. Thus, to completely specify the algorithm, a **select procedure** must be defined, which returns the next vertex to be switched, if any. We describe a number of different versions of the select procedure later in this section.

The proof that this algorithm works is based on the fact that after any iteration, the value of the switched vertex has decreased, and the value of no vertex has increased. Thus, the total sum of the values of all the vertices decreases. Because of this, no strategy is reached twice during the execution of the algorithm. Since there are at most an exponential number of strategies, the algorithm must halt within exponential time. (See [1] or [4] for details of the proof.)

The policy iteration algorithm can be extended to SSG's with max, min and average vertices as follows. The algorithm starts from an initial strategy for each player. Then the following two steps are repeated, until a locally optimal pair of strategies is found: (i) Keeping the strategy for player 1 fixed, switchable min vertices are repeatedly switched as before until there is no switchable min vertex. (ii) A switchable max vertex is switched.

In the rest of the paper, we only consider SSG's with just min and average vertices, since we can prove that even with this restriction, the policy iteration algorithm requires exponential time. Our results also apply immediately to SSG's with max, min and average vertices. The performance of the algorithm is measured by the number of switches because all the versions of the select procedure that we describe are polynomial time.

We consider a number of different versions of the policy iteration algorithm, based on the way the select procedure chooses a switchable vertex. In the simplest version, we assume that the select procedure makes an arbitrary decision, by choosing the highest numbered switchable min vertex. We call this version the **Simple Policy Iteration Algorithm**, and in Section 2 we show that this algorithm requires exponential time in the worst case. To do this, we show how to construct, for any n, a graph which has a number of vertices polynomial in n but for which the number of switches needed by the policy iteration algorithm is exponential in n.

In the following sections, we examine some natural improvements to the select procedure. In our first attempt (**Topological Policy Iteration Algorithm**), the select procedure gives priority to vertices on which the values of many other min vertices depend. Another idea is to select the switchable vertex with the largest difference between the values of its neighbors (**Difference Policy Iteration Algorithm**). A still better algorithm would be to select the switchable vertex whose value will be most improved (decreased) if switched (**Improvement Policy Iteration Algorithm**). We build on our proof for the simple policy iteration algorithm, to show that all of these algorithms require exponential time in the worst case.

We use the following notations in the construction of exponential time graphs for the policy iteration algorithms. Min vertices are represented with circles numbered 1, 2 , ..., average vertices with circles numbered $0'$, $1'$, ..., and the sink vertices with values 0 and 1, with squares. The two edges for every min vertex are labeled with 0 and 1; so a strategy for a graph with n min vertices can be represented as an n-bit vector, $S = S_n S_{n-1} \cdots S_1$, where S_i is the label of the edge in S that originates from i. For simplicity, when describing the execution of a policy iteration algorithm, we denote by $V(i)$ the value of vertex i with respect to the current strategy for player 0.

2. Simple Policy Iteration Algorithm

The select procedure that the Simple Policy Iteration Algorithm uses selects arbitrarily one of the switchable min vertices. Specifically, it selects the switchable vertex with the largest number. The algorithm is simple in the sense that the select procedure does not select the vertex to be switched based on either the values of the vertices or the structure of the graph.

In the rest of the section we present a counterexample for the algorithm; a graph on which we prove that the algorithm needs exponential time.

The structure of the graph is first described. For each n, $n \geq 1$, the graph has n min vertices (1, 2, ..., n), n average vertices ($1'$, $2'$, ..., n'), and, of course, the two sinks. The graphs for n equal to 2 and 3 are presented in Figures 2.1 and 2.2 respectively.

The structure can be described recursively; to construct the graph for n min vertices, add to the graph for $n-1$ vertices the following: the average vertex n', with two edges going to the min vertex $n-2$ and the average vertex $(n-1)'$, and the min vertex n, with an edge labeled 0 to the previous min vertex, $n-1$, and an edge labeled 1 to the new average vertex n'. The graph for general n is presented in Figure 2.3.

In all these examples we suppose that the initial strategy of the policy iteration algorithm is represented by a 0-vector ($S = S_n S_{n-1} \cdots S_1 = 00...0$). With respect to this strategy, every min vertex has

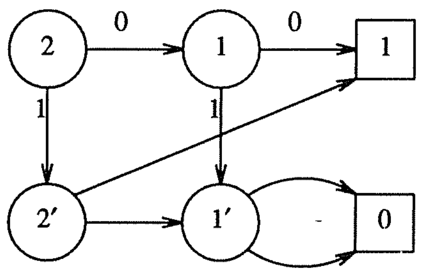

Figure 2.1: Counterexample for the simple policy iteration algorithm for $n = 2$.

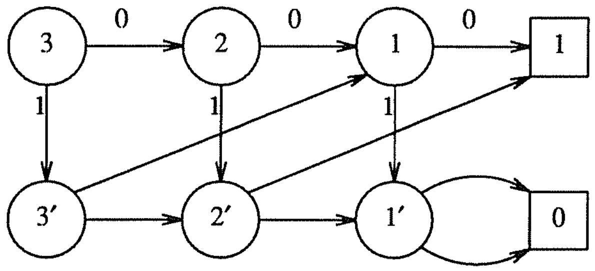

Figure 2.2: Counterexample for the simple policy iteration algorithm for $n = 3$.

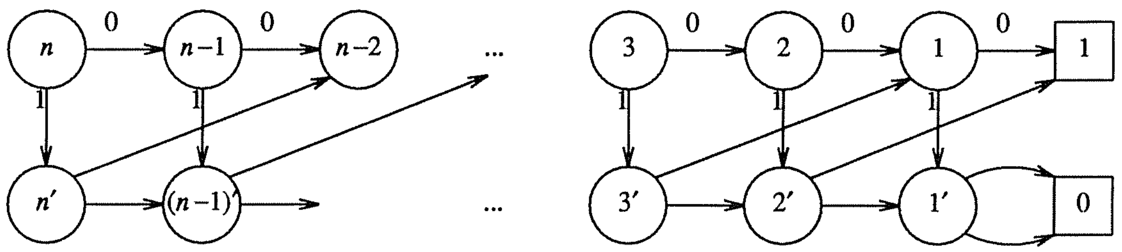

Figure 2.3: Counterexample for the simple policy iteration algorithm for general n.

value 1. It is easy to see that the best strategy is edge 0 for every min vertex except vertex 1, and edge 1 for that (S = 00...01). In this case, the value of every min vertex is 0, so this strategy is the best (note that the best strategy is unique for this structure). Although this does not differ a lot from the initial strategy (it may be found in one step by a clever algorithm), the simple policy iteration algorithm takes 2^n-1 steps to find it.

We next describe how the simple policy iteration algorithm performs for the first graph, where n is 2. Initially both min vertices 1 and 2 are switchable (because average vertex 1′ has value 0, and 2′ has value 1/2). The simple policy iteration algorithm switches min vertex 2 first and then 1. Vertex 2 is switchable again, back to edge 0 (its value can be decreased from 1/2 to 0). We have a total of $2^2-1 = 3$ switches.

The sequence of strategies and values is summarized in the table of Figure 2.4, where S denotes the strategy, $V(i)$ denotes the value of vertex i, and t denotes the number of switches. The value of the average vertex $2'$ is constant (equal to 1/2), so it is not in the table. The table of Figure 2.5 summarizes the same information for n equal to 3.

t	$S = S_2 S_1$	$V(2)$	$V(1)$
0	00	1	1
1	10	1/2	1
2	11	1/2	0
3	01	0	0

Figure 2.4: Table of sequence of strategies and values for the graph of Figure 2.1.

t	$S = S_3 S_2 S_1$	$V(3)$	$V(3')$	$V(2)$	$V(1)$
0	000	1	3/4	1	1
1	100	3/4	3/4	1	1
2	110	3/4	3/4	1/2	1
3	010	1/2	3/4	1/2	1
4	011	1/2	1/4	1/2	0
5	111	1/4	1/4	1/2	0
6	101	1/4	1/4	0	0
7	001	0	1/4	0	0

Figure 2.5: Table of sequence of strategies and values for the graph of Figure 2.2.

Note that every time that vertex i is switched, from edge 0 to 1 or from edge 1 to 0, S_{i+1} is equal to 1, and S_k is 0, for every $k > i+1$.

The values $V(1)$, $V(2)$, $V(3)$ can be expressed by the following expressions (if n is greater than or equal to 3):

$$V(1) = 1 - S_1$$

$$V(2') = \frac{1}{2}$$

$$V(2) = S_2 V(2') + (1-S_2)V(1) = V(1) - S_2\left[\frac{1}{2} - S_1\right].$$

The value $V(3)$ is derived in the same way:

$$V(3) = V(2) - S_3\left[\frac{1}{2} - S_1\right]\left[\frac{1}{2} - S_2\right].$$

This leads us to Lemma 2.2 which is the first step of the proof that the simple policy iteration algorithm makes an exponential number of switches to reach the best strategy for this structure. In the main lemma of this proof (Lemma 2.7) we show that if the strategy for the leftmost k vertices is 0...00, and these vertices are switchable, the next 2^k-1 switches of the algorithm are made on these vertices to reach the strategy 0...01 for these vertices. The most important step to that lemma is to express the value of every vertex with a formula that depends on the current strategy and on values of other vertices (Lemma 2.2). (After that point we do not need the graph.) The formulas give necessary and sufficient conditions (depending only on the current strategy), for a vertex to be switchable (Corollary 2.4). The next two results (Corollary 2.5 and Lemma 2.6, which are also used in the proof of the main lemma) give relations between the switching of a vertex and other vertices becoming switchable.

Definition 2.1. $a(k)$, for every positive k, is defined to be

$$a(k+1) = a(k)\left[\frac{1}{2} - S_k\right] \text{ and } a(1) = -1.$$

The next lemma is technical and it is only used in the proof of Lemma 2.2.

Lemma 2.1. For every $2 \le k \le n$:

$$V(k') - V(k-1) = \frac{a(k)}{a(k-1)}\left[V((k-1)') - V(k-2)\right],$$

where $V(0)$ should be substituted by 1 in case k is 2.

Proof:

$$V(k') - V(k-1) = \left[\frac{1}{2}V((k-1)') + \frac{1}{2}V(k-2)\right] - \left[S_{k-1}V((k-1)') + (1-S_{k-1})V(k-2)\right] \text{ (by the construction)} =$$

$$= \left[\frac{1}{2} - S_{k-1}\right]\left[V((k-1)') - V(k-2)\right] =$$

$$= \frac{a(k)}{a(k-1)}\left[V((k-1)') - V(k-2)\right] \text{ (by Definition 2.1.) } \square$$

Lemma 2.2. For every k, $2 \le k \le n$, the values of the vertices, with respect to the current strategy S, are given by the following formulas:

$$V(k) = V(k-1) + S_k a(k) \text{ and } V(1) = 1 + S_1 a(1), \qquad \text{(I)}$$

$$V(k') = V(k-1) + a(k),\ \ V(1') = 1 + a(1), \qquad \text{(II)}$$

where $a(k)$ is as in Definition 2.1.

Proof: The formulas can be proved by induction on k. The pair (I) is proved first:

Basis Case: It has been proved that (I) holds for k equal to 1, 2 and 3 by the relations presented before the lemma.

Induction Hypothesis: We suppose that (I) holds for every $k \le m$.

Induction Step: We prove that (I) holds for k equal to $m+1$.

$$V(m+1) = S_{m+1} V((m+1)') + (1-S_{m+1})V(m) \text{ (by the construction)} =$$

$$= V(m) + S_{m+1}\left[V((m+1)') - V(m-1) - S_m a(m)\right] \text{ (induction hypothesis)} =$$

$$= V(m) + S_{m+1}\left[\frac{1}{2}V(m') + \frac{1}{2}V(m-1) - V(m-1) - S_m a(m)\right] \text{ (by the construction)} =$$

$$= V(m) + S_{m+1}\left[\frac{1}{2}\frac{a(m)}{a(m-1)}(V((m-1)') - V(m-2)) - S_m a(m)\right] \text{ (by Lemma 2.1)} = \cdots =$$

$$= V(m) + S_{m+1}\left[\frac{1}{2}\frac{a(m)}{a(1)}(V(1') - 1) - S_m a(m)\right] \text{ (by applying Lemma 2.1 } m-2 \text{ times)} =$$

$$= V(m) + S_{m+1} a(m)\left[\frac{1}{2} - S_m\right] \text{ (by the construction)} =$$

$$= V(m) + S_{m+1} a(m+1) \text{ (by Definition 2.1).}$$

Proof of the pair (II):

Basis Case: The formulas obviously hold for k equal to 1 and 2.

Induction Hypothesis: We suppose that (II) holds for every $k \le m$.

Induction Step: We prove that (II) holds for k equal to $m+1$.

$$V((m+1)') = \frac{1}{2}\left[V(m') + V(m-1)\right] \text{ (by the construction)} =$$

$$= V(m-1) + \frac{1}{2}a(m) \text{ (induction hypothesis)} =$$

$$= V(m) + \left[\frac{1}{2} - S_m\right]a(m) \text{ (by applying (I) for } m\text{)} =$$

$$= V(m) + a(m+1) \text{ (by Definition 2.1).} \ \square$$

From Definition 2.1, the following corollary is easily derived.

Corollary 2.3. If $a(k)$ is as Definition 2.1, then the following holds, for every positive k.

$$a(k+1) = \frac{(-1)^{S_k}}{2} a(k)$$

Proof: The proof is based on the recursive formula for $a(k+1)$: $a(k+1) = a(k)\left[\frac{1}{2} - S_k\right]$.

If S_k is equal to 1 then $a(k+1) = -\frac{1}{2}a(k) = \frac{(-1)^{S_k}}{2}a(k)$. If S_k is equal to 0 then $a(k+1) = \frac{1}{2}a(k) =$

$\frac{(-1)^{S_k}}{2}a(k)$. □

Corollary 2.4. Vertex k is switchable if and only if either S_k is 0 and $a(k)$ is negative, or S_k is 1 and $a(k)$ is positive, for every positive k.

Proof: Proof that if k is switchable and S_k is 0, $a(k)$ is negative:
If k is greater than 1 then, according to Lemma 2.2,

$$V(k) = V(k-1) + S_k \bar{a}(k).$$

Since S_k is 0, $V(k) = V(k-1)$. Since the switch of the vertex to S_k equal to 1 gives a smaller value,

$$V(k-1) > V(k-1) + a(k) => a(k) < 0.$$

The proof also works for vertex 1 and the formula $V(1) = 1 + S_1 a(1)$. The rest can be proved by a very similar reasoning. □

Corollary 2.5. If vertex k (for every positive k) is switchable, and one or more larger numbered vertices are switched arbitrarily, vertex k is still switchable.

Proof: The fact that k is switchable means that either S_k is 0 and $a(k)$ is negative, or S_k is 1 and $a(k)$ is positive. Since the switches of larger numbered vertices do not have any effect on the value of $a(k)$, k is still switchable after these switches. □

Lemma 2.6. If vertex k (for every positive k less than n) is switched, and $S_n \cdots S_{k+2}S_{k+1} = 0...01$, then all the vertices $k+1, k+2, ..., n$ are switchable.

Proof: First, suppose that S_k is switched from 0 to 1. From Corollary 2.4, it is deduced that $a(k)$ is negative. By Corollary 2.3, $a(k+1)$ is positive, so by Corollary 2.4, vertex $k+1$ is switchable. By Corollary 2.3, $a(k+2)$ is negative, so by Corollary 2.4, vertex $k+2$ is switchable. It can be proved by induction that the other larger numbered vertices are switchable, too.
If S_k is switched from 1 to 0, the proof that the lemma holds is the same. □

Lemma 2.7. The following two statements hold (for every positive k):
— If $S_n \cdots S_{k+1}S_k = 0...01$, and the vertices $k, k+1, .., n$ are switchable, the next $2^{n-k+1}-1$ switches of the simple policy iteration algorithm are made on these vertices, to reach the strategy where $S_n \cdots S_{k+1}S_k =$ 0...00.
— If $S_n \cdots S_{k+1}S_k = 0...00$, and the vertices $k, k+1, .., n$ are switchable, the next $2^{n-k+1}-1$ switches of the simple policy iteration algorithm are made on these vertices, to reach the strategy where $S_n \cdots S_{k+1}S_k =$ 0...01.

Proof: The lemma will be proved by induction on k.
Basis Case: It is obvious that the statements hold for the case $k = n$.
Induction Hypothesis: The statements hold for every $k \geq m$.
Induction Step: We will prove that the statements hold for k equal to $m-1$. We prove the first one first:

The simple policy iteration algorithm works first on the vertices numbered n, $n-1$, ..., m, since the select procedure always chooses the highest numbered switchable vertex. From the second statement of the induction hypothesis, we deduce that it performs $2^{n-m+1}-1$ switches to reach the strategy where $S_n \cdots S_{k+1} S_m = 0...01$. Vertex $m-1$ is still switchable (Corollary 2.5), so it is switched (from $S_{m-1} = 1$ to 0). By Lemma 2.6, the vertices n, $n-1$, ..., m are again all switchable. From the first statement of the induction hypothesis, we deduce that the simple policy iteration algorithm performs $2^{n-m+1}-1$ switches on these vertices to reach the strategy where $S_n \cdots S_{k+1} S_m = 0...00$. The total number of the switches is $2^{n-m+2}-1$.
The second statement is proved following the same reasoning. □

We can now present the main result of the section.

Lemma 2.8. The simple policy iteration algorithm requires exponential time in the worst case. For every positive n, the algorithm makes 2^n-1 switches, on the graph with n min vertices of Figure 2.3, to find the best strategy (vector 00...01) if the initial strategy is the 0-vector.

Proof: It follows from the previous results if initially every vertex is switchable. Since the initial strategy is the 0-vector, and $a(1)$ is negative, every other $a(k)$ is negative, too; so, by Corollary 2.4, every vertex is switchable. □

Two interesting observations on the values of the vertices are the following statements which can be easily derived. (a) Min vertex n is switched 2^{n-1} times. Initially its value is 1, and it is decreased by $1/2^{n-1}$ each time. (b) The average vertex $(n+1)'$'s value is changed $2^{n-1}-1$ times. Initially its value is $(2^n-1)/2^n$, and it is decreased by $1/2^{n-1}$ each time.

3. Topological Policy Iteration Algorithm

The select procedure of this algorithm needs some preprocessing of the graph given. The vertices are topologically sorted, that is, an integer order is assigned to each min vertex, such that if there is a path from vertex i to vertex j, then the order of i is greater or equal to the order of j. The graph is in this way divided into components of vertices with the same order. The select procedure selects the largest numbered switchable vertex in the component with the lowest order that contains switchable vertices. In this way, when a vertex i is switched, there is a decrease in the value of every vertex with larger order from which there is a path to i, according to the current strategy.

Note that this algorithm, applied on the structure of the previous section, finds the best strategy after one switch (if the initial strategy is the 0-vector). The algorithm divides the graph into components, where each min vertex is found to be in its own component. According to the topological sorting, the component of min vertex i comes just after the component of min vertex $i-1$, for every $i > 1$, so vertex 1 is selected first.

If a directed graph is divided into k strongly connected components, and the numbers of min vertices in them are $n_1, n_2, ..., n_k$, then, in the worst case the algorithm takes $2^{n_1} + 2^{n_2} + \cdots + 2^{n_k}$ steps. If the maximum size of a component is bounded by a constant, the algorithm can find a best strategy in a polynomial number of steps. In particular, if the graph is acyclic the algorithm makes a linear number of switches.

In the rest of this section we present a counterexample for this algorithm. The structure is very similar to the one of the previous section. The main difference is that, in this one, min vertex 1 depends on min vertex n, through a new average vertex, $0'$. Therefore this structure is cyclic, there is only one component, and the algorithm performs like the simple policy iteration algorithm. We will prove that the algorithm needs an exponential number of steps to find the best strategy, which is unique for this structure too.

There is another, slight, difference between the structures. In the previous structure, there is a choice for vertex 1 to get the value 0. This is a weak point of the structure in the sense that any algorithm could easily find that, start from that switch (because this is the minimum value that a vertex can get), and end with only one switch. In the new structure there is no such choice (the value of $1'$ is 1/2).

The graphs for 2, 3 and, generally, n min vertices are presented in Figures 3.1, 3.2 and 3.3.

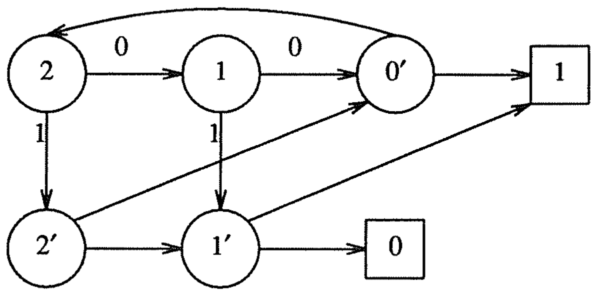

Figure 3.1: Counterexample for the topological policy iteration algorithm for $n = 2$.

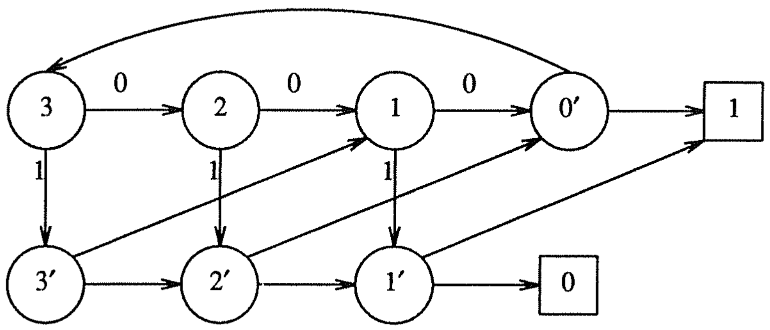

Figure 3.2: Counterexample for the topological policy iteration algorithm for $n = 3$.

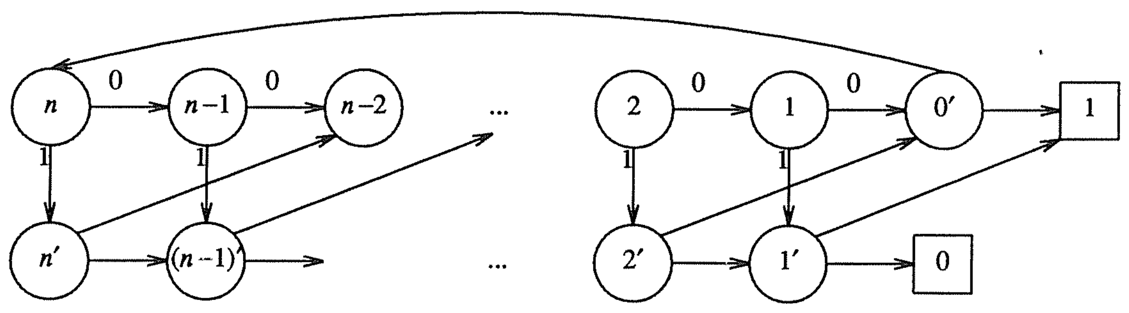

Figure 3.3: Counterexample for the topological policy iteration algorithm for general n.

The graph for general n has n min vertices, and $n+1$ average vertices (labeled $0'$, $1'$, ..., n'). To construct the graph for $n+1$, given this graph, we add a new min vertex, vertex $n+1$, and a new average vertex, vertex $(n+1)'$. The two edges for vertex $n+1$ lead to vertex $(n+1)'$ and vertex n. Vertex $(n+1)'$ is the average of n' and $n-1$ (or $0'$ if n is 1). Also, there is an edge from $0'$ to min vertex $n+1$ instead of n.

Starting from the same initial strategy (0-vector), the topological policy iteration algorithm follows the same sequence of strategies as the simple policy iteration algorithm of the previous section (note that the strategy $S = 00...01$ is the unique best strategy that gives value 1/2 to every min vertex). The tables in Figures 3.4 and 3.5 present the sequence of strategies and values for the graphs with two and three min vertices.

t	$S = S_2 S_1$	$V(2)$	$V(2')$	$V(1)$	$V(0')$
0	00	1	3/4	1	1
1	10	2/3	2/3	5/6	5/6
2	11	2/3	2/3	1/2	5/6
3	01	1/2	5/8	1/2	3/4

Figure 3.4: Table of sequence of strategies and values for the graph of Figure 3.1.

Using the same technique that we used in Section 2, we can express the values $V(0')$, $V(1)$, $V(2)$, $V(3)$ by the following expressions (if n is greater than or equal to 3):

$$V(0') = \frac{1}{2}(1 + V(n)) = \frac{1}{2} + \frac{V(n)}{2},$$

$$V(1) = S_1 V(1') + (1-S_1)V(0') = \frac{S_1}{2} + (1-S_1)\left[\frac{1}{2} + \frac{V(n)}{2}\right] = \frac{1}{2} + \frac{V(n)}{2} - \frac{S_1 V(n)}{2} = V(0') - \frac{S_1 V(n)}{2},$$

$$V(2') = \frac{1}{2}(V(1') + V(0')) = \frac{1}{2}\left[\frac{1}{2} + \frac{1}{2} + \frac{V(n)}{2}\right] = \frac{1}{2} + \frac{V(n)}{4},$$

t	$S=S_3S_2S_1$	$V(3)$	$V(3')$	$V(2)$	$V(2')$	$V(1)$	$V(0')$
0	000	1	7/8	1	3/4	1	1
1	100	4/5	4/5	9/10	7/10	9/10	9/10
2	110	4/5	4/5	7/10	7/10	9/10	9/10
3	010	2/3	3/4	2/3	2/3	5/6	5/6
4	011	2/3	7/12	2/3	2/3	1/2	5/6
5	111	4/7	4/7	9/14	9/14	1/2	11/14
6	101	4/7	4/7	1/2	9/14	1/2	11/14
7	001	1/2	9/16	1/2	5/8	1/2	3/4

Figure 3.5: Table of sequence of strategies and values for the graph of Figure 3.2.

$$V(2)=S_2V(2')+(1-S_2)V(1)=V(1)-\frac{S_2V(n)}{4}+\frac{S_1S_2V(n)}{2}=V(1)-\frac{S_2}{2}\left[\frac{1}{2}-S_1\right]V(n).$$

The value $V(3)$ is derived in the same way:

$$V(3)=V(2)-\frac{S_3}{2}\left[\frac{1}{2}-S_1\right]\left[\frac{1}{2}-S_2\right]V(n).$$

These formulas leads us to Lemma 3.2 which is the first step of the proof that the topological policy iteration algorithm makes an exponential number of switches to reach the best strategy for this structure. The proof consists of a similar sequence of lemmas that were used to prove the same result for the simple policy iteration algorithm on the structure of the previous section. We omit the proofs that are very similar to the proofs of the corresponding lemmas of the previous section.

Definition 3.1. $a(k)$, for every positive k, is defined to be

$$a(k+1)=a(k)\left[\frac{1}{2}-S_k\right] \text{ and } a(1)=-\frac{1}{2}.$$

(Note that $a(1)$ is different from Definition 2.1.) The next lemma states the same result as Lemma 2.1.

Lemma 3.1. For every $2\le k\le n$:

$$V(k')-V(k-1)=\frac{a(k)}{a(k-1)}\left[V((k-1)')-V(k-2)\right],$$

where $V(0)$ should be substituted by $V(0')$ in case k is 2.

Lemma 3.2. For every k, $2\le k\le n$, the values of the vertices, with respect to the current strategy S, are given by the following formulas:

$$V(k)=V(k-1)+S_kV(n)a(k) \text{ and } V(1)=V(0')+S_1V(n)a(1), \qquad \text{(I)}$$

$$V(k') = V(k-1) + V(n)a(k),\ \ V(1') = V(0') + V(n)a(1),\ \text{ and }\ V(0') = \frac{1}{2} + \frac{V(n)}{2}, \tag{II}$$

where $a(k)$ is as in Definition 3.1.

Proof: The proof for (I) is the same as the proof of the pair (I) of Lemma 2.2. Pair (II) can be proved by induction in the same way as in the proof of the pair (II) of Lemma 2.2, except for the basis case for k equal to 2.

$$V(2') = V(1) + V(n)a(2) = V(0') + S_1 V(n)a(1) + V(n)a(1)\left[\frac{1}{2} - S_1\right] =$$

$$= \frac{1}{2} + \frac{V(n)}{2} + \frac{1}{2}V(n)a(1) = \frac{1}{2} + \frac{V(n)}{4}.\square$$

Corollary 2.3 holds obviously again and it is derived in exactly the same way for this case (from Definition 3.1).

Corollary 3.3. If $a(k)$ is as Definition 3.1, then the following holds, for every positive k.

$$a(k+1) = \frac{(-1)^{S_k}}{2}a(k)$$

Corollary 3.4. Vertex k is switchable if and only if either S_k is 0 and $a(k)$ is negative, or S_k is 1 and $a(k)$ is positive, for every positive k.

Proof: The proof is the same as the proof of Corollary 2.4, since $V(n)$ is always positive. □

Corollary 3.5. If vertex k (for every positive k) is switchable, and one or more larger numbered vertices are switched arbitrarily, vertex k is still switchable.

Lemma 3.6. If vertex k (for every positive k less than n) is switched, and $S_n \cdots S_{k+2}S_{k+1} = 0...01$, then all the vertices $k+1, k+2, ..., n$ are switchable.

Lemma 3.7. The following two statements hold (for every positive k):
— If $S_n \cdots S_{k+1}S_k = 0...01$, and the vertices $k, k+1, .., n$ are switchable, the next $2^{n-k+1}-1$ switches of the topological policy iteration algorithm are made on these vertices, to reach the strategy where $S_n \cdots S_{k+1}S_k = 0...00$.
— If $S_n \cdots S_{k+1}S_k = 0...00$, and the vertices $k, k+1, .., n$ are switchable, the next $2^{n-k+1}-1$ switches of the topological policy iteration algorithm are made on these vertices, to reach the strategy where $S_n \cdots S_{k+1}S_k = 0...01$.

We can now present the main result of the section.

Lemma 3.8. The topological policy iteration algorithm requires exponential time in the worst case. For every positive n, the algorithm makes 2^n-1 switches, on the graph with n min vertices of Figure 3.3, to find the best strategy (vector 00...01) if the initial strategy is the 0-vector.

Proof: It follows from the previous results (remember that the topological policy iteration algorithm works like the simple one for this structure), if initially every vertex is switchable. Since the initial strategy is the 0-vector, and $a(1)$ is negative, every other $a(k)$ is negative, too; so, by Corollary 3.4, every vertex is switchable. □

4. Difference Policy Iteration Algorithm

As in the topological policy algorithm, the vertices are first topologically ordered. The select procedure then computes for every switchable min vertex in the component with the lowest order that contains switchable vertices, the difference between the values of the two vertices that its two outgoing edges lead to, and selects the vertex with the largest difference (if more than one vertex has this difference, the largest numbered one is selected).

We believe that this is a natural approach to the problem of reducing the maximum number of switches required by the policy iteration algorithm, because when a vertex with a large difference is switched, the decreases of the values of the vertices with larger order may be large too. The algorithm performs well for many structures. If the initial strategy is the 0-vector, the algorithm finds the best one in one switch for the structure presented in Section 3. However a small change in that structure can mislead the algorithm and make it run for exponentially many steps. In the rest of the section the new counterexample is presented.

The new structure for n min vertices is constructed by adding the following gadget in appropriate places of the graph of Figure 3.3. The **gadget g_k** is presented in Figure 4.1.

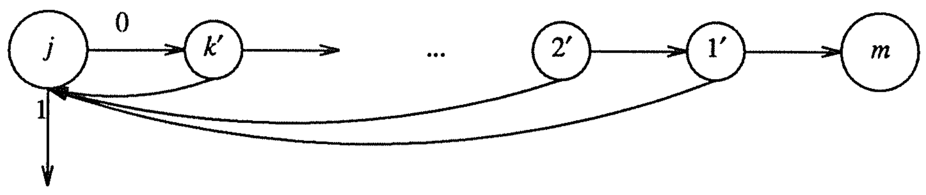

Figure 4.1: Gadget g_k

The gadget g_k is placed between a min vertex j and another, m, which can be a min, average or sink vertex. It consists of k average vertices, $1', 2', ..., k'$. Vertex i' ($1 \le i \le k$) has one edge going to min vertex j and another going to the previous average vertex, $(i-1)'$, or to vertex m if i is 1.

It is easy to notice that this gadget has the following properties:

— If a strategy contains the edge (j, k'), then the value of j is equal to the value of m, so the gadget has no effect.

— If a strategy contains the other edge from j, then

$$V(k') = \frac{(2^k-1)}{2^k} V(j) + \frac{V(m)}{2^k} \Rightarrow V(j) - V(k') = \frac{(V(j) - V(m))}{2^k},$$

so the effect in this case is that the difference between the values of the neighbors of vertex j has been decreased by a factor of 2^k.

Figure 4.2 presents the graph for two min vertices (which is produced by adding g_2 to the graph of Figure 3.1). The difference between the values of the neighbors of vertex 1 is 1/8, whereas the corresponding difference for vertex 2 is 1/4, so it is first switched, and the algorithm falls into the well known, by now, scenario which takes 3 switches.

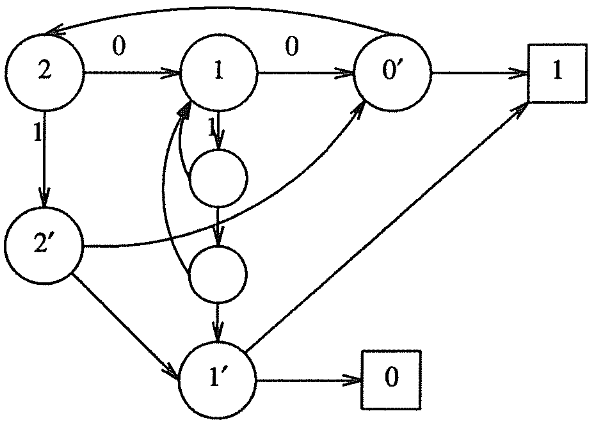

Figure 4.2: Counterexample for the difference policy iteration algorithm for $n = 2$.

Figure 4.3 presents the graph for three min vertices.

To construct the graph on n min vertices, we first construct the graph on n min vertices, of the structure presented in Section 3 (counterexample for the topological policy iteration algorithm). Then we add the appropriate gadgets in the following way: an instance of $g_{2(n-k)}$ is added between vertex k and each of its two neighbors, for every k, $1 < k < n$, and $g_{2(n-1)}$ is added only between vertex 1 and 1′ (because vertex 1 is switched only once). Note that the number of new average vertices is polynomial ($2(n-1)^2$) in the number of the min vertices n.

Since all the min vertices lie on a cycle, the topological ordering does not affect the choices of switchable vertices. We suppose again that the initial strategy is the 0-vector. There is a unique best strategy, $S = 00...01$, for the structure with respect to which the value of every min vertex is 1/2.

Lemma 4.1. The difference policy iteration algorithm requires exponential time in the worst case. For every positive n, the algorithm makes 2^n-1 switches on the graph on n min vertices to find the best strategy (vector 00...01) if the initial strategy is the 0-vector.

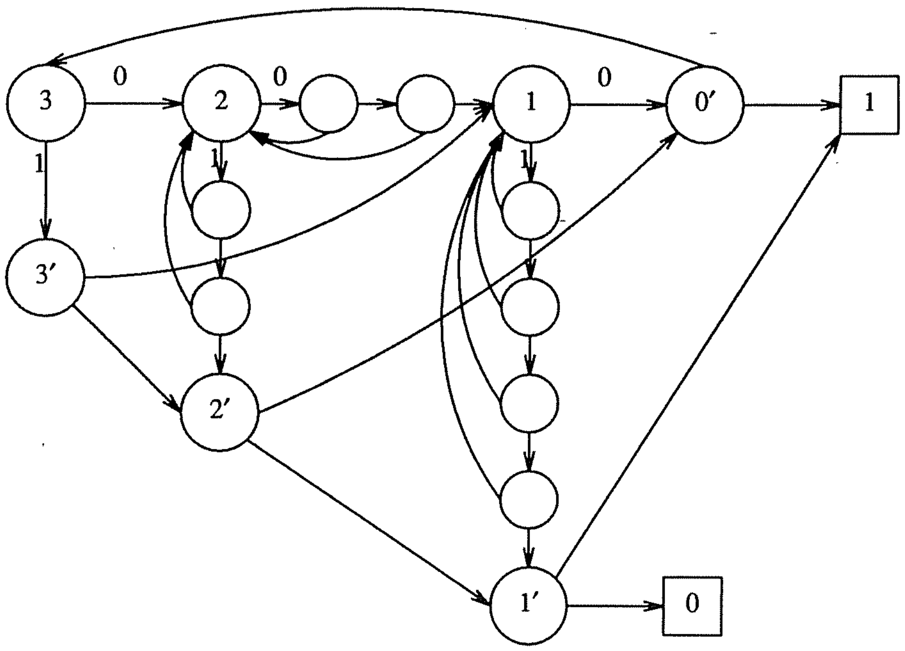

Figure 4.3: Counterexample for the difference policy iteration algorithm for $n = 3$.

Proof: Lemma 3.2 can be used for this proof. Note that the lemma holds for this structure too, because the addition of the gadgets does not affect the values that the vertices get, but only hides the actual difference between the min vertices. According to the lemma, the value of vertex k ($k \le n$), according to the strategy $S = S_n S_{n-1} \ldots S_1$, is

$$V(k) = V(k-1) + S_k V(n) a(k) \text{ and } V(1) = V(0') + S_1 V(n) a(1).$$

From these formulas it follows that the difference between the values of the two neighbors of vertex k is $V(n)a(k)$, or $\frac{V(n)}{2^k}$ (by Corollary 3.3), for the original structure (without the gadgets). Note that the difference is constant for every vertex and does not depend on the strategy. With the gadgets added to it, the difference for k becomes $\frac{V(n)}{2^{2n-k}}$, which is an increasing function of k, so the algorithm performs like the simple policy algorithm, which, as proved in Section 3, takes an exponential number of steps for the structure. □

5. Improvement Policy Iteration Algorithm

The main result from the previous section is that the difference between the values of the two neighbors of a vertex is not proportional to the decrease that will actually result by making the switch. So, that greedy algorithm can require exponential time. A better approach would be to compute for each switchable vertex the decrease that its switch will give, and decide which vertex to select based on that. This observation leads us to the next greedy algorithm, the Improvement Policy Iteration Algorithm.

The select procedure of this algorithm first topologically sorts the vertices of the graph. For every switchable min vertex in the component with the lowest order that contains switchable vertices, the decrease of the value of the vertex if it is switched is computed. The vertex that will get the largest decrease is selected (if more than one vertex can get this decrease, the largest numbered one is selected).

On the structure presented in the previous section the algorithm can find the best strategy in one step, if the initial strategy is the 0-vector. Unfortunately, based on the same structure, we can construct another, on which the improvement policy iteration algorithm takes an exponential number of steps to find the best strategy. In the rest of this section we will present the new structure and we will prove that the algorithm makes an exponential number of switches on that.

For the new structure we will use another gadget; the **gadget $g_{m,l}$** is presented in Figure 5.1.

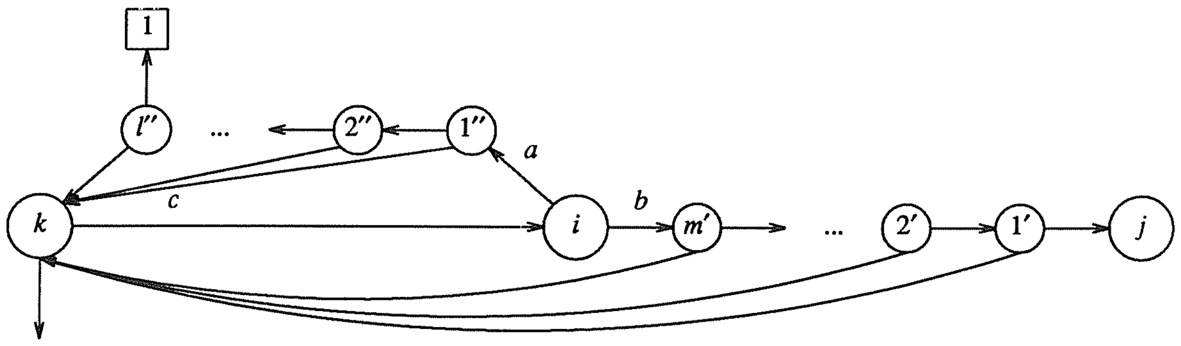

Figure 5.1: Gadget $g_{m,l}$

The gadget $g_{m,l}$ is placed between a min vertex k and another vertex j, that can be a min, average or sink vertex. The gadget introduces a new min vertex i, and $l+m$ new average vertices. Each vertex p' of the set of m average vertices, has one edge going to min vertex k and another going to the previous average vertex $(p-1)'$ or to vertex j if p is 1. Each vertex p'' of the set of l average vertices (double-primed vertices), has one edge going to min vertex k and another going to the next average vertex $(p+1)''$ or to the 1-sink if p is l. Finally, there are edges from the new min vertex i to the average vertices $1''$ and m', and an edge from min vertex k to min vertex i. The letters a, b and c denote the labels (0 or 1) of the edges $(i, 1'')$, (i, m') and (k, i), respectively.

Roughly, the key property of this gadget is as follows. If $S_k \neq c$, $S_i \neq b$ and $V(j)$ is much smaller than $V(k)$, then k and i can decrease in value by a large amount if **both** are switched. However, if only one of k or i is switched, the decrease in value of k or i is small.

It can be easily checked that the gadget has the following properties:

— If $S_k = c$ and $S_i = b$, then $V(k) = V(j)$.

— If $S_i = a$, then $V(i) = (1 - 2^{-l})V(k) + 2^{-l}$.

— If $S_i = b$, then $V(i) = (1 - 2^{-m})V(k) + 2^{-m}V(j)$.

— If $S_k \neq c$, then the improvement of $V(i)$ when it is switched from a to b is:

$$V(k) + 2^{-l}(1 - V(k)) - V(k) - 2^{-m}(V(j) - V(k)) = 2^{-m}(V(k) - V(j)) + 2^{-l}(1 - V(k)).$$

— If $S_k \neq c$, then the improvement of $V(i)$ when it is switched from b to a is:

$$V(k) + 2^{-m}(V(j) - V(k)) - V(k) - 2^{-l}(1 - V(k)) = 2^{-m}(V(j) - V(k)) - 2^{-l}(1 - V(k)).$$

The last two properties make sense only if the value of vertices j and k are the same before and after the switch of vertex i, because in this case $V(j)$ and $V(k)$ are the same with respect to the current strategy before and after the switch. This is true for the structure that will be presented because no edge (other than the edge labeled c in the gadget) will connect any vertex with vertex i.

Figure 5.2 presents the graph for n equal to 2. Gadget $g_{0,6}$ is used twice, between vertices 2 and 1, and between vertices 2 and 2′. Gadget $g_{2,6}$ is placed between vertices 1 and 0′, and between vertices 1 and 1′. Initially, only vertices 1_1 and 2_1 are switchable. 1_1 can get an improvement of 1/8, and 2_1 an improvement of 1/4, so 2_1 is switched first. Vertices 1_1 (improvement 1/8), and 2 (improvement 1/3) are now switchable, so 2 is switched next. Vertex 1_1 is still switchable (improvement 9/128), and vertex 2_0 (improvement 31/192) too. Therefore, 2_0 is selected. Vertex 1_1 is the only switchable one, and it is switched next. Again, only one vertex is switchable, vertex 1, so it is switched. Vertices 1_0 (improvement 29/384), and 2_0 (improvement 11/64) are now switchable, and 2_0 is selected. Between the switchable vertices 1_0 (improvement 29/384), and 2 (improvement 1/6), vertex 2 is selected. Vertex 1_0 is still switchable (improvement 7/128) but it is only switched after vertex 2_1 (improvement 15/128). At this point none of the vertices is switchable. The total number of switches made is 9. The sequence of strategies and values are summarized in the table of Figure 5.3.

To construct the graph for general n, G_n, we first construct the graph on n vertices, of the structure presented in Section 3 (counterexample for the topological policy iteration algorithm). Then we add the appropriate gadgets in the following way: an instance of $g_{2(n-k),2n+2}$ is added between vertex k and each of its two neighbors, for every k, $1 \leq k \leq n$. In this gadget, the new min vertex between vertex k and the average vertex k' is labeled k_1, and the new min vertex between vertex k and the previous min vertex $k-1$ (or average vertex 0′ if k is 1) is labeled k_0. The edge of k_1 that goes vertically down is labeled with 1, and the other one with 0. The edge of k_0 that goes horizontally right is labeled with 0, and the other one with 1. Note that the number of new vertices is polynomial in the number of the original vertices (the number of the new min vertices is $2n$, and the number of the new average vertices is $6n^2 + 2n$).

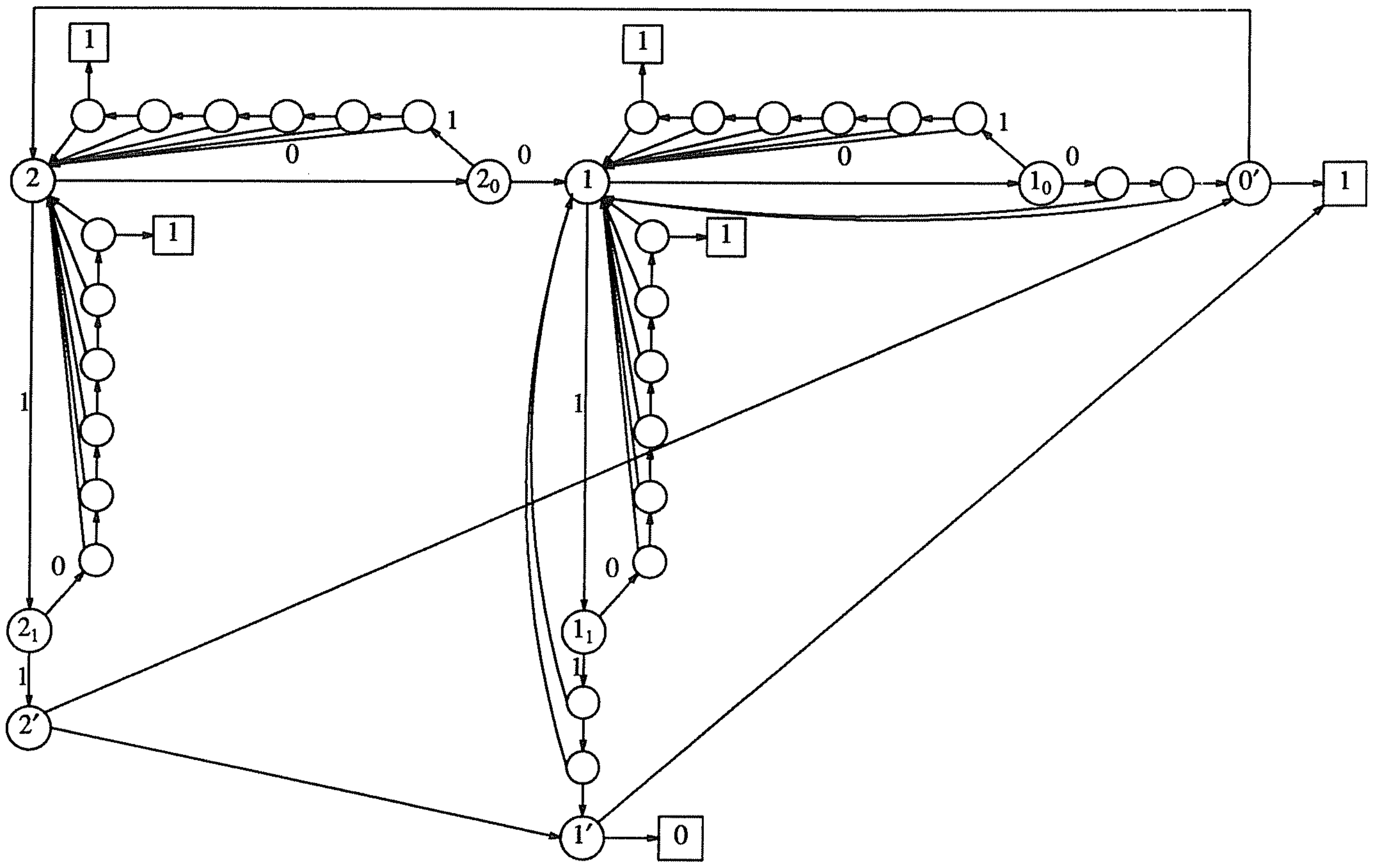

Figure 5.2: Counterexample for the improvement policy iteration algorithm for $n = 2$.

The topological ordering does not affect the choices of switchable vertices because all the min vertices are given the same order. The initial strategy will be assumed to be the 0-vector. The best strategy for the structure is unique (strategy S where $S_n \cdots S_2 S_1 = S_{n_0} \cdots S_{2_0} S_{1_0} = S_{n_1} \cdots S_{2_1} S_{1_1} = 0...01$).

The proof that the algorithm makes an exponential number of switches before it reaches the best strategy will follow the steps that were followed in Section 3 (we will only need a new definition and a new lemma). We will omit the proofs that are the same as the proofs of the corresponding lemmas in Section 3.

Definition 5.1. $a(k)$, for every positive k, is defined to be

$$a(k+1) = a(k)\left[\frac{1}{2} - S_k\right] \text{ and } a(1) = -\frac{1}{2}.$$

Definition 5.2. A strategy is defined to be **stable** if for all k, either

— $S_k = 0$ and $S_{k_0} = 0$, or

t	$S = S_2 S_{2_0} S_{2_1} S_1 S_{1_0} S_{1_1}$	$V(2)$	$V(2_0)$	$V(2_1)$	$V(1)$	$V(1_0)$	$V(1_1)$	$V(0')$
0	000000	1	1	1	1	1	1	1
1	001000	1	1	3/4	1	1	1	1
2	101000	2/3	5/6	2/3	5/6	5/6	105/128	5/6
3	111000	2/3	43/64	2/3	5/6	5/6	105/128	5/6
4	111001	2/3	43/64	2/3	5/6	5/6	3/4	5/6
5	111101	2/3	43/64	2/3	1/2	7/12	1/2	5/6
6	101101	2/3	1/2	2/3	1/2	7/12	1/2	5/6
7	001101	1/2	1/2	5/8	1/2	9/16	1/2	3/4
8	000101	1/2	1/2	65/128	1/2	9/16	1/2	3/4
9	000111	1/2	1/2	65/128	1/2	65/128	1/2	3/4

Figure 5.3: Table of sequence of strategies and values for the graph of Figure 5.2.

— $S_k = 1$ and $S_{k_1} = 1$.

Note that from any stable strategy, it is impossible to reach an unstable strategy on any iteration of any policy iteration algorithm. This is because this would result in value 1 for vertices k and k_0, or k and k_1. The reason we introduce this definition is because the sequence of lemmas of Section 3 can only be extended to this structure if the current strategy of the improvement policy itertion algorithm is always stable.

Lemma 5.1. If the current strategy is stable, for every $2 \le k \le n$:

$$V(k') - V(k-1) = \frac{a(k)}{a(k-1)} \left[V((k-1)') - V(k-2) \right],$$

where $V(0)$ should be substituted by $V(0')$ in case k is 2.

Proof: The proof is the same as the proof of Lemma 3.1, since the current strategy is stable, which, combined with the first property of the gadget, means that the original vertices of the graph (vertices 1, ..., n and $0'$, $1'$, ..., n') take the same sequence of values that they would take without the addition of the gadgets. □

Lemma 5.2. If the current strategy, S, is stable, for every k, $2 \le k \le n$, the values of the vertices k and k', with respect to S, are given by the following formulas:

$$V(k) = V(k-1) + S_k V(n) a(k) \text{ and } V(1) = V(0') + S_1 V(n) a(1), \tag{I}$$

$$V(k') = V(k-1) + V(n) a(k),\ V(1') = V(0') + V(n) a(1), \text{ and } V(0') = \frac{1}{2} + \frac{V(n)}{2}, \tag{II}$$

where $a(k)$ is as in Definition 5.1.

Corollary 5.3. If $a(k)$ is as Definition 5.1, then the following holds, for every positive k.

$$a(k+1) = \frac{(-1)^{S_k}}{2} a(k)$$

Corollary 5.4. For every positive k, if the current strategy is stable,

— Vertex k is switchable if and only if either $S_k = 0$, $S_{k_1} = 1$, and $a(k)$ is negative, or $S_k = 1$, $S_{k_0} = 0$, and $a(k)$ is positive.

— Vertex k_i is switchable if and only if either $S_{k_i} = 0$, $S_k = 1-i$, and $a(k)$ is negative, or $S_{k_i} = 1$, $S_k = 1-i$, and $a(k)$ is positive, for i equal to 0 or 1.

Proof: The first statement can be proved as Corollary 2.4 was proved. Note the additional conditions that have been added for this corollary; if S_{k_1} were not equal to 1, vertex k would not be switchable from 0 to 1, because this would result in value 1 for vertices k and k_1 (same for S_{k_0} and vertex k_0).

Proof of the second statement: We present the proof for i equal to 0 and S_{k_0} equal to 0; the rest can be proved in a very similar way.

First, suppose that k_0 is switchable from 0 to 1. S_k should obviously be equal to 1 (otherwise the switch will result in value 1 for vertices k and k_0). Since k_0 is switchable from 0 to 1, its value will be improved by the formula given in the third property of the gadget (note that l is equal to $2n+2$ and m is equal to $2(n-k)$ in the structure). Hence, the following quantity is positive.

$$2^{-2(n-k)}(V(k-1) - V(k)) - 2^{-2n-2}(1 - V(k)) > 0 \iff$$

$$-2^{2k}V(n)a(k) - 2^{-2}(1 - V(k)) > 0 \text{ (by Lemma 5.2 (I), and } S_k = 1) \implies a(k) < 0.$$

Next, suppose that $a(k)$ is negative and S_k is equal to 1. We will prove that if S_{k_0} is 0, vertex k_0 is switchable to edge 1. It suffices to prove that

$$-2^{2k}V(n)a(k) - 2^{-2}(1 - V(k)) > 0 \text{ or } -2^{2k}V(n)a(k) > \frac{1}{4}$$

which is true, since $a(k)$ is equal to $-\frac{1}{2^k}$ (Corollary 5.3), and $V(n) \geq 1/2$. □

Corollary 5.5. Suppose the current strategy, S, is stable, and the improvement policy iteration algorithm reaches a new stable strategy, S', by switching only vertices with numbers greater than k (for any positive k). Then, for any $j \leq k$, j, j_0 or j_1 is switchable with respect to S' if and only if it were switchable with respect to S.

Proof: Since the switchability of vertex j, j_0 or j_1 depends only on the values of S_j, S_{j_0}, S_{j_1}, and $a(j)$, and these are not affected by switches of larger numbered vertices (by Corollary 5.4), the corollary holds. □

Lemma 5.6. If the current strategy is stable, $S_n \cdots S_{k+2}S_{k+1} = 0...01$, $S_{n_1} \cdots S_{(k+2)_1} = 0...0$, $S_{(k+1)_0} = 1$, and vertex k is switched (for any positive k less than n), then the vertices $(k+1)_0$, $(k+2)_1$, ..., n_1 are switchable.

Proof: First suppose that vertex k is switched from 0 to 1. By Corollary 5.4, $a(k)$ is negative, and by Corollary 5.3 $a(k+1)$ is positive. Therefore, by Corollary 5.4, vertex $(k+1)_0$ is switchable. By Corollary 5.3 $a(k+2)$ is negative, so, by Corollary 5.4 vertex $(k+2)_1$ is switchable. It can be proved by induction that the vertices $(k+3)_1$, ..., n_1 are switchable too. If vertex k is switched from 1 to 0, it can be proved in exactly the same way that the vertices $(k+1)_0$, $(k+2)_1$, ..., n_1 are switchable. □

The next lemma is a new one; it gives a relation between the amounts by which the values of the vertices are improved when switched.

Lemma 5.7. Suppose that the current strategy, S, is stable, and that vertices k_x, p_y (or p) are both switchable (with respect to S), where x, y in $\{0, 1\}$ and k, p are any positive integers such that $k < p$. Then the improvement of vertex k_x, if switched, is less than that of vertex p_y (or p).

Proof: We first prove the statement for p_y. From Lemma 5.2 and Corollary 5.3, it follows that, with respect to any stable strategy, $|V(j) - V(j-1)| = 2^{-j}V(n)$ or 0 for every $j > 1$, and $|V(1) - V(0')| = 2^{-1}V(n)$ or 0. Similarly, $|V(j) - V(j')| = 2^{-j}V(n)$ or 0 for every positive j. Therefore, from the properties of the gadget, the maximum improvement of k_x is

$$2^{2(k-n)}\, 2^{-k}V(n) + 2^{-2n-2}(1 - V(k)) \leq 2^{-2n+k}V(n) + 2^{-2n-2}.$$

Similarly, the minimum improvement of p_y is

$$2^{2(p-n)}\, 2^{-p}V(n) - 2^{-2n-2}(1 - V(p)) \geq 2^{-2n+p}V(n) - 2^{-2n-2}.$$

Hence, the minimum difference between the improvement of p_y and that of k_x is

$$2^{-2n}(2^p - 2^k)V(n) - 2^{-2n-1} > 0 \quad (\text{since } p > k,\ k \geq 1, \text{ and } V(n) \geq 1/2).$$

The statement for vertex p can be proved in a similar way. The improvement from switching p is $2^{-p}V(n)$, so the minimum difference between the improvement of p and that of k_x is

$$2^{-p}V(n) - 2^{-2n+k}V(n) - 2^{-2n-2} > 2^{-p}V(n) - 2^{-2n+k}V(n) > 0 \quad (\text{since } p > k \text{ and } p \leq n). \ \square$$

Lemma 5.8. If the current strategy is stable, the following two statements hold (for every positive k):

— If $S_n \cdots S_{k+1}S_k = S_{n_0} \cdots S_{k+1_0}S_{k_0} = S_{n_1} \cdots S_{k+1_1}S_{k_1} = 0...01$, the vertices k_0, $(k+1)_1$, .., n_1 are switchable, and none of the vertices 1, ..., $k-1$ is switchable, the next $3(2^{n-k+1}-1)$ switches of the improvement policy iteration algorithm are made on the vertices numbered n, $n-1$, ..., k, to reach the stable strategy where $S_n \cdots S_{k+1}S_k = S_{n_0} \cdots S_{k+1_0}S_{k_0} = S_{n_1} \cdots S_{k+1_1}S_{k_1} = 0...00$, and none of these vertices is switchable.

— If $S_n \cdots S_{k+1}S_k = S_{n_0} \cdots S_{k+1_0}S_{k_0} = S_{n_1} \cdots S_{k+1_1}S_{k_1} = 0...00$, the vertices k_1, $(k+1)_1$, .., n_1 are switchable, and none of the vertices 1, ..., $k-1$ is switchable, the next $3(2^{n-k+1}-1)$ switches of the improvement policy iteration algorithm are made on the vertices numbered n, $n-1$, ..., k, to reach the stable strategy where $S_n \cdots S_{k+1}S_k = S_{n_0} \cdots S_{k+1_0}S_{k_0} = S_{n_1} \cdots S_{k+1_1}S_{k_1} = 0...01$, and none of these vertices is switchable.

Proof: The lemma will be proved by induction on k.

Basis Case: The basis case of the first statement can be proved as follows (the basis case of the second statement can be proved in the same way). If k is equal to n, vertex n_0 is switchable, and therefore, vertices n, n_1 are not (Corollary 5.4). Vertex n_0 is switched (by Lemma 5.7) from 1 to 0. By Corollary 5.4, vertex n is now switchable, and it is switched next (by Lemma 5.7) from 1 to 0, which results in vertex n_1

becoming switchable from 1 to 0. By Lemma 5.7 it is switched too, before any other switch. If the initial strategy were stable, the current one is too. The total is 3 switches.

Induction Hypothesis: The statements hold for every $k \geq m$.

Induction Step: We will prove that the statements hold for k equal to $m-1$. We prove the first one (the second one can be proved in the same way).

Since vertex $(m-1)_0$ is switchable, by Corollary 5.4, the vertices $m-1$ and $(m-1)_1$ are not switchable. Therefore all the conditions of the second statement for k equal to m hold. The algorithm will make the next $3(2^{n-m+1}-1)$ switches on the vertices numbered n, $n-1$, ..., m, to reach the stable strategy where $S_n \cdots S_{m+1} S_m = S_{n_0} \cdots S_{m+1_0} S_{m_0} = S_{n_1} \cdots S_{m+1_1} S_{m_1} = 0...01$, and none of these vertices is switchable. Vertex $(m-1)_0$ is still switchable (by Corollary 5.5), and it is switched next from 1 to 0 (Lemma 5.7). Now vertex $m-1$ is switchable from 1 to 0 (Corollary 5.4), and it is switched (Lemma 5.7). By Lemma 5.6, the vertices m_0, $(m+1)_1$, ..., n_1 are switchable now. All the conditions of the first statement for k equal to m hold (note that the strategy is stable), so the next $3(2^{n-m+1}-1)$ switches of the algorithm are made on the vertices numbered n, $n-1$, ..., m, to reach the stable strategy where $S_n \cdots S_{m+1} S_m = S_{n_0} \cdots S_{m+1_0} S_{m_0} = S_{n_1} \cdots S_{m+1_1} S_{m_1} = 0...00$, and none of these vertices is switchable. After the switch of vertex $m-1$, vertex $(m-1)_1$ became switchable from 1 to 0 (Corollary 5.4), and it is still switchable (Corollary 5.5), so it is switched (Lemma 5.7). The strategy reached is stable, and $S_n \cdots S_{m-1} = S_{n_0} \cdots S_{(m-1)_0} = S_{n_1} \cdots S_{(m-1)_1} = 0...00$. The total number of switches made is $3(2^{n-m+2} - 1)$. Note also that the leftmost $3(n-m+2)$ vertices are not switchable. □

We can now present the main result of the section.

Lemma 5.9. The improvement policy iteration algorithm requires exponential time in the worst case. For every positive n, the algorithm makes $3(2^n-1)$ switches, on the graph G_n of the structure described, to find the best strategy (vector 00...01) if the initial strategy is the 0-vector.

Proof: It follows from the previous results, and particularly from the second statement of Lemma 5.8 (for k equal to 1), since the initial strategy is the 0-vector, which is stable. Also, $a(k)$ is negative for every k, so the vertices $1_1, \ldots, n_1$ are all initially switchable (by Corollary 5.4). □

6. Conclusion and Open Problems

We have studied the complexity of the policy iteration algorithm for simple stochastic games and have shown that many natural variations of this algorithm require exponential time in the worst case. The results of this paper indicate that it is unlikely that an approach based on the policy iteration algorithm will yield a polynomial time algorithm for the SSG value problem. Thus, it is still unresolved whether the SSG value problem is in P.

Apparently, no probabilistic analysis of the policy iteration algorithm has been undertaken, motivating the following questions. Suppose we define a new algorithm, the **Randomized Policy Iteration**

Algorithm, by defining the select procedure to choose a switchable vertex randomly and uniformly from the set of switchable vertices. Does the randomized policy iteration algorithm run in expected polynomial time on all inputs? A positive answer to this question would imply that the SSG value problem is in the complexity class ZPP, that is, it has an errorless, polynomial time algorithm. (See Gill, [3], for a formal definition of the class ZPP.) A negative answer would shed more insight on the limitations of policy iteration algorithms in general. For all of the constructions in this paper, the randomized policy iteration algorithm requires only polynomial expected time (we leave it to the reader to verify that this is the case).

It would also be interesting to obtain results on the average case performance of the policy iteration algorithm, on reasonable distributions of instances of the SSG problem. The work of Tovey, [12], may be a useful start in this direction.

There are other algorithms for the SSG value problem, whose performance has not been analyzed. One such algorithm is derived from the fact, mentioned in Section 1, that if a SSG has only min and average, or max and average, vertices, a locally optimal strategy can be found in polynomial time using linear programming. In the same way, suppose that in a SSG with min, max and average vertices, the strategy of one player is fixed, say strategy S_1 of player 1. Then a strategy S_0 of player 0 that is locally optimal with respect to S_1 can be found in polynomial time. That is, for each min vertex i, the value of vertex i, with respect to strategies S_0 and S_1, is the minimum of the values of its children.

Thus the following algorithm is guaranteed to find a locally optimal pair of strategies. Initially, choose an arbitrary strategy S_1 for player 1. Then repeatedly do the following: find a locally optimal strategy, S_0 for player 0 with respect to S_1. Now, find a new strategy S'_1 for player 1, such that S'_1 is locally optimal with respect to S_0. Halt if the current pair of strategies S'_1, S_0 is a locally optimal pair of strategies; otherwise repeat, letting the new value of S_1 equal the current value of S'_1. In the worst case, how many iterations of this procedure are necessary to reach a locally optimal pair of strategies?

A new framework for the study of local search problems has been proposed by Johnson, Papadimitriou and Yannakakis, [6]. They define a new complexity class called PLS, for polynomial local search. This complexity class contains optimization problems with finitely many solutions, where the goal is to find a locally optimal solution. The problem of finding a locally optimal pair of strategies for a SSG is in the class PLS. PLS-complete problems include a number of local search versions of NP-complete problems, such as finding a locally optimal tour for the traveling salesman problem, under the Lin-Kernighan heuristic (Papadimitriou, Schäffer and Yannakakis, [9]). Is the problem of finding a locally optimal pair of strategies for a SSG PLS-complete?

References

[1] A. Condon. The Complexity of Stochastic Games. *Information and Computation*, to appear, 1990. Also available as Technical Report Number 863, Computer Sciences Department, University of Wisconsin-Madison.

[2] C. Derman. *Finite State Markov Decision Processes.* Academic Press, 1972.

[3] J. Gill. The Computational Complexity of Probabilistic Turing Machines. *SIAM Journal on Computing*, 6:675-695, 1977.

[4] Howard. *Dynamic Programming and Markov Processes.* M.I.T. Press, 1960.

[5] R. J. Jeroslow. The Simplex Algorithm with the Pivot Rule of Maximizing Criterion Improvement. *Discrete Math.*, 4:367-378, 1973.

[6] D. S. Johnson, C. H. Papadimitriou and M. Yannakakis. How Easy is Local Search? *Journal on Computer and System Sciences*, 37:79-100, 1988.

[7] L. G. Khachiyan. A Polynomial algorithm in linear programming. *Soviet Math Dokl.*, 20:191-194, 1979.

[8] V. Klee and G. Minty. How Good is the Simplex Algorithm? *Inequalities III*, O. Shisha, Academic Press, New York, 159-175, 1979.

[9] C. H. Papadimitriou, A. A. Schäffer and M. Yannakakis. On the Complexity of Local Search. *Proceedings of the 22nd Annual Symposium on the Theory of Computing (STOC)*, 438-445, 1990.

[10] H. J. M. Peters and O. J. Vrieze. *Surveys in game theory and related topics, CWI Tract 39.* Centrum voor Wiskunde en Informatica, Amsterdam, 1987.

[11] L. S. Shapley. Stochastic Games. *Proceedings of the National Academy of Sciences, U.S.A*, 39: 1095-1100, 1953.

[12] C. A. Tovey. Low Order Polynomial Bounds on the Expected Performance of Local Improvement Algorithms. *Mathematical Programming*, 35(2): 193-224, 1986.