Fundamenta Informaticae XXII (2002) 1001–1012
IOS Press

Relaxed Update and Partition Network Games

Hans L. Bodlaender
Institute of Information and Computing Sciences
Utrecht University P.O. Box 80.089, 3508 TB
Utrecht, the Netherlands
hansb@cs.uu.nl

Michael J. Dinneen [C]
Department of Computer Science
University of Auckland, Private Bag 92019
Auckland, New Zealand
mjd@cs.auckland.ac.nz

Bakhadyr Khoussainov
Department of Computer Science
University of Auckland, Private Bag 92019
Auckland, New Zealand
bmk@cs.auckland.ac.nz

Abstract. In this paper, we study the complexity of deciding which player has a winning strategy in certain types of McNaughton games. These graph games can be used as models for computational problems and processes of infinite duration. We consider the cases (1) where the first player wins when vertices in a specified set are visited infinitely often and vertices in another specified set are visited finitely often, (2) where the first player wins when exactly those vertices in one of a number of specified disjoint sets are visited infinitely often, and (3) a generalization of these first two cases. We give polynomial time algorithms to determine which player has a winning strategy in each of the games considered.

Keywords: graph and network algorithms, complexity, infinite graph games, McNaughton games.

[C]Corresponding author

1. Introduction and Basic Definitions

Motivated by the work of Gurevich and Harrington [3], McNaughton [5] introduced a type of infinite games played on finite graphs. These games can be used as models for certain computational problems and can provide game-theoretic foundations for studying infinite duration processes such as operating systems, networks, communication systems and concurrent computations. For example, Nerode *et al.* [7, 6] introduce the idea of investigating and identifying distributed concurrent programs as strategies in Gurevich-Harrington and McNaughton type of games. We also mention a related paper [4] that uses a modal logic version of these games as a model for problems in control theory.

Assume we have an infinite duration system. A run of the system can be thought as an infinite sequence $s_0, s_1, s_2, s_3, \ldots$ of states. The state s_0 is the initial state. The state s_{i+1} is obtained by the execution of a certain command at s_i. The success of the run depends on whether or not the run satisfies certain specifications given by (or inherited from) software or hardware of the system. One can look at this run as a play between two players, *Survivor* and *Adversary*. The goal of one of the players, say *Survivor*, is to satisfy the specifications, while the goal of the opponent (in this case *Adversary*) is not to allow the specifications to be satisfied. During the play there is no termination point. Instead there are some special events that may happen continually. If some combination of these events happens infinitely often then one player wins, otherwise the other player wins. We now formalize these games, as was first given in [5].

Definition 1.1. A **game** $\mathcal{G}$ is a seven tuple $(V, S, A, E, v_0, W, \Omega)$, where:

1. V is the set of nodes called **positions**.
2. S and A are subsets of V such that $S \cap A = \emptyset$ and $S \cup A = V$. The nodes of S are **positions** of *Survivor*, and the nodes of A are **positions** of *Adversary*.
3. $E \subseteq S \times A \cup A \times S$ is a set of directed edges between S and A such that
   (a) for each $s \in S$ there exists at least one $a \in A$ with $(s, a) \in E$, and
   (b) for each $a \in A$ there exists at least one $s \in S$ with $(a, s) \in E$.
4. v_0 is the initial position of the game.
5. W is a subset of V called the set of **special positions**.
6. Finally, Ω is a set of some subsets of W. These are called **winning sets** or **winning conditions** for *Survivor*.

For the game $\mathcal{G}$ the **graph of the game** is the graph $(A \cup S, E)$. All plays of $\mathcal{G}$ occur in the graph of the game. To visualize a play we describe it informally as follows. There is a placemarker, that is initially placed on node v_0. At any given time the placemarker is placed on a node. If the node is in S, then it is *Survivor*'s turn to move the placemarker. Otherwise it is *Adversary*'s turn. The placemarker is always moved along the edges of the game graph determined by E. There is always a possibility to move the placemarker as stipulated by conditions $3a)$ and $3b)$ of the definition.

Let s_0 be a position, say of *Survivor*. Assume that *Survivor* begins its move by putting the placemarker on a_0 (so $(s_0, a_0) \in E$). *Adversary* responds by putting the placemarker on a s_1 (so $(a_0, s_1) \in E$). This procedure repeats and the players' actions produce an infinite sequence:

$$\mathbf{p} = s_0, a_0, s_1, a_1, \ldots$$

called a **play** that begins from position s_0. In the play $\mathbf{p}$ consider the set of all nodes t that have the following properties:

1. t belongs to W, and
2. t occurs in the play $\mathbf{p}$ infinitely often.

We denote this set by $In(\mathbf{p})$ and call it the **infinity set of $\mathbf{p}$**. *Survivor* wins the play if $In(\mathbf{p}) \in \Omega$. Otherwise *Adversary* wins the play. Thus, every play is won by one of the players.

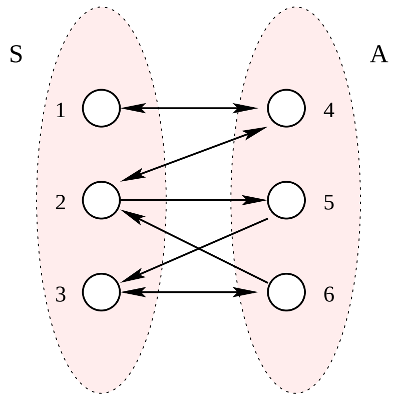
The **histories** of the play $\mathbf{p} = q_0, q_1, q_2, \ldots$ are the finite prefixes of $\mathbf{p}$. The set $H(S)$ consists of all histories whose last positions are positions where *Survivor* makes move. The set $H(A)$ is defined similarly. A **strategy** for *Survivor* is a function f that maps $H(S)$ into A such that for all $u = q_0 \ldots q_n \in H(S)$, $(q_n, f(u)) \in E$. A **strategy** for *Adversary* is defined similarly.

Let f be a strategy for a player. Let q be a position in the game. Consider all the plays that begin from q which are played when the player follows the strategy f. We call these **plays consistent with f from q**.

Definition 1.2. The strategy f of a player is a **winning strategy** if all plays consistent with f from v_0 are won by the player. In this case we say that the player **wins the game**.

McNaughton [5] proved that for every McNaughton game, it is decidable who has a winning strategy. However, his algorithm is by no means an efficient one. Thus, it is natural to ask for which type of McNaughton games it can be decided in polynomial time which player has a winning strategy. Some polynomial time solvable instance were given by Dinneen and Khoussainov in [2] and Nerode *e*t al. in [6]. In [2] games with $W = V$ and $\Omega = \{V\}$, called **update network games** are studied and it is shown that there is an $O(|V||E|)$ time algorithm to determine if *S*urvivor wins these games. In this paper, we extend this result. First, we consider for networks with a partition of the set of nodes into three sets $V = I \cup F \cup D$, games of the form $(V, S, A, E, v_0, I \cup F, \{I\})$. I.e., *Survivor* wins if every node in I is visited infinitely often, and every node in F is visited finitely often. Thus, each play in such games is indifferent whether or not the nodes in $D = V \setminus W$ are visited finitely or infinitely often. Therefore we call the nodes D **don't care nodes**. We provide a $O(|V||E|)$ time algorithm to decide which player has a winning strategy in such games. Secondly, we consider the games where $W = V$ and Ω is a collection of pairwise disjoint winning sets. We show that there exists a polynomial time algorithm to decide who wins such games. Finally, we combine these results, and allow W to be a proper subset of V, with Ω a collection of pairwise disjoint non-empty winning sets.

Example 1.1. We now illustrate two of these network games with the following bipartite graph and $v_0 = 1$.



- With $I = \{1, 4\}$, $F = \{3, 5, 6\}$ and $D = \{2\}$, we have a relaxed update game where Survior's winning strategy is achievable by avoiding vertex 5.
- With $W = V$ partitioned into the set $\Omega = \{\{1, 4\}, \{2, 3, 5, 6\}\}$, we have a game where Adversary wins by avoiding both vertices 1 and 2, as appropriate.

2. Preliminary Results

Given a McNaughton game $\mathcal{G}$ and a subset of the nodes $X \subseteq V$, a node v is in the set REACH(S, X) if *Survivor* can force every play starting at v into a node in X after a finite number of steps. Note that REACH$(S, \emptyset)$ is assumed to be $\emptyset$ which is consistent with the definition.

Lemma 2.1. The set REACH(S, X) can be computed in $O(|V| + |E|)$ time.

Proof:
We build a set R, that will eventually be REACH(S, V). Initially, we take $R = X$. If a node x, owned by *Survivor*, has an edge to a node in R, then x is added to R. If a node x, owned by player *Adversary*, has only edges to nodes in R, then x is added to R. One can note that from every node in R *Survivor* can always force a play to go to a node in X. Moreover, when no nodes can be added to R anymore, then $R =$ REACH(S, X). *Adversary* has a strategy such that only nodes in $V \setminus R$ are visited. Indeed, *Adversary* has a strategy to always stay inside of $V \setminus R$ when game begins in a node from $V \setminus R$. The procedure of constructing REACH(S, X) can be implemented in $O(|V|+|E|)$ time, by giving each node not in X a counter, that is initially 1 for nodes owned by *Survivor* and its outdegree for nodes owned by *Adversary*. Whenever we add a node v to R, we subtract 1 from the counters of each node with an edge to v; when a counter becomes 0 then the node is also added to R. □

Let $v \notin$ REACH(S, X) be an *Adversary*'s node. We iteratively define the set AVOID(v, A, X) as follows. Initially, we take AVOID(v, A, X)=$\{v\}$. If a node x is owned by *Adversary* and $x \in$ AVOID(v, A, X) then we add a neighbor y into AVOID(v, A, X) if $(x, y) \in E$ and $y \notin$ REACH(S, X). If a node x is owned by *Survivor* and $x \in$ AVOID(v, A, X) then we add all y into AVOID(v, A, X) for which $(x, y) \in E$.

From Lemma 2.1 we obtain the following lemma.

Lemma 2.2. Given X and $v \notin$ REACH(S, X) the set AVOID(v, A, X) has the following properties:

1. The set AVOID(v, A, X) can be constructed in $O(|V| + |E|)$ time.
2. AVOID$(v, A, X) \cap$ REACH$(S, X) = \emptyset$.
3. *Adversary* has a strategy, such that when the game visits a node in AVOID(v, A, X) then all nodes visited afterwards also belong to AVOID(v, A, X).
4. For all s in AVOID$(v, A, X) \cap S$ and all $a \in A$ if $(s, a) \in E$ then a is in AVOID(v, A, X).

If the game starts at v, then a strategy for *Adversary* not to play to a node in X is to always play to a node in AVOID(v, A, X). Note that the sets REACH(A, X) and AVOID(v, S, X) can be defined in a similar matter. The two lemmas above hold true for these sets too.

3. Relaxed Update Networks

In [2] the games where $W = V$ and $\Omega = \{V\}$ are studied. These games are called **update network games**. An update network game is an **update network** if *Survivor* wins the game. We generalize these games in the following definition.

Definition 3.1. A game $\mathcal{G}$ is **relaxed update network game** if Ω consists of a fixed subset I of W. We say that a relaxed update network game from a position q is a **relaxed update network** if *Survivor* has a winning strategy from q.

Thus, in a relaxed update network the set of nodes is partitioned into three sets $V = I \cup F \cup D$, where I is a given subset of W, $F = W \setminus I$, and $D = V \setminus W$. *Survivor* wins a play if every node in I is visited infinitely often, and every node in F is visited finitely often. Thus, each play in such games is indifferent whether or not the nodes in D are visited finitely or infinitely often. Therefore we can call the nodes in D **don't care nodes**.

3.1. The Case $I = \emptyset$

Let $\mathcal{G}$ be a relaxed update game. Here we consider the case that $I = \emptyset$, i.e., we have nodes that must be visited only finitely often (F) and don't care nodes. Of course, the problem is trivial when $F = \emptyset$ and $I = \emptyset$. So we assume that $F \neq \emptyset$.

Let $V_0 = V \setminus$ REACH(A, F). If V_0 is empty, then *Adversary* has a winning strategy: from every node, *Adversary* has a forced play into a node in F. Thus, *Adversary* can force some of the nodes in F to be visited infinitely often.

If *Survivor* begins the game from a node v in V_0, then *Survivor* has a winning strategy: he plays always inside the set AVOID(v, S, F) which is possible by Lemma 2.2.

If neither V_0 is empty nor the game starts at a node in V_0, then we start with an iterative process. In order to describe the process we make the following notes.

Consider REACH(S, V_0). Note that for each node in REACH(S, V_0), *Survivor* has a winning strategy when the game starts at that node. *Survivor* can force all plays from the node into V_0. When a node in V_0 is reached *Survivor* has a strategy such that no node in F is visited anymore. Thus, *Adversary* should not play into a node in REACH(S, V_0). In particular, *Adversary* should not play to nodes in $F \cap \text{REACH}(S, V_0)$.

Let us consider $F_1 = F \setminus \text{REACH}(S, V_0)$ and $V_1 = V \setminus \text{REACH}(A, F_1)$. Note that $V_0 \subseteq V_1$. *Survivor* has a winning strategy when the game starts at a node of V_1. *Survivor* can always play inside V_1 again by Lemma 2.2, and hence no nodes in F_1 are visited. So the only nodes in F *Adversary* can possibly direct the plays to are those in REACH(S, V_0). But from these nodes *Survivor* can force all plays into V_0. Hence nodes in F are visited in total a finite number of times.

Thus, when the game starts at a node in V_1 we are done: *Survivor* has a winning strategy. When $V_1 = V_0$ and the game starts at a node in $V \setminus V_1$, then we are also done as *Adversary* has a winning strategy; *Adversary* always forces all the plays into $F_1 = F$ staying in $V \setminus V_1$.

The step above can be repeated which leads us to an iterative procedure. Thus, let $F_0 = F$ and $V_0 = V \setminus \text{REACH}(A, F_0)$. For each $i \geq 1$, let

$$F_i = F_{i-1} \setminus \text{REACH}(S, V_{i-1}) \quad \text{and} \quad V_i = V \setminus \text{REACH}(A, F_i).$$

With arguments similar as above, we can show that *Survivor* has a winning strategy in all nodes in V_i.

As each $V_i \subseteq V_{i+1}$, the process stops when we have an i with $V_i = V_{i+1}$. In that case, there is a winning strategy for *Survivor* if and only if the game starts at a node in V_i. Suppose the game starts at a node in $V \setminus V_i$ and $V_i = V_{i+1}$. Then, *Adversary* can force a play to a vertex in F_{i+1}; and either it is owned by *Survivor* and has all outgoing edges to a vertex in $V \setminus V_i$ or is owned by *Adversary* and has one outgoing edge to $V \setminus V_i$, (as follows from Lemma 2.2), hence *Adversary* can force the game to stay in $V \setminus V_i$.

This gives a polynomial time algorithm for the problem with $I = \emptyset$. The algorithm takes $O(|V||E|)$ time, as there are $O(|V|)$ iterations, each taking $O(|V| + |E|)$ time.

3.2. Reducing to the Case $F = \emptyset$

Now assume $I \neq \emptyset$. In this section, we show that an instance with $F \neq \emptyset$ can be transformed to an equivalent instance with $F = \emptyset$, assuming $I \neq \emptyset$.

We may assume that the initial position belongs to REACH(S, I); if not, then clearly *Adversary* has a winning strategy from Lemma 2.2. (*Adversary* forces that no node in I is ever reached.) Now, *Survivor* can start the game by forcing to go to any node in I, and, as all nodes in I have to be visited infinitely often, it is not important for the analysis to which node in I the game goes first.

There are two cases:

REACH$(A, F) \cap I \neq \emptyset$. This means that there is a node $i \in I$, such that *Adversary* has a strategy that forces all the plays of the game (consistent with the strategy) to visit a node in F after a finite number of steps. If this is the case, then *Adversary* has a winning strategy for the game. Here either i is not visited infinitely often, or he can force after every visit to i a play to a node in F, in which case at least one node in F is visited infinitely often.

REACH$(A, F) \cap I = \emptyset$. This means that for all nodes $i \in I$, *Adversary* can not force any play of the game visit a node in F. Therefore if *Survivor* has a winning strategy then he has one that prevents movement to a node in F after the first node in I has been reached.

Once a node in I has been reached, *Survivor* wants to avoid the plays reaching nodes in F. (Any play to a node in F now could possibly be repeated by *Adversary*.) So if *Survivor* can avoid reaching a node in F infinitely many times, he can avoid visiting it once.

So, what we can do is compute REACH(A, F), and remove all nodes in REACH(A, F) from the graph, and obtain an equivalent instance, but now with $F = \emptyset$.

3.3. Case with Infinite-Visit Nodes

In this section, we consider the game with $F = \emptyset$ and $I \neq \emptyset$. Suppose the game starts at node v_0.

Lemma 3.1. There is a winning strategy for *Survivor* if and only if $v_0 \in$ REACH(S, I) and $I \subseteq$ REACH$(S, \{v\})$ for all $v \in I$.

Proof:
Suppose $w \in I$, $w \notin$ REACH$(S, \{v\})$. Then *Adversary* has a winning strategy. If w is never visited in the game, then *Survivor* loses. If w is visited, then after w has been visited, *Adversary* has a strategy that avoids v, so *Adversary* again wins.

If $v_0 \notin$ REACH(S, I), then *Adversary* can prevent any node in I to be visited as follows from Lemma 2.2.

Now suppose for all $v \in I$, $I \subseteq$ REACH$(S, \{v\})$, and $v_0 \in$ REACH(S, I). The latter condition makes that *Survivor* can start by forcing all plays from v_0 into I. The former condition means that for every pair of nodes $v, w \in I$, *Survivor* has a strategy that forces, after w has been visited, that in a finite number of moves v will be visited. This enables *Survivor* to force that every vertex in I to be visited infinitely often. □

The condition of Lemma 3.1 can be checked in $O(|V||E|)$ time. Thus we have proved the following theorem.

Theorem 3.1. There is a $O(|V||E|)$ time algorithm to decide whether a given game is a relaxed update network.

3.4. P-Completeness

The previous section shows that we can decide in polynomial time if a relaxed update game is a relaxed update network. Let's call this the RELAXEDNETWORK problem. Here we show that this problem is P-complete and hence any decision algorithm for it is inherently sequential.

Proposition 3.1. The RELAXEDNETWORK problem is P-complete.

Proof:
All that remains to see is that RELAXEDNETWORK is log-space hard for the complexity class P. To do this, we reduce the AGAP (And/Or Graph Accessibility Problem), which is known to be P-complete [1], to RELAXEDNETWORK. An instance of AGAP is an and/or graph $D = (V, A)$ with two vertices s and t. The problem is to decide whether t is reachable from s. We say that t is reachable from s in an and/or graph $D = (V, A)$ if a pebble can be placed on the specified vertex t by using the following rules:

1. We can place a pebble on s.
2. For an AND vertex v, a pebble can be placed if all in-neighbors of v are pebbled.

3. For an OR vertex v, a pebble can be placed if at least one in-neighbor is pebbled.

We can transform this instance into an instance of RELAXEDNETWORK as follows. We map an instance (D, s, t) of AGAP into a game instance $(V, S, A, E, v_0, I \cup F, \{I\})$. First let D' be a bipartite version of D where we subdivide any arcs with two end-points of the same type (the new vertex is the opposite type). Then declare $V = V(D') \cup \{t'\}$, S equal to the OR vertices, A equal to the AND vertices, $E = E(D') \cup \{(t,t'), (t',t)\} \setminus \{(t,v) \mid v \in V(D')\}$, $v_0 = s$, $I = \{t, t'\}$, $F = V \setminus I$. There is a pebbled path from s to t in D if and only if Survivor wins the game defined. This transformation is clearly doable in log space. □

3.5. A Dual Case

This case is obtained when we interchange the players of games. Let us consider the case when $I = \emptyset$ in a relaxed update game and interchange the roles of the players. Thus, now *Survivor*'s winning conditions are nonempty subsets of F. Then Subsection 3.1 can be explained as follows. Consider the following sequence:

$$F_0 = \text{REACH}(S, F),\ F_{i+1} = \{x \mid x \in \text{REACH}(S, F_i \setminus \{x\}) \text{ and } x \in F_i\}.$$

The iteration guarantees that F_i consists of all nodes from which *Survivor* can visit the set F at least $i+1$ times. Note that $F_{i+1} \subseteq F_i$ for all i. Let i be such that $F_i = F_{i+1}$. We can show that *Survivor* wins the game from v if and only if $v \in \text{REACH}(S, F_i)$. The proof is basically given in Subsection 3.1.

4. Partition Games and Partition Networks

In this section we study games where winning conditions are pairwise disjoint collections of nonempty sets with $W = V$. Formally, a **partition network game** is a game $\mathcal{G}$ of the form $(V, S, A, E, v_0, V, \{W_1, \ldots, W_n\})$, where $W_1, \ldots, W_n$ is a collection of pairwise disjoint nonempty winning sets. We say that a partition network game is a **partition network** if *Survivor* is the winner of the game. An important concept of closed winning conditions (sets) is defined as follows:

Definition 4.1. A winning condition W_i in a game $\mathcal{G}$ is **S-closed** if the following two conditions are satisfied:

1. For any *Survivor*'s position $s \in W_i$ there exists an a such that $(s, a) \in E$ and $a \in W_i$.

2. For any *Adversary*'s position $a \in W_i$ and all s such that $(a, s) \in E$ we have $a \in W_i$.

Informally, if W_i is a closed winning set then *Survivor* can always stay inside of W_i no matter what the opponent does. The next lemma gives a necessary condition for *Survivor* to win a partition network game.

Lemma 4.1. If *Survivor* wins the partition network game $\mathcal{G}$ then one of the winning conditions must be S-closed.

Proof:
Suppose that each W_i is not S-closed. Then for each W_i one of the following cases hold:

1. There exists a *Survivor*'s node $s_i \in W_i$ so that all the outgoing edges from s lead to nodes outside of W_i.

2. There exists an *Adversary*'s node $a_i \in W_i$ such that $(a_i, s_i) \in E$ and $s_i \notin W_i$.

We construct the following strategy g for *Adversary*. For all *Adversary*'s positions a if $a = a_i$ then $g(a) = s_i$; in all other cases $g(a)$ is the first node s for which $(a, s) \in E$. We claim that g is a winning strategy for *Adversary* thus contradicting the assumption. Indeed let $\mathbf{p} = p_0, p_1, p_2, \ldots$ be a play consistent with g. Consider the infinity set $In(\mathbf{p})$. Assume that $In(\mathbf{p}) = W_i$. Then from some stage m in the play all nodes from W_i and only those will appear infinitely often. Therefore W_i does not satisfy the first case listed above. Hence for W_i there exists an *Adversary*'s node $a_i \in W_i$ such that $(a_i, s_i) \in E$ and $s_i \notin W_i$. From the definition of g, as a_i must appear in $\mathbf{p}$ after point m, we see that $\mathbf{p}$ must contain a position from outside of W_i after stage m. This contradicts the choice of m. Therefore $In(\mathbf{p}) \neq W_i$ for all winning sets W_i. □

For our next lemma we need the following concept. We say that a winning condition W is an **update component** if W is S-closed and *Survivor* wins the update game played in W.

Lemma 4.2. If *Survivor* wins the partition network game $\mathcal{G}$ $(V, S, A, E, v_0, V, \{W_1, \ldots, W_n\})$, then one of the winning conditions is an update component.

Proof:
By the lemma above, one of the winning conditions W_i must be S-closed. Without lost of generality we may assume that $W_1, \ldots, W_k$ are all the S-closed winning conditions among $W_1, \ldots, W_n$, where $k \leq n$.

In order to obtain a contradiction, assume that none of W_1, …, W_k is an update component. Hence for every t with $1 \leq t \leq k$ and every $x \in W_t$, *Adversary* has a winning strategy $g_{t,x}$ to win the update game (W_t, x) from x. Note that for each W_i, $i > k$, one of the following cases hold:

1. There exists a *Survivor*'s node $s_i \in W_i$ so that all the outgoing edges from s lead to nodes outside of W_i.

2. There exists an *Adversary*'s node $a_i \in W_i$ such that $(a_i, s_i) \in E$ and $s_i \notin W_i$.

Now we define the following strategy g for *Adversary*. Let a be an *Adversary*'s position. Consider any finite history $h = p_0, \ldots, p_m$ of a play that begins from v so that $a = p_m$. If $a = a_i$ for some $i > k$ then $g(h) = s_i$. Now assume $a \in W_t$ with $1 \leq t \leq k$. Let p_r be a node in the history so that all $p_r, \ldots, p_m \in W_t$ and $p_{r-1} \notin W_t$. Then $g(h) = g_{t,p_r}(p_r, \ldots, p_m)$. In all other cases, $g(h)$ is the first s with $(a, s) \in E$.

We claim that g is a winning strategy for *Adversary*. Indeed, let $\mathbf{p} = p_0, p_1, p_2, \ldots$ be a play consistent with g. Consider the infinity set $In(\mathbf{p})$. Assume that $In(\mathbf{p}) = W_i$. Then $i \leq t$ which can be proved by using the reasoning similar to the proof of the previous lemma. Assume that $i \leq t$. Let m be the first point in the the play $\mathbf{p}$ so that all nodes from W_i and only those will appear infinitely often. Then g will always follow the strategy g_{i,p_m}. Hence $In(\mathbf{p})$ can not be equal to W_i. Again we have a contradiction. □

From these two lemmas we have the following result.

Corollary 4.1. In a partition network game, if either (1) each winning conditions is not S-closed or (2) each S-closed winning condition does not form an update component then *Adversary* wins the partition game.

Now assume that one of the winning conditions of the partition network game is an update component. Without loss of generality we can assume that it is W_1. Consider the set REACH(S, W_1). If $v_0 \in$ REACH(S, W_1) then *Survivor* clearly wins the game. Otherwise, we define the following game $\mathcal{G}'$:

1. Set $V' =$ AVOID(v_0, A, W_1).
2. For each W_i if $W_i \cap$REACH$(S, W_1) \neq \emptyset$ then W_i is not a winning set of the new game. Otherwise, W_i is a winning set of the new game.
3. The set E' of edges is obtained by restricting E to V'.
4. The initial position of the game is v_0.

Lemma 4.3. Assume that W_1 is an update network component and $v_0 \notin$ REACH(S, W_1). *Survivor* wins the original game if and only if *Survivor* wins the new game $\mathcal{G}'$.

Proof:
Indeed, assume that *Adversary* wins the new game $\mathcal{G}'$. Let g' be winning strategy. Then since g' is inside the AVOID(v_0, A, W_1) strategy, we see that *Adversary* wins the whole game. Assume that *Survivor* wins the new game. Let f' be winning strategy. Define a strategy f as follows. If a play is inside the game $\mathcal{G}'$ then always follow f'. Otherwise, force the place into W_1 and win the update game W_1. It is not hard to see that *Survivor* wins the game. □

We call the game $\mathcal{G}'$ obtained from $\mathcal{G}$ the reduced game at v_0. Now consider the following procedure that for any $x \in V$ proceeds by stages as follows.

Stage 0. Set $\mathcal{G}_0 = \mathcal{G}$.

Stage $i + 1$. Consider $\mathcal{G}_i$. If all of the winning conditions of $\mathcal{G}_i$ are not S-closed or all S-closed winning conditions of $\mathcal{G}_i$ are not update components then declare *Adversary* the winner. Otherwise take the first winning condition W which is an update network component. If $x \in$ REACH(S, W) then *Survivor* is the winner. If not, reduce $\mathcal{G}_i$ to $\mathcal{G}_{i+1}$ at node x.

Note that at some stage k the process stops at which the winner at x is found. The algorithm to decide the game runs in $O(|E||V|^2)$ time yielding:

Theorem 4.1. There is a $O(|V|^2|E|)$ time algorithm to decide whether a given game is a partition network.

5. Relaxed Partition Networks

In this section, we combine the results of Sections 3 and 4. We consider partition games where possibly $W \neq V$. We now have **relaxed partition network games** of the form

$$\mathcal{G} = (V, S, A, E, v_0, W, \{W_1, \ldots, W_n\}),$$

where $W \subseteq V$, and $W_1, \ldots, W_n$ is a collection of pairwise disjoint nonempty winning sets, each a subset of W. Again, the set of don't care nodes is denoted by $D = V \setminus W$.

For sets $X, Y \subseteq V$, $X \cap Y = \emptyset$, define the set $RA(S, X, Y)$ of nodes from which *Survivor* can force a play that reaches, in a finite number of steps, a node in X by avoiding Y. Thus, $v \in RA(S, X, Y)$ if *Survivor* has a winning strategy in the game that starts at a v where *Survivor* wins as soon as a node in X is visited; *Adversary* wins as soon as a node in Y is visited or when infinitely many moves occur without a visit to a node in $X \cup Y$.

Lemma 5.1. Given X, Y, $X \cap Y = \emptyset$, $RA(X, Y)$ can can be computed in $O(|V||E|)$ time.

Proof:
The set $RA(S, X, Y)$ can be computed as follows. Initially, set $R = X$. If a node $s \in S \setminus Y$ has an edge $(s, a) \in E$ and $a \in R$, then add s to R. If a node $a \in A \setminus Y$ has for all s with $(a, s) \in E$, $s \in R$, then add s to R. Repeat this process until we cannot add nodes to R using these rules. One easily sees with induction that $R \subseteq RA(S, X, Y)$. We also have, after no further nodes can be added to R, that $R = RA(S, X, Y)$; any Adversary node in $V \setminus R \setminus Y$ has an edge to a node in $V \setminus R$, and any Survivor node in $V \setminus R \setminus Y$ has only edges to nodes in $V \setminus R$. Thus, when *Adversary* follows a strategy to always play to nodes in $V \setminus R$, he wins either by having the game moved to a node in Y, or by an infinite play. Finally, use the same data structure as in Lemma 2.1. □

Definition 5.1. A winning condition W_i in a game G is S-closed with respect to W, if the following two conditions are satisfied:

1. For any *Survivor*'s position $s \in W_i$, there exists an a such that $(s, a) \in E$ and $a \in RA(S, W_i, W \setminus W_i)$.
2. For any *Adversary*'s position $s \in W_i$ and all a with $(s, a) \in E$, we have $a \in RA(S, W_i, W \setminus W_i)$.

Note that the definition of S-closedness of the previous section is the same as S-closedness with respect to V. Informally, when W_i is an S-closed winning set with respect to W, then *Survivor* can force a play that visits only nodes in W_i and don't care nodes in D. Similar to the Lemma 4.1, we can show:

Lemma 5.2. If *Survivor* wins the relaxed partition network game $\mathcal{G}$ then one of the winning conditions must be S-closed with respect to W.

For a set of nodes $X \subseteq V$ with for all $s \in X \cap S$, there is an $a \in X \cap A$ with $(s, a) \in E$ and for all $a \in X \cap A$, there is an $s \in X \cap S$ with $(a, s) \in E$, we can define the subgame, induced by X with initial position $v' \in X$:

$$(X, S \cap X, A \cap X, E \cap (X \times X), v', W \cap X, \Omega \cap \mathcal{P}(X)),$$

where $\Omega \cap \mathcal{P}(X)$ is the collection of sets in Ω that are a subset of X. In other words, the game is similar to the original game, but now only nodes in X are visited.

Lemma 5.3. If *Survivor* wins the relaxed partition network game $\mathcal{G}$ then for one of the winning conditions W_i, we have that W_i is S-closed with respect to W, the subgame, induced by $RA(S, W_i, W \setminus W_i)$, with initial position an arbitrary $v \in W_i$ has a winning strategy for *Survivor*, and the start node v_0 of $\mathcal{G}$ belongs to REACH(S, W_i).

The proof of this lemma is similar to (but somewhat more detailed as) the proof of Lemma 4.2. The conditions of these lemmas can again be checked in $O(|V||E|)$ time, as the game, induced by $RA(S, W_i, W \setminus W_i)$ is a relaxed update game.

Suppose the conditions of the preceding lemma is fulfilled for winning condition W_1. If $v_0 \in$ REACH(S, W_1), *Survivor* wins the game. Otherwise, game $\mathcal{G}'$ can be defined as in the previous section, and we again have that *Survivor* wins the game, if and only if *Survivor* wins game $\mathcal{G}'$. The time to decide which player has a winning strategy is again bounded by $O(|E||V|^2)$. Thus, we finally have the following result.

Theorem 5.1. There is a $O(|V|^2|E|)$ time algorithm to decide whether a given game is a relaxed partition network.

6. Conclusions

In this paper, we gave some types of McNaughton games where one can decide in polynomial time which player has a winning strategy. The interest in these games is that they can be used as a model for infinite processes.

Several directions for further research remain open. At one hand, one can try to design faster algorithms for the problems solved in this paper. In addition, it would be interesting to see which kind of conditions on the winning sets produce efficient algorithms to solve the games, and what conditions turn this problem computationally intractable. Another problem is to pinpoint the precise complexity (in terms of complexity class) of the question to decide if a given player has a winning strategy for a given McNaughton game.

References

[1] Chandra, A. K., Kozen, D. C., Stockmeyer, L. J.: Alternation. *Journal of the ACM*, 28(1):114-133, January 1981.

[2] Dinneen, M. J., Khoussainov, B.: Update networks and their routing strategies. In *Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science, WG2000*, volume 1928 of *Lecture Notes on Computer Science*, pages 127–136. Springer-Verlag, June 2000.

[3] Gurevich, Y., Harrington, L.: Trees, Automata, and Games, STOCS, 1982, pages 60–65.

[4] Knight, J. F., Luense, B.: Control Theory, Modal Logic, and Games, In *Hybrid Systems IV*. Panos J. Antsaklis, Wolf Kohn, Anil Nerode, Shankar Sastry (Eds.), volume 1273 of *Lecture Notes in Computer Science*, pages 160–173. Springer, 1997.

[5] McNaughton, R.: Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.

[6] Nerode, A., Remmel, J., Yakhnis, A.: McNaughton games and extracting strategies for concurrent programs. *Annals of Pure and Applied Logic*, 78:203–242, 1996.

[7] Nerode, A., Yakhnis, A., Yakhnis, V.: Distributed concurrent programs as strategies in games. Logical methods (Ithaca, NY, 1992), pages 624–653, *Progr. Comput. Sci. Appl. Logic*, 12, Birkhauser Boston, Boston, MA, 1993.