数理解析研究所講究録 1375 巻 2004 年 229-232

On Strategies of Update Networks

Shinya Umeno
Department of Information Science
Tokyo Institute of Technology
Supervisor: Prof. Osamu Watanabe
梅野真也
東京工業大学情報科学科
指導教官:渡辺治

February, 2004

1 Update Networks

In the distriduted systems, it is often desiable to prove and maintain robustness of the network. In particular, the important property to the network is that the critical information of dynamic network is distributed to all the nodes.

An *update game* was introduced in [1] as a simple mathematical model for analyzing this kind of property in networks, by seeing the communication process in a given network as a two player infinite duration 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. More formally, an update game is the game on the bipartite finite digraph $G = (S \cup A, E)$. Survivor and Adversary play a game according to their *strategies*. Given an initial vertex, each player controls a *move* on the graph, alternatively by Survivor and Adversary. A sequence of moves makes an infinite sequence of vertices $\langle v_1, v_2, v_3, \ldots \rangle$ of the underlying graph, called a *play*. We call a finite prefix of a platy *history*. A strategy is defined as a function from a history to a next vertex. Survivor wins the game if and only if the vertices appearing on a play infinitely often, called *persistent vertices*, is equal to V. $\mathcal{P}(h_S, h_A, v_0)$ indicates the play by Survivor's strategy h_S, Adversary's starategy h_A, and an initial vertex v_0. A strategy is called a *winning strategy* if using it, a player can win a game no matter what the opponent does. In this paper, we mainly study a winning strategy for Survivor, called a *routing strategy*. Formally, a routing strategy is defined as follows.

Definition 1.1 *Routing Strategy*
A Survivor's strategy h_s is a routing strategy if the following condition folds.

$$\forall v_0 : \textit{initial vetrex}, \forall h_A : \textit{Adversary's strategy}$$
$$[\mathcal{P}(h_S, h_A, v_0) = V]$$

We call a update game with a routing strategy *update network* (Figure 1).

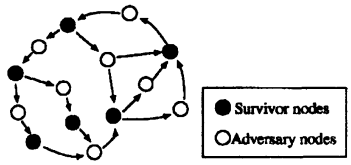

Figure 1: An example of an update network

2 Known Results

To consider an update game as a model of a network, it is desirable to decide if a network has a routing strategy or not in efficient time. In [1], Dinneen and Khoussainov showed an algorithm to decide if a given update-game graph $G = (V, E)$ is an update network in time $\mathcal{O}(|V||E|)$.

Theorem 2.1 *(Dinneen and Khoussainov[1]) There exists an algorithm to decide whether an update-game graph $G = (V, E)$ is an update network in time $\mathcal{O}(|V||E|)$.*

In [2] they also introduced a simple routing strategy, the *cycling neighbor strategy* of linear space w.r.t. the number of Survivor's nodes.

Theorem 2.2 *(Dinneen and Khoussainov[1]) The cyclic neighbor strategy is a winning strategy for any update network.*

Crasmaru, in [3], studied the computational complexity of a routing strategy, and showed a space lower bound of a routing strategy logarithmic with respect to the number of Survivor's nodes.

Theorem 2.3 *(Crusmaru) There is no strategy either independent of the underlying graph or with advice using less than logarithmic space w.r.t. the number of branching Survivor vertices.*

In the same paper he also proved that a simple randomized strategy h^{rand} for Survivor is a randomized routing strategy.

Theorem 2.4 *(Crusmaru)* h^{rand} *is a randomized routing strategy for any update network.*

3 Our Results

Even though the cycling neighbor strategy is simple and has a interesting property, i.e., independence of the network topology, it uses a linear space w.r.t. the number of Survivor nodes. Whereas the construction of the strategy independent of a relying graph with smaller space is desirable, we set forth the intermediate king of strategies of which a strategy itself is independent of the graph, but can use "advice" deduced from the graph.

First we construct a computational-model framework of an update game to analyze a strategy of update games using a Turing machine. Then, we study the space complexity of routing strategies, and randomized strategies.

In a conputational model model, we assume that any Survivor's strategy is a recursive function. This assumption allows us to see a strategy as a Turing Machine M_S. On the contrary Adversary's strategy is a black box M_A; thus, we do not constrain Adversary except for the game rules.

Both machines have access to the read-only input tape where an encoding of the game graph $G = (S \cup A, E)$ and the initial vertex v_0 lie. Besides, M_S and M_A share the communication tape in which each player indicates the next vertex of the play (represented as a binary sequence), and there is a channel between them to notify the opponent that it has finished to write to the communication tape. M_S has an internal working tape, and furthermore it has access to an external read-only *advice tape* when specified . Note that an advice tape is introduced to analyze the space complexity of the strategy, and its content is given by some algorithm whose input is the information of an underlying game graph. Thus, this additional tape does not change the power of the Survivor's strategy in terms of the ability to win a game. In the case of randomized strategy, M_S has access to a random-bit string (Figure 2).

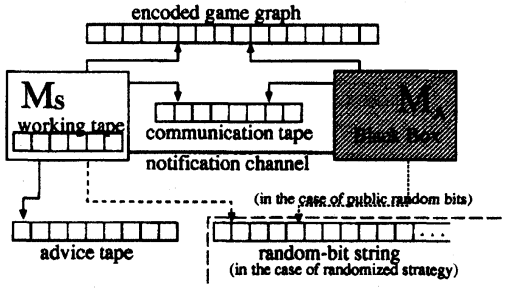

Figure 2: A Computational Model of Strategies

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

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

In the case of deterministic strategies, we distinguish the following kinds of strategy depending on

how M_S uses its internal and external tape during the computation and the computational bounds imposed on M_S.

We say that M_S is a *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.

We say that M_S is a *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 contents of 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 the 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.

We constructed a logarithmic space strategy with advice, which is almost optimal in terms of a space lower bound of a routing strategy. The strategy has acsess to the table represented on the advice tape. Reffering table to decide a next move in each turn, the strategy can update all nodes infinitely often.

Theorem 3.1 *There is a logarithmic-space routing strategy with advice.*

We also analyze a randomized strategy introduced in [3] more deeply. We consider three kinds of randomized strategies; a randomized strategy with private random bits; one with public random bits; and one with pseudo-random bits. The difference between former two kinds of random bits is that while Adversary cannot see private random bits, it can see public random bits, which intuitively means that Adversary can precisely "predict" the random bits used by Survivor in the future. A Survivor's strategy with pseudo-random bits has access to the pseudo-random bits. This pseudo-random bits are made from some pseudo-random generator whose input is some finite random bits called a *seed*. We assume that Adversary knows how to make infinite-length pseudo-random bits from a seed and the length of the seed, while it cannot see a seed itself. In randomized case, we assume h_S has an additional argument for a randomized-bit string. h_S^B indicate a function such that for any history ξ, $h_S^B(\xi) = h_S(\xi, B)$. h_A^B is similarly defined. In a case of pseudo-random bit, we use σ, instead of B, to represent a seed. We formally define the notion of a winning strategy for randomized cases.

Definition 3.1 *Randomized Routing Strategy with Private Random Bits*
A Survivor's strategy h_S is a randomized routing strategy with private random bits if the following condition holds.

$$\forall v_0 : \textit{initial vetrex}, \forall h_A : \textit{Adversary's strategy}$$
$$[\Pr_B\{\mathcal{P}(h_S^B, h_A, v_0) = V\} = 1]$$

Definition 3.2 *Randomized Routing Strategy with Public Random Bits*
A Survivor's strategy h_S is a randomized routing strategy with public random bits if the following condition holds.

$$\forall v_0 : \textit{initial vetrex}, \forall h_A^B : \textit{Adversary's strategy}$$
$$[\Pr_B\{\mathcal{P}(h_S^B, h_A^B, v_0) = V\} = 1]$$

Definition 3.3 *Randomized Routing Strategy with Pseudo-Random Bits*
A Survivor's strategy h_S is a randomized routing strategy with pseudo-random bits if the following condition holds.

$$\forall v_0 : \textit{initial vetrex}, \forall h_A : \textit{Adversary's strategy}$$
$$[\Pr_{\sigma}\{\mathcal{P}(h_S^{\sigma}, h_A, v_0) = V\} = 1]$$

Next, we analyze the difference between three kinds of randomized strategies, in terms of a property "winning", using a simple randomized strategy, an *ignorant randomized strategy.* An ignorant strategy uses a random bits without any technique or trick; Survivor's next move is determined just by random bits, like if the current random bit is 0, it moves to right, otherwise moves to left. We prove that while a ignorant random strategy with private random bits is a randomized routing strategy, those with public random bits and pseudo-random bits are not winning.

Theorem 3.2 *An ignorant randomized strategy with private random bits is a randomized routing strategy for any update networks.*

Theorem 3.3 *There is an update network such that an ignorant randomized strategy with public random bits is not a randomized winning strategy.*

Theorem 3.4 *There is an update network such that an ignorant randomized strategy with pseudo random bits is not a randomized winning strategy.*

References

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

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

[3] Crasmaru 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.

[4] Crasmaru Marcel and Shinya Umeno, The Complexity of Routing Strategies in Update Networks, *Research Reports on Mathematical and Computing Sciences C-186,* Dept. of Math. and Computing Sciences, Tokyo Institute of Technology, 2003.