Technical Reports on Mathematical and Computing Sciences: TR-C186
title: The Complexity of Routing Strategies in Update Networks
author: Crasmaru Marcel[1] and Shinya Umeno[1]
affiliation:
1. Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology
Meguro-ku Ookayama, Tokyo 152-8552
email: {marcel, umeno1}@is.titech.ac.jp

Abstract. An update game is a new model for studying networks and infinite duration processes. In this paper we analyze routing strategies in (bipartite) update networks. We show that any routing strategy, both independent of a relying graph and with advice, uses a space of size at least the logarithm of the number of vertices. We also present a nearly optimal strategy with advice to this lower bound.

1 Introduction

A lot of network can be represented as a finite directed graph, nodes standing for a type of system (processors, servers, etc.), and edges standing for communication channels. When we see the process of communication in a network as a infinite duration process, proving robustness of the network is desirable. In particular, the important property to the network is that the critical information of dynamic network is distributed to all the nodes.

A simple mathematical model for analyzing this kind of property in networks was introduced in [2], by seeing the communication process in a given network as a two player game where one player, called *Survivor*, tries to ensure that the desirable property for the given network holds and the other player, called *Adversary*, tries to prevent it.

An *(bipartite) update game* is a game on a bipartite graph $G = (S \cup A, E)$, where S and A is a partition of V. (We stipulate that the in and the out degree of every vertex is nonzero.) There are two players, called *Survivor* and *Adversary*, which control vertices in S and A, respectively.

The rules of the game allow moves along the edges of G, alternately by Survivor and by Adversary. A play of an infinite duration game is an infinite sequence of vertices $v_0, v_1, \ldots, v_i, \ldots$, such that the game rules are followed. Since the number of nodes is finite, some of the nodes, called *persistent nodes*, have to appear infinitely often in the play. A finite prefix sequence of a play is called a history. We say that a vertex v is *visited* if it occurs in the play. Survivor begins a play if the initial vertex is in S, otherwise Adversary begins a play. Survivor wins a play if the set of the persistent vertices of the play is V, otherwise Adversary wins. A *strategy* for Survivor is a function H from play histories $v_0, \ldots, v_i$, where $v_i \in S$, to A such that $(v_i, H(v_0, \ldots, v_i))$ is an edge. Adversary's strategy is similarly defined.

Given H, a strategy for a player, it may either win or lose the game when starting at a given vertex $v_0 \in V$, called an *initial vertex* . A winning strategy assigned to an initial vertex for a given player is one that leads to a win no matter how the opponent moves.

Definition 1.1 *An update game graph is an* update network *if Survivor has a winning strategy for every initial vertex on the game on it.*

We will call Survivor's winning strategies as *routing strategies.*

While deciding whether an update game is an update network or not is computationally easy (see [1] for an algorithm that runs in $\mathcal{O}(|V| \cdot |E|)$ time), designing Survivor winning strategies was not studied in detail. Some simple routing strategies in update networks were analyzed first in [2]. These strategies have the advantage of being independent of a relying digraph, but require linear space w.r.t. the number of vertices. A nice result would be to show that there exist a strategy independent of a relying digraph that requires much smaller space w.r.t. the number of vertices. However, we will show the lower bound of the space amount used by routing strategies is logarithmic w.r.t. the number of vertices.

Whereas the construction of the optimal strategy independent of a relying graph is desirable, we set forth the intermediate kind of strategies of which a strategy machine itself is independent of the graph, but can use "advice" deduced from the graph. In this paper, we also present a $\mathcal{O}(\log n)$ space strategy for the computational model, which is the nearly optimal strategy in terms of the space amount it uses.

2 Simplifying Update Networks

In the following we will show that given an update game on the graph $G = (V, E)$, it can be reduced in polynomial time to an update game $G^* = (V^*, E^*)$ having vertices of in and out degrees at most 2, and such that G is an update network iff G^* is an update network.

First we modify $G = (V, E)$ by replacing any vertex $u \in A$ of out degree $|Out(u)| = |\{v_1, v_2, v_3\}| = 3$ with a new subgraph as shown in the left part of Figure 1.

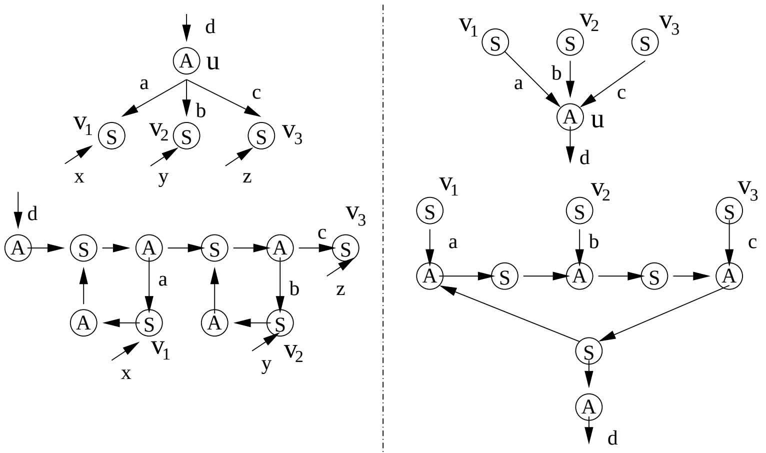

Figure 1: Reduction of an Adversary node

Let $G^1 = (V^1, E^1)$ with $= S^1 \cup A^1$ be the obtained update game. Suppose that G is an update network. Then there exists a routing strategy H such that all vertices V appear infinitely often in a play, no matter what the Adversary is doing. Then Survivor plays in G^1 against Adversary, by simulating in parallel a game on G.

Let v be the current vertex in game G^1 and w be the current vertex in game $G^{\cdot}$ Assume H_1 be the Survivor strategy that follows:

1. If $v = w$, then play in G and G^1 the move given by H, if $v \in S$; otherwise the move of

Adversary.

2. If $v \neq w$, $v \notin S$, and $w \in A - A^1$, then play a move in G^1, and do not move in G.

3. If $v \neq w$, $v \in S$, and $w \in A - A^1$ then play in G the move (w, v) and do not move in G^1.

Because by playing in G^1 following strategy H_1 we simply copy a play in G following routing strategy H, the vertices $V \cap V^1$ have to appear infinitely often in the play. However, H_1 does not update all the vertices V^1. By observing that from the set $V \cap V^1$ we may update all vertices V^1 in game G^1, we can easily modify H_1 to a strategy H_2 which updates all vertices V^1. This implies that G^1 is an update network.

Conversely, suppose that G^1 is an update network. Notice that by substituting vertices belonging to $V^1 - V$ with the correspondingly ones replaced from $V - V^1$ in any play of G^1, creates a play in G. Then a winning strategy in G^1 can be easily modified to produce a winning strategy in G. It follows that G is an update network, because G^1 is an update network.

We may apply the above reduction on vertices $u \in A$ of out degree over 3 inductively. We end up with an update game such that all Adversary vertices do have an out degree at most 2.

It is easy to check that the substitution shown on the right part of Figure 1 reduces further to an update game where in degrees of Adversary vertices are at most 2.

The vertices of Survivor of out and in degree over 3 may be substituted by subgraphs shown in Figure 2. It is easy to prove that the above reductions applied in the order given above, creates an update game $G^* = (V^*, E^*)$ which is an update network if and only if the initial game $G = (V, E)$ is. Moreover, each vertex has an in and an out degree of at most 2.

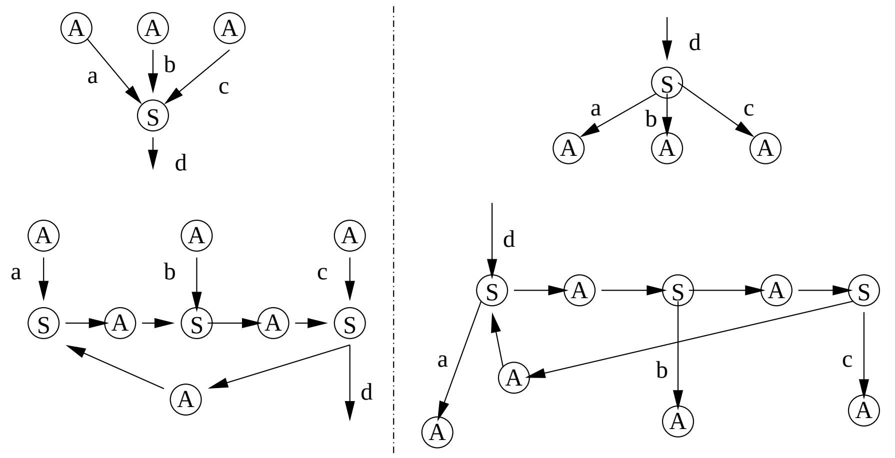

Figure 2: Reduction of a Survivor node

Lemma 2.1 *Any update game on graph $G = (V, E)$, can be reduced in polynomial time to an update game on a graph $G^* = (V^*, E^*)$, such that the following properties hold:*

1. $|In(v)|, |Out(v)| \leq 2$ for all $v \in V^*$;

2. G is an update network iff G^* is an update network.

3 The Computational Model of a Strategy

Before analyzing the complexity of computing a strategy, let us define the computational models of strategies. In the computational model, we assume that Survivor's strategy is restricted to a recursive function in order for a strategy to be represented as an Turing Machine M_S; whereas Adversary's strategy is a black box M_A so as not to constrain Adversary except the game rule.

Both machines can access to the read-only input tape where the encoding of the game graph $G = (S \cup A, E)$ and the initial vertex v_0 lies. Besides, M_S and M_A share the communication tape in which each player indicate the vertex to choose in its turn (represented in a binary sequence), and there is a channel between them to notify the opponent that it has finished to write to the communication channel. M_S has a internal working tape as does usual Turing Machine, and furthermore it can use an external read-only *advice tape* when specified (Figure 3).

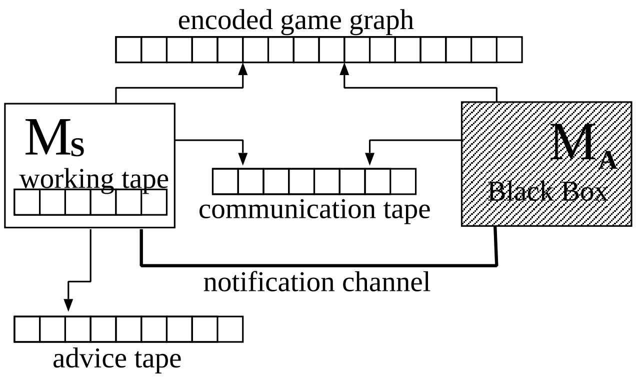

Figure 3: The Computational Model of Strategies

Given M_S and M_A, the play of the game goes as follows.

1. The first player to take a turn is decided according to the initial vertex given in the game.
2. The player to take a turn computes the next legal vertex – i.e., the vertex s.t. there is a edge from a current vertex to it – and write it on the communication tape.
3. To change turns, notify the opponent that it has finished writing by sending a signal to a channel.
4. The play never stops.

For the sake of analysis of the space complexity, we moreover impose M_S on some restrictions in this paper.

1. When sending a signal, M_S must set all its heads of the tapes back to the left most cell.
2. When a turn signal comes from M_A to M_S, the internal state of M_S must be reset to the initial state.
3. M_S can use only `0` or `1` on the working tape in it, and each cell is initially set to `0`. (We don't allow even the alphabet to indicate that the cell is empty.)

Owing to these restrictions, we have to focus only on the working tape when analyzing the space complexity, because the information passing trough each turn during the game must be only on the working tape.

We can consider the new notion of the space complexity used on the working tape under these restrictions; namely, *the passing information complexity.* In terms of this complexity, we just focused on the space complexity of the information passing through turns of the game, and omit the space used in computation on each game turn.

Depending on how M_S uses its internal and external tape during the computation and the computational bounds imposed to M_S, we distinguish the following types of routing strategies. In the rest of the paper, we use S_{choice} to represent the set of Survivor's vertices that have choice, i.e., $S_{choice} = \{v \in S | outgoing(v) \geq 2\}$, where *outgoing* is a function from a vertex to the number of outgoing edge from it. A_{choice} is defined analogously.

We say that M_S is an *s-space routing strategy independent of a relying digraph*, where s is a proper complexity function, if the following conditions hold.

1. M_S does not use the advice tape in its computation;
2. for any input $(G = (S \cup A, E), v_0)$, M_S uses at most $s(|S_{choice}|)$ cells on the working tape during its infinite computation; and
3. for any input $(G = (S \cup A, E), v_0)$ and for any adversary's strategy M_A, M_S updates the nodes V infinitely often.

Note that the second condition is appropriate in that the complexity of a strategy should be measured based on how many vertices belonging to Survivor have their choice.

We say that M_S is an *s-space routing strategy with advice*, where s is a proper complexity function, if the following conditions hold.

1. M_S use the advice tape in its computation;
2. the advice tape is constructed by a recursive function, which we call the *advice function*, that takes a game graph as input, and returns some strings that encodes advice;
3. for any input $(G = (S \cup A, E), v_0)$, M_S uses at most $s(|S_{choice}|)$ cells on the working tape during its infinite computation; and
4. for any input $(G = (S \cup A, E), v_0)$ and for any adversary's strategy M_A, M_S updates the nodes V infinitely often.

The length of each advice is at most $2^{s(|S_{choice}|)+\lfloor \log |V| \rfloor}$, because the strategy machine cannot use more than this amount of cells in the advice tape since all possible configurations of the machine at each time starting computation are $2^{s(|S_{choice}|)+\lfloor \log |V| \rfloor}$.

This model is based on the *interactive proof system* model introduced by Goldwasser, Micali, and Rackoff [3] (See, for example, [4], chapter 12).

4 A Space Lower Bound of Routing Strategies

In this section, we will consider only update games played on graphs having vertices of out degree at most 2, thank to Lemma 2.1.

Theorem 4.1 *There is no routing strategy either independent of a relying digraph or with advice using less than* $\lfloor \log(|V|) \rfloor - 2$ *space in its computation.*

Proof: Let M be a TM such that M is a log-space routing strategy either independent on the relying digraph or with advice.

Suppose that for every update network G, every vertex v of G, and every Adversary strategy all nodes of G are visited an unbounded number of times through a walk performed by M on $G, v, (a_i)_{i>0}$.

Consider the family of bi-partite digraphs $G_{d>0} = (V_d, E_d)$, where $V_d = S_d \cup A_d$, as shown in Figure 3. One can easily check that G_d is an update network, that is, Survivor can visit any vertex infinitely often no matter what Adversary responds. Let S_{G_d} be the set of branching Survivor vertices, that is $S_{G_d} = \{v_1^1, v_1^2, v_2^2, \ldots, v_1^d, \ldots, v_{2^{d-1}}^d\}$. By a simple counting one can see that S_{G_d} has cardinality $2^d - 1$, and $|V_d| = 4 \times |S_{G_d}| + 2$ holds. We have then $\lfloor \log(|V|) \rfloor - 2 = \lfloor \log |S_{G_d}| \rfloor$ for all $d \geq 2$.

Let us fix a walk performed by M starting from input G_d, v. Because v_1^1 appears infinitely often in this walk, we can define $C(d, v, v_1^1)$ as the set of all strings appearing infinitely often on the working tape of M, in any possible configuration of the walk in which M provides a move for vertex v_1^1.

One can observe that there are exactly 2^d different paths starting from v_1^1 and finishing at vertices of type a_{i0}, a_{i1}, with $1 \leq i \leq 2^{d-1}$. Since the Turing machine M is deterministic, we see that any configuration provided by M for a move at vertex v_1^1 follows exactly one of the paths finishing at vertices of type a_{i0}, a_{i1}, with $1 \leq i \leq 2^{d-1}$ (see Figure 3).

From $\lfloor \log(|V|) \rfloor - 2 = \lfloor \log(|S_{G_d}|) \rfloor = d - 1$, it follows that $|C(d, v, v_1^1)| \leq 2^{d-1}$. We have then $|C(d, v, v_1^1)| \leq 2^{d-1} < 2^d$. We conclude that there is at least one path starting from v_1^1 and finishing to a certain vertex a_{ij}, with $1 \leq i \leq 2^{d-1}, j \in \{0, 1\}$, such that a_{ij}, is not visited infinitely often on walk performed by M starting from input G_d, v. This contradicts the assumption that M is a routing strategy. $\square$

5 A Logarithmic-Space Routing Strategy with Advice

In this section, we present a logarithmic-space routing strategy with advice. According to the space lower bound of routing strategies in Section 4 and careful examination of the strategy, we conclude that the strategy presented is nearly optimal in terms of the space complexity.

The following is the intuitive explanation of how the strategy works. On the advice tape we write the encoding of a table that tells which choices to take to the strategy Turing machine. The column of the table indicates the position of the game, and its row indicates some kind of "phase" of the strategy. The element of the table tells which vertex to choose based on the current position and the current phase of the strategy.

5.1 Making The Table

To construct the table for the advice tape, we first make a multigraph G', called a *derived graph* of G, from the original game graph G in order to distill the essential information of the graph to decide a winner.

We first define the following notion.

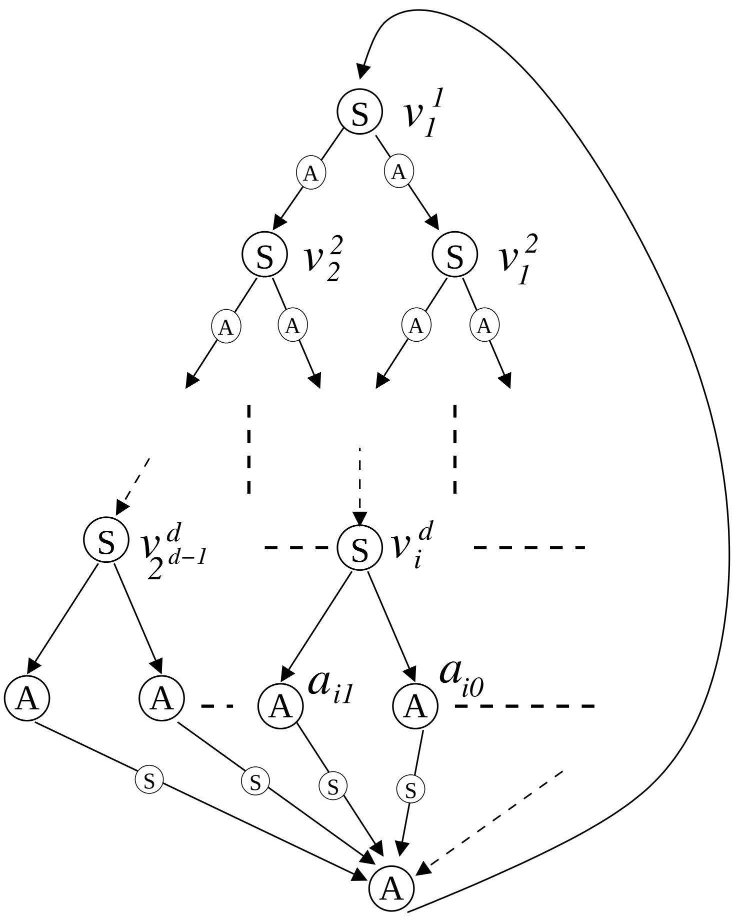

Figure 4: The Family of Update Networks G_d, where $d > 0$

Definition 5.1 *Determined Path*
Given a game graph $G = (V, E)$, a finite sequence $v_1, ..., v_n$ of the vertices is called a determined path from v_1 to v_n iff for all $i, 1 \leq i \leq n-1$, $(v_i, v_{i+1}) \in E$, and for all $j, 2 \leq j \leq n-1$, $outdegree(v_j) = 1$

Note that all edges are determined paths. Intuitively, along a determined path (except its first and last vertex), both Survivor and Adversary have no choice.

We now define a derived graph G'. Intuitively, in G', we represent each determined path of G by one edge. Precisely, G' is a multigraph derived from $G = (S \cup A, E)$ as follows.

1. $V' = S_{choice} \cup A_{choice}$
2. $E'_S = \{(u, v, l) \in S_{choice} \times V' \times V|$ there is a determined path from u to v and the label of the edge is $l\}$
3. $E'_A = \{(u, v, l) \in A_{choice} \times V' \times V|$ there is a determined path from u to v and the label of the edge is $l\}$
4. $E' = E'_S \cup E'_A$

Note that each edge of G' is specified by a triple whose third element, l, the label of the edge is used to distinguish edges that have the same start and goal vertex, but original paths are different. We may use polynomial time w.r.t. the number of vertices to construct a derived graph from an original game graph.

Figure 4 represents a example of the derived graph.

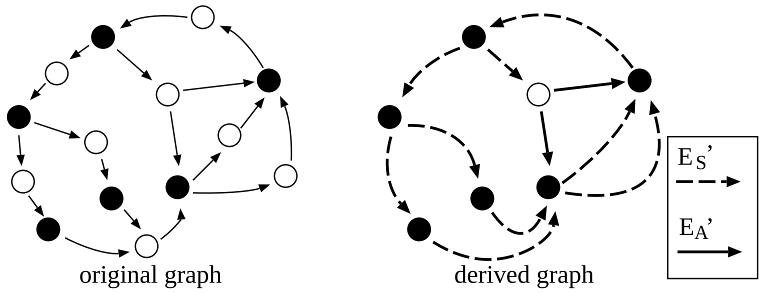

Figure 5: The Exsample of A Derived Graph

For each edge in E' of the derived graph, we can think of the corresponding vertices in V of the original graph.

Definition 5.2 *Corresponding Vertices*
Given the game graph $G = (V, E)$ and derived graph $G' = (V', E')$ from it, we define the corresponding vertices of each edge e' in E', represented as corr(e'), as the following.
$corr(e') = \{v \in V |\ v \textit{ is on a determined path from which } e' \textit{ is derived }\}$

Note that, the first and the last vertices of the path are also in $corr(e)$. We also use the $corr$ notation to a set of edges; that is, for a set of edges E, $corr(E) = \bigcup_{e \in E} corr(e)$

The following lemma is important to prove the correctness of the strategy presented in this section.

Lemma 5.1 *If a given game is an update network, $corr(E'_S) = V$*

Proof: For the sake of contradiction, suppose $V \setminus corr(E'_S) \neq \phi$. That is, there is a vertex v_{fail} such that $v_{fail} \notin corr(E'_S)$. If v_{fail} is the first vertex in the corresponding vertices of some edge in E'_A, v_{fail} is also the last vertex of another edge in E'_A due to the fact that v_{fail} is only in $corr(E'_A)$. Hence, we claim that v_{fail} is either the last vertex or the intermediate vertex of some edge in E'_A. In either case, if Adversary does not choose to visit the determined path where v_{fail} belongs, v_{fail} cannot be visited. It, in turn, follows that there is an Adversary's strategy that prevent v_{fail} from being visited infinitely often. This contradicts the assumption that the given graph is an update network. □

Thanks to this lemma, we can consider the routing strategy based on a derived graph. This is because we can use the following procedure to deduce a routing strategy for the original game. First, we make a strategy that can visit all edges in E'_S on the derived graph. Then, we naturally deduce a routing strategy for the original game; instead of choosing an edge to take next, the strategy choose the corresponding vertices of that edge in sequence. Thus, we consider the strategy based on a derived graph in the rest of this section.

Next, we focus on how to make a table encoded into the advice tape. First, we explain the outline of our strategy that makes use of the table. The strategy has $|E'_S|$ many phases each of which represents an edge in E'_S. While the phase of the strategy is $e \in E'_S$, it tries to visit e. When it achieves this goal, it changes its phase to next one. Changing its phase unbounded times, the strategy can visit all edges in E'_S, which guarantees that the strategy is a routing strategy.

The table helps the Survivor's machine to tell which choice to take to achieve its goal. The table has $|E'_S|$ columns to represent phases, and $|S_{choice}|$ rows to represent Survivor vertices with choice. Each entry of the table shows a vertex to take next depending on the current phase of the strategy and the current vertex on the game.

To construct the table, we need the predecessor function Pre, which intuitively computes, given a set of vertices U of the derived graph, the vertices out of U from which Survivor can reach V just at one step on the derived graph no matter what Adversary does; formally

$$Pre(U) = \{v \in V' \setminus U | \, [\exists w \in U : (v, w, l) \in E'_S] \vee [\forall w \in V' : (v, w, l) \in E'_A \rightarrow w \in U]\}$$

Using this function, we construct a table that tells which vertex to take next and whether or not the current vertex is a goal vertex. We assume the pointer to the table is given as T.

Algorithm ConstructTable
input : a derived graph G' (a pointer of the table T is given)
output : no value is returned, the effect is given through a pointer.

1: E:=E'_S
2: **while** E$\neq\emptyset$
3: Choose one edge e = (v,v',ℓ) from E and E:=E$-$\{e\}
4: c:=ℓ
5: T(v,c):=(ℓ, *true*)
6: Y:= \{v\}
7: **while** Y$\neq V'$
8: Compute Pre(Y) and for each Survivor's vertex v in Pre(Y):
T(w,c):=(ℓ,*false*), where e' = (v,w,ℓ) for some w in Y
9: Y:=Y$\cup Pre$(Y)

For each key (v, c), the second element of $\mathsf{T}(v, c)$ indicates whether or not v is the goal vertex for phase c. The computation of the while loop between line 7 and line 8 fills one column of the table T. When Y equals V', all element for the current column is filled. Choosing a edge in E'_S one after another (line 3), the algorithm fills all columns of the table.

We have to prove that the algorithm construct the table correctly; that is, the algorithm halts eventually, and it construct the correct table used by the strategy machine. Since the correctness of the tables itself correlates with how to use it, we only prove here that the algorithm terminates.

Lemma 5.2 *Given the derived graph of an update network as input, the algorithm* ConstructTable *halts eventually.*

Proof: It is sufficient to prove that while loop stated at line 7 eventually halts in every while loop stated at line 3. To lead the contradiction, suppose while loop at line 7 never stops. It implies that during the computation in while loop at line 7, Y never equals V'. Since V' is finite, and Y does not decrease its size in the loop, there must be a fixpoint $Y^* \neq V'$ of the operation "$\mathsf{Y} := \mathsf{Y} \cup Pre(\mathsf{Y})$." It follows $Pre(Y^*) = \phi$ because $Pre(Y^*) \subseteq V' \setminus Y^*$. It, in turn, implies that for every Survivor's vertex v in $V' \setminus Y^*$, there is no edge from v to a vertex in Y^*, and every Adversary's vertex in $V' \setminus Y^*$ has at least one choice to force the game not to go to Y^*. Then, there is a Adversary's strategy with which once the game goes to $V' \setminus Y^*$, the game never leaves $V' \setminus Y^*$. It leads the contradiction to the assumption that the given graph is the derived graph of an update network. $\square$

5.2 The Strategy Machine with Advice

We are now ready to construct the strategy machine with advice.

We consider the following abstract machine M.

1. When the signal comes from the opponent, M write a output to the communication tape according to the table; M looks up the table with key (v,p), where v,p is the current vertex and the current phase respectively, and get which vertex to write.
   If there is no entry to such a key, it means the current vertex is not in V'. It implies the current vertex has no choice, and M has just one vertex to write.

2. Then, if the current vertex is the goal vertex of the present phase, change the phase to the next one. (We suppose, without loss of generality, that phases are totally ordered, because each of them is originated from a label of a certain edge.)

3. After that, M get back its heads to the left most cells, and send a signal to the opponent.

We now prove that M is the routing strategy machine.

Lemma 5.3 *Given the table constructed in the previous subsection, M computes a routing strategy.*

Proof: Thanks to theorem 5.1, it is sufficient to prove that M can force the game to visit all vertices in $corr(E'_S)$ infinitely often. If the phases are changed unbounded times, M can visit all edges in E'_S infinitely often.

Thus, we only need to show that for every phase, the strategy achieve its goal vertex eventually. We prove it by the induction on the iteration of the while loop of ConstructTable at line 7. Choose arbitrarily one phase p. Let us indicate the value of Y at the k-th iteration of "$\mathsf{Y} = \mathsf{Y} \cup \mathsf{Pre_c}(\mathsf{Y})$" as Y_k, and $Y_0 = \{v^p_{goal}\}$, where v^p_{goal} is the goal vertex of the phase p

Base step: Suppose the current vertex is the goal vertex of p, then the condition is obviously satisfied.

Induction step: Suppose that if the current vertex is in Y_k, then M can visit the goal edge of p eventually. Consider when the current vertex is in Y_{k+1}, and M decides to choose w according to the table. This implies that choosing the edge labeled w on the derived graph, M can reach Y_k in one step. It follows that by the assumption of the induction, M can reach the goal vertex of p eventually. $\square$

We conclude this section with analysis of the space amount of working tape the M uses during its computation. Because M needs $|E'_S|$ many configurations to express its phases, M needs $\log|E'_S|$ cells in its working tape. Indeed, we can use the following Turing machine with $\log|E'_S|$ cells to compute an routing strategy. The machine store its phase in working tape, and it has $\log|E'_S|$ working-tape cells. To use a table constructed in this section, the table is encoded on the advise tape; specifically, each entry of the table is represented as $((v,p), entry)$ form, where v is the binary expression of a vertex and p the binary expression of a phase, and all entries lie on the advise tape. To look up the table, the machine check if the current vertex and phase match each key. The way how the machine communicates with opponent is same as one specified in a machine M above. It is easy to see that this machine does not need any extra working space than $\log|E'_S|$ cells. Because $|E'_S| < |S_{choice}|^2$ holds, M uses less than $\log|S_{choice}|^2$ many cells. It follows that this strategy is a logarithmic routing strategy with advice.

6 Future Work

There are several ways to expand the result on this paper.

First, a small space routing strategy independent of an relying graph is desirable. This is because the realistic condition of the network is severe, and we may not rely on any kind of advice.

We came up with the new notion of the complexity, *the passing information complexity*. It is also interesting to seek the optimal routing strategy independent of an relying graph in terms of this complexity, since in this notion, a Turing machine is allowed to use more space than usual one to compute the next vertex.

7 acknowledgments

We are deeply indebted to our supervisor, Professor Osamu Watanabe , for his tremendous support throughout preparing this technical report.

References

[1] M.J. Dineer and B. Khoussainov, Update games and Update Networks, *Journal of discrete Algorithms*,**2(1)**, pp.55–68, 1999

[2] M.J. Dineer and B. Khoussainov, Update Networks and Their Routing Strategies, *WG2000, Lecture Notes in Computer Science*, **1928**, pp. 127–136, 2000

[3] S. Goldwasser, S.Macali, and C. Rackoff, The knowledge complexity of interactive proof systems, *Proc. 17th ACM Symp. on the Theory of Computing*, pp.291-304, 1985; also, *SIAM J. of Comp.*, 18, pp. 186-208,1989.

[4] Christos H. Papadimitriou, Computational Complexity, *Addison-Wesley Publishing Company* ,1994.

[5] Crasmal Marcel, On Routing in Update Networks, *Research Reports on Mathematical and Computing Sciences C-170,* Dept. of MAth. and Computing Sciences, Tokyo Institute of Technology, 2003