Computer Science Journal of Moldova, vol.**13**, no.2(38), 2005

Polynomial Time Algorithm for Determining Max-Min Paths in Networks and Solving Zero Value Cyclic Games*

Dmitrii D. Lozovanu

Abstract

We study the max-min paths problem, which represents a game version of the shortest and the longest paths problem in a weighted directed graph. In this problem the vertex set V of the weighted directed graph $G = (V, E)$ is divided into two disjoint subsets V_A and V_B which are regarded as positional sets of two players. The players are seeking for a directed path from the given starting position v_0 to the final position v_f, where the first player intends to maximize the integral cost of the path while the second one has aim to minimize it. Polynomial-time algorithm for determining max-min path in networks is proposed and its application for solving zero value cyclic games is developed.

Mathematics Subject Classification 2000: 90B10, 90C35, 90C27.

Keywords and phrases: max-min path, positional games, c–game on networks.

1 Introduction and Problem Formulation

In this paper we consider the max-min paths problem on networks, which generalizes classical combinatorial problems of the shortest and the longest paths in weighted directed graphs. This max-min paths problem arose as an auxiliary one when searching optimal stationary

*Supported by BGP CRDF-MRDA MOM2-3049-CS-03

strategies of players in cyclic games [1-3]. The main results are concerned with the existence of polynomial-time algorithms for determining max-min paths in networks and elaboration of such algorithms. The application of the proposed algorithms for studying and solving zero value cyclic games is shown.

The statement of the considered problem is the following.

Let $G = (V, E)$ be a directed graph with vertex set V, $|V| = n$, and edge set E, $|E| = m$. Assume that G contains a vertex $v_f \in V$ such that it is attainable from each vertex $v \in V$, i.e. v_f is a sink in G. On edge set E it is given a function $c : E \to R$, which assigns a cost c(e) to each edge $e \in E$. In addition the vertex set is divided into two disjoint subsets V_A and V_B ($V = V_A \bigcup V_B$, $V_A \bigcap V_B = \emptyset$), which we regard as position sets of two players.

On G we consider a game of two players. The game starts at position $v_0 \in V$. If $v_0 \in V_A$, then the move is done by the first player, otherwise it is done by the second one. The move means the passage from a position v_0 to a neighbour position v_1 through the edge $e_1 = (v_0, v_1) \in E$. After that if $v_1 \in V_A$, then the move is done by the first player, otherwise it is done by the second one and so on. As soon as the final position is reached the game is over. The game can be finite or infinite. If the final position v_f is reached in finite time, then the game is finite. In the case when the final position v_f is not reached, the game is infinite. The first player in this game has the aim to maximize $\sum_i c(e_i)$ while the second one has the aim to minimize $\sum_i c(e_i)$.

Strictly the considered game in normal form can be defined as follows. We identify the strategies s_A and s_B of players with the maps

$$s_A : u \to v \in V_G(u) \text{ for } u \in V_A;$$

$$s_B : u \to v \in V_G(u) \text{ for } u \in V_B,$$

where $V_G(u)$ represents the set of extremities of edges $e = (u, v) \in E$, i.e. $V_G(u) = \{v \in V | e = (u, v) \in E\}$. Since G is a finite graph then the set of strategies of players

$$S_A = \{s_A : u \to v \in V_G(u) \text{ for } u \in V_A\};$$

$$S_B = \{s_B : u \to v \in V_G(u) \text{ for } u \in V_B\}$$

are finite sets. The payoff function $F_{v_0}(s_A, s_B)$ on $S_A \times S_B$ is defined in the following way.

Let be in G a subgraph $G_s = (V, E_s)$ generated by edges of form $(u, s_A(u))$ for $u \in V_A$ and $(u, s_B(u))$ for $u \in V_B$. Then either a unique directed path $P_s(v_0, v_f)$ from v_0 to v_f exists in G_s or such a path does not exist in G_s. In the second case in G_s there exists a unique directed cycle C_s, which can be reached from v_0.

For given s_A and s_B we set

$$F_{v_0}(s_A, s_B) = \sum_{e \in E(P_s(v_0, v_f))} c(e),$$

if in G_s there exists a directed path $P_s(v_0, v_f)$ from v_0 to v_f, where $E(P_s(v_0, v_f))$ is a set of edges of the directed path $P_s(v_0, v_f)$. If in G there are no directed paths from v_0 to v_f, then we define $F_{v_0}(s_A, s_B)$ as follows. Let $P'_s(v_0, u_0)$ be a directed path, which connects the vertex v_0 with the cycle C_s and $P_s(v_0, u_0)$ has no other common vertices with C_s except u_0. Then we put

$$F_{v_0}(s_A, s_B) = \begin{cases} +\infty, & \text{if } \sum\limits_{e \in E(C_s)} c(e) > 0; \\ \sum\limits_{e \in E(P'_s(v_0, u_0))} c(e), & \text{if } \sum\limits_{e \in E(C_s)} c(e) = 0; \\ -\infty, & \text{if } \sum\limits_{e \in E(C_s)} c(e) < 0. \end{cases}$$

This game is related to zero-sum positional games of two players and it is determined by the graph G with the sink vertex v_f, the partition $V = V_A \bigcup V_B$, the cost function $c : E \to R$ and the starting position v_0. We denote the network, which determines this game, by (G, V_A, V_B, c).

In [4] it is shown that if G does not contain directed cycles, then for every $v \in V$ the following equality holds

$$p(v) = \max_{s_A \in S_A} \min_{s_B \in S_B} F_v(s_A, s_B) = \min_{s_B \in S_B} \max_{s_A \in S_A} F_v(s_A, s_B), \tag{1}$$

which means the existence of optimal strategies of players in the considered game. Moreover, in [4] it is shown that in G there exists a tree $T^* = (V, E^*)$ with sink vertex v_f, which gives the optimal strategies of players in the game for an arbitrary starting position $v_0 \in V$. The strategies of players are obtained by fixing

$$s_A^*(u) = v, \text{ if } (u, v) \in E^* \text{ and } u \in V_A \setminus \{v_f\};$$

$$s_B^*(u) = v, \text{ if } (u, v) \in E^* \text{ and } u \in V_B \setminus \{v_f\}.$$

In general case for an arbitrary graph G equality (1) may fail to hold. Therefore we formulate necessary and sufficient conditions for the existence of optimal strategies of players in this game and a polynomial-time algorithm for determining the tree of max-min paths from every $v \in V$ to v_f. Furthermore we show that our max-min paths problem on the network can be regarded as an zero value ergodic cycle game. Therefore the proposed algorithm can be used for solving such games.

The formulated game on network (G, V_A, V_B, c) in [4] is named the dynamic c-game. Some preliminary results related to this problem have been obtained in [4-7]. More general models of positional games on networks with p players have been studied in [8,9].

2 Algorithm for solving the problem on acyclic networks

The formulated problem for acyclic networks has been studied in [4].

Let $G = (V, E)$ be a finite directed graph without directed cycles and given sink vertex v_f. The partition $V = V_A \bigcup V_B$ $(V_A \bigcap V_B = \emptyset)$ of vertex set of G is given and the cost function $c : E \to R$ on edges is defined. We consider the dynamic c-game on G with given starting position $v \in V$.

It is easy to observe that for fixed strategies of players $s_A \in S_A$ and $s_B \in S_B$ the subgraph $G_s = (V, E_s)$ has a structure of directed tree with sink vertex $v_f \in V$. This means that the value $F_{v_0}(s_A, s_B)$ is determined uniquely by the sum of edge costs of the unique directed path $P_s(v_0, v_f)$ from v_0 to v_f. In [5,6] it is proved that for acyclic c-game on network (G, V_A, V_B, c) there exist the strategies of players s_A^*, s_B^* such that

$$\begin{aligned} p(v) = F_v(s_A^*, s_B^*) &= \max_{s_A \in S_A} \min_{s_B \in S_B} F_v(s_A, s_B) = \\ &= \min_{s_B \in S_B} \max_{s_A \in S_A} F_v(s_A, s_B) \end{aligned} \qquad (2)$$

and s_A^*, s_B^* do not depend on starting position $v \in V$, i.e. (2) holds for every $v \in V$.

The equality (2) is evident in the case when $\text{ext}(c, u) = 0$, $\forall u \in V \setminus \{v_f\}$, where

$$\text{ext}(c, u) = \begin{cases} \max\limits_{v \in V_G(u)} \{c(u, v)\}, u \in V_A; \\ \min\limits_{v \in V_G(u)} \{c(u, v)\}, u \in V_B. \end{cases}$$

In this case $p(u) = 0$, $\forall u \in U$ and the optimal strategies of players can be obtained by fixing the maps $s_A^* : V_A \setminus \{v_f\} \to V$ and $s_B^* : V_B \setminus \{v_f\} \to V$ such that $s_A^* \in \text{VEXT}(c, u)$ for $u \in V_A \setminus \{v_f\}$ and $s_B^* \in \text{VEXT}(c, u)$ for $u \in V_B \setminus \{v_f\}$, where

$$\text{VEXT}(c, u) = \{v \in V_G(u) | c(u, v) = \text{ext}(c, u)\}.$$

If the network (G, V_A, V_B, c) has the property $\text{ext}(c, u) = 0$, $\forall u \in V \setminus \{v_f\}$, then it is named the network in canonic form. So, for the acyclic c-game on network in canonic form equality (2) holds and $p(v) = 0$, $\forall v \in V$.

In general case equality (2) can be proved using properties of the potential transformation $c'(u, v) = c(u, v) + \varepsilon(v) - \varepsilon(u)$ on edges $e = (u, v)$ of the network, where $\varepsilon : V \to R$ is an arbitrary real function on V (the potential transformation for positional games has been introduced in [2]). The fact is that such transformation of the costs on edges of the acyclic network in c-game does not change the optimal strategies of players, although values $p(v)$ of positions $v \in V$ are changed by $p(v) + \varepsilon(v_f) - \varepsilon(v)$. It means that for an arbitrary function $\varepsilon : V \to R$ the optimal strategies of the players in acyclic c-games on the networks

(G, V_A, V_B, c) and (G, V_A, V_B, c') are the same. Using such property in [4,5] the following theorem is proved.

Theorem 1. *For an arbitrary acyclic network (G, V_A, V_B, c) with a sink vertex v_f there exists a function $\varepsilon : V \to R$ which determines the potential transformation $c'(u,v) = c(u,v) + \varepsilon(v) - \varepsilon(u)$ on edges $e = (u,v)$ such that the network (G, V_A, V_B, c) has the canonic form. The values $\varepsilon(v)$, $v \in V$, which determine $\varepsilon : V \to R$, can be found by using the following recursive formula*

$$\varepsilon(v_f) = 0$$

$$\varepsilon(u) = \begin{cases} \max\limits_{v \in V_G(u)} \{c(u,v) + \varepsilon(v)\} \text{ for } u \in V_A \setminus \{v_f\}; \\ \min\limits_{v \in V_G(u)} \{c(u,v) + \varepsilon(v)\} \text{ for } u \in V_B \setminus \{v_f\}. \end{cases} \tag{3}$$

On the basis of this theorem the following algorithm for determining optimal strategies of players in c-game is proposed in [4,5].

Algorithm 1.

1. Find the values $\varepsilon(u)$, $u \in V$, according to recursive formula (3) and the corresponding potential transformation $c'(u,v) = c(u,v) + \varepsilon(v) - \varepsilon(u)$ on edges $(u,v) \in E$.
2. Fix arbitrary maps $s_A^* : V_A \setminus \{v_f\} \to V$ and $s_B^*(u) \in \mathrm{VEXT}(c', u)$ for $u \in V_B \setminus \{v_f\}$. $\square$

Remark 1. *The values $\varepsilon(u)$, $u \in V$, represent the values of the acyclic c-game on (G, V_A, V_B, c) with starting position u, i.e. $\varepsilon(u) = p(u)$, $\forall u \in V$. Algorithm 1 needs $O(n^2)$ elementary operations because the tabulation of the values $\varepsilon(u)$, $u \in V$, using formula (3) for acyclic networks needs such number of operations.*

3 The main results for the problem on an arbitrary network

First of all we give an example which shows that equality (1) may fails to hold. In fig.1 it is given the network with starting position $v_0 = 1$ and final position $v_f = 4$, where positions of the first player are represented by cycles and positions of the second player are represented by squares; values of cost functions on edges are given alongside them.

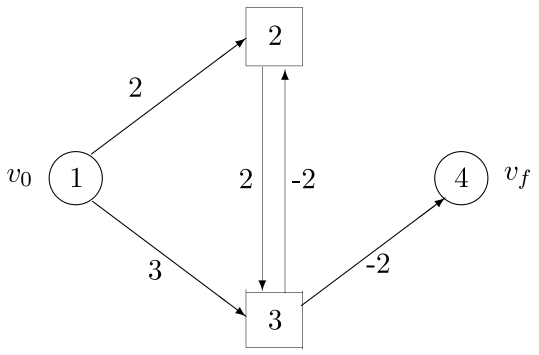

Fig. 1

It is easy to observe that

$$\max_{s_A \in S_A} \min_{s_B \in S_B} F_{12}(s_A, s_B) = 1, \quad \min_{s_B \in S_B} \max_{s_A \in S_A} F_{12}(s_A, s_B) = 2.$$

The following theorem gives conditions for the existence of settle values $p(v)$ for each $v \in V$ in the c-game.

Theorem 2. *Let (G, V_A, V_B, c) be an arbitrary network with sink vertex $v_f \in V$. Moreover let us consider that $\sum_{e \in E(C_s)} c(e) \neq 0$ for every directed cycle C_s from G_s. Then for c-game on (G, V_A, V_B, c) condition (1) holds if and only if there exists a function $\varepsilon : V \to R$, which determines a potential transformation $c'(u, v) = c(u, v) + \varepsilon(v) - \varepsilon(u)$ on edges $(u, v) \in E$ such that* $\text{ext}(c', u) = 0$, *$\forall v \in V$. If $\sum_{e \in E(C_s)} c(e) \neq 0$ for every directed cycle and in G there exists the potential transformation $c'(u, v) = c(u, v) + \varepsilon(v) - \varepsilon(u)$ on edges $(u, v) \in E$, then $\varepsilon(v) = p(v)$, $\forall v \in V$.*

Proof. $\Longrightarrow$ Let us consider that $\sum_{e\in E(C_s)} c(e) \neq 0$ for every directed cycle C_s in G and condition (1) holds for every $v \in V$. Moreover, we consider that $p(v)$ is a finite value for every $v \in V$. Taking into account that the potential transformation does not change the cost of cycles, we have that such transformation does not change optimal strategies of players although values $p(v)$ of positions $v \in V$ are changed by $p(v) - \varepsilon(v) + \varepsilon(v_f)$. It is easy to observe that if we put $\varepsilon(v) = p(v)$ for $v \in V$, then the function $\varepsilon : E \to R$ determines the potential transformation $c'(u,v) = c(u,v) + \varepsilon(v) - \varepsilon(u)$ on edges $(u,v) \in E$ such that $\text{ext}(c', u) = 0$, $\forall v \in V$.

$\Longleftarrow$ Let us consider that there exists a potential transformation $c'(u,v) = c(u,v) + \varepsilon(v) - \varepsilon(u)$ on edges $(u,v) \in E$ such that $\text{ext}(c', u) = 0$, $\forall v \in V$. The value $p(v)$ of the game after the potential transformation is zero for every $v \in V$ and optimal strategies of players can be found by fixing s_A^* and s_B^* such that $s_A^*(u) \in \text{VEXT}(c', u)$ for $u \in V_A \setminus \{v_f\}$ and $s_B^*(u) \in \text{VEXT}(c', u)$ for $u \in V_B \setminus \{v_f\}$. Since the potential transformation does not change optimal strategies of players we put $p(v) = \varepsilon(v) - \varepsilon(v_f)$ and obtain (1). $\Box$

Corollary 1. *The values $p(v)$, $v \in V$, can be found as follows $p(v) = \varepsilon(v) - \varepsilon(v_f)$, i.e. the difference $\varepsilon(v) - \varepsilon(v_f)$ is equal to the cost of the max-min path from v to v_f. If $\varepsilon(v_f) = 0$, then $p(v) = \varepsilon(v)$, $\forall v \in V$.*

Corollary 2. *If for every directed cycle C_s in G the condition $\sum_e c(e) \neq 0$ holds then the existence of the potential transformation $c'(u,v) = c(u,v) + \varepsilon(v) - \varepsilon(u)$ on edges $(u,v) \in E$ such that*

$$\text{ext}(c', u) = 0, \forall v \in V \tag{4}$$

represents necessary and sufficient conditions for validity of equality (1) for every $u \in V$. In the case when in G there exists cycle C_s with $\sum_{e\in E(C_s)} c(e) = 0$ condition (4) becomes only necessary one for validity (1) for every $v \in V$.

Corollary 3. *If in c-game there exist the strategies s_A^* and s_B^*, for which (1) holds for every $v \in V$ and these strategies generate in G*

a tree $T_{s^*} = (V, E_{s^*})$ with sink vertex v_f, then there exists the potential transformation $c'(u,v) = c(u,v) + \varepsilon(v) - \varepsilon(u)$ on edges $(u,v) \in E$ such that the graph $G^0 = (V, E^0)$, generated by the set of edges $E^0 = \{(u,v) \in E \mid c'(u,v) = 0\}$, contains the tree T_{s^*} as a subgraph.

Taking into account the mentioned above results we may propose the following algorithm for determining the optimal strategies of players in c-game based on the constructing of the tree of min-max paths.

Algorithm 2.

Preliminary step (step 0) Set $V^* = \{v_f\}$, $\varepsilon(v_f) = 0$.
General step (step k) Find the set of vertices

$$V' = \{u \in V \setminus V^* \mid (u,v) \in E, v \in V^*\}.$$

For each $u \in V'$ we calculate

$$\varepsilon(u) = \begin{cases} \max\limits_{v \in O_{V^*}(u)} \{\varepsilon(v) + c(u,v)\}, \ u \in V_A \bigcap V'; \\ \min\limits_{v \in O_{V^*}(u)} \{\varepsilon(v) + c(u,v)\}, \ u \in V_B \bigcap V', \end{cases} \tag{5}$$

where $O_{V^*}(u) = \{v \in V^* \mid (u,v) \in E\}$. Then in $V^* \bigcup V'$ we find the subset

$$U^k = \left\{ u \in V^* \bigcup V' \;\middle|\; \operatorname*{extr}_{v \in O_{V^* \cup V'}(u)} \{\varepsilon(v) - \varepsilon(u) + c(u,v) = 0\} \right\}$$

and change V^* by U^k, i.e. $V^* = U^k$. After that we check if $V^* = V$. If $V^* \neq V$, then go to the next step. If $V^* = V$, then define the potential transformation $c'(u,v) = c(u,v) + \varepsilon(v) - \varepsilon(u)$ on edges $(u,v) \in E$ and find the graph $G^0 = (V, E^0)$, generated by the set of edges $E^0 = \{(u,v) \in E | c'(u,v) = 0\}$. In G^0 fix an arbitrary tree $T^* = (V, E^*)$, which determines the optimal strategies of players as follows:

$$s_A^*(u) = v, \text{ if } (u,v) \in E^* \text{ and } u \in V_A \setminus \{v_f\};$$

$$s^*_B(u) = v, \text{ if } (u,v) \in E^* \text{ and } u \in V_B \setminus \{v_f\}.$$

Now let us show that this algorithm finds the tree of max-min paths $T^* = (V, E^*)$ if such tree exists in G. $\square$

Denote by V^i the subset of V, where $v \in V^i$ if in T^* there exists the directed path $P_T(v, v_0)$ from v to v_0 which contains i edges, i. e. $V^i = \{v \in V \mid |P_{T^*}(v, v_0)| = i\}$. So, $V = V^0 \bigcup V^1 \bigcup V^2 \bigcup \cdots \bigcup V^r$ $(V^i \bigcap V^j = \emptyset)$, where $V^0 = \{v_f\}$ and V^i, $i \in \{1, 2, \ldots, r\}$, represents the level i of vertex set of T^*. If in G there exists several max-min trees $T^*_1 = (V, E^*_1)$, $T^*_2 = (V, E^*_2)$, …, $T^*_q = (V, E^*_q)$ then we will select the one which has number of levels $r = \min\limits_{1 \le i \le q} \{r_i\}$.

Theorem 3. *If in G there exists a tree of max-min path $T^* = (V, E^*)$ with sink vertex v_f then Algorithm 2 finds it using $k = r$ iterations. The running time of the algorithm is $O(n^3)$.*

Proof. We prove the theorem by using the induction principle on number of levels of max-min tree. If $r = 1$ the theorem is evident. Assume that the theorem is true for any $r \le p$ and let us show that it is true for $r = p + 1$.

Denote by $V^0, V^1, \ldots, V^r$ the level sets of the tree $T^* = (V, E^*)$, $V = V^0 \bigcup V^1 \bigcup V^2 \bigcup \cdots \bigcup V^r$ $(V^i \bigcap V^j = \emptyset)$. It is easy to observe that if we delete from T^* the vertex set V^r and corresponding pendant edges $e = (u, v)$, $v \in V^r$, then we obtain a tree $\overline{T}^* = (\overline{V}, \overline{E}^*)$, $\overline{V} = V \setminus V^r$. This tree $\overline{T}^*$ represents the tree of max-min paths for the subgraph $\overline{G} = (\overline{V}, \overline{E})$ of G generated by vertex set $\overline{V}$.

If we apply Algorithm 2 with respect to $\overline{G}$ then according to the induction principle we find the tree of max-min paths $\overline{T}^*$, which determines $\varepsilon : \overline{V} \to R$ and the potential transformation $\overline{c}(u, v) = c(u, v) + \varepsilon(u) - \varepsilon(v)$ on edges $(u, v) \in \overline{E}$ such that $\text{extr}(c', v) = 0$, $\forall v \in \overline{V}$. So, Algorithm 2 on $\overline{G}$ determines uniquely the values $\varepsilon(u)$ according to (5).

It is easy to observe that in G for an arbitrary vertex $u \in V^r$ calculated on the basis of formula (5) the following condition holds:

$$\varepsilon(u) = \begin{cases} \max\limits_{v \in V_G(u)} \{\varepsilon(v) + c(u,v)\}, \ u \in V^r \bigcap V_A; \\ \min\limits_{v \in V_G(u)} \{\varepsilon(v) + c(u,v)\}, \ u \in V^r \bigcap V_B. \end{cases}$$

This means that if we apply Algorithm 2 on G then after $r-1$ iterations the vertex set U^{r-1} coincides with $V \setminus V^r$. So, Algorithm 2 determines uniquely the values $\varepsilon(v)$, $v \in V$. Nevertheless here we have to note that in the process of the algorithm $V^k \subset U^k$ and V^k may differ from U^k for some $k = 1, 2, \dots, r$.

Taking into account that at the general step of the algorithm it needs $O(n^2)$ elementary operations and $k \le r(r \le n)$ we obtain that the running time of the algorithm is $O(n^3)$. $\square$

4 An application of the algorithm for solving zero value cyclic games

In this section we show that zero value ergodic cycle game can be regarded as max-min paths problem and therefore the proposed algorithm can be used for determining the optimal strategies of players in such cyclic games.

At first we remind the formulations of cyclic games and some necessary preliminary results.

4.1 Cyclic games: problem formulation

Let $G = (V, E)$ be a finite directed graph in which every vertex $u \in V$ has at least one leaving edge $e = (u, v) \in E$. On edge set E a function c: $E \to R$ is given which assigns a cost $c(e)$ to each edge $e \in E$. In addition the vertex set V is divided into two disjoint subsets V_A and V_B ($V = V_A \cup V_B$, $V_A \cap V_B = \emptyset$) which we will regard as positions sets of two players.

On G we consider the following two-person game from [1,2]. The game starts at position $v_0 \in V$. If $v_0 \in V_A$ then the move is done by first player, otherwise it is done by second one. The move means the passage from position v_0 to the neighbour position v_1 through the edge $e_1 = (v_0, v_1) \in E$. After that if $v_1 \in V_A$ then the move is done by first player, otherwise it is done by second one and so on indefinitely. The first player has the aim to maximize $\lim_{t\to\infty} \inf \frac{1}{t} \sum_{i=1}^{t} c(e_i)$ while the second player has the aim to minimize $\lim_{t\to\infty} \sup \frac{1}{t} \sum_{i=1}^{t} c(e_i)$.

In [1,2] it is proved that for this game there exists a value $p(v_0)$ such that the first player has a strategy of moves that insures $\lim_{t\to\infty} \inf \frac{1}{t} \sum_{i=1}^{t} c(e_i) \geq p(v_0)$ and the second player has a strategy of moves that insures $\lim_{t\to\infty} \sup \frac{1}{t} \sum_{i=1}^{t} c(e_i) \leq p(v_0)$. Furthermore in [1,2] it is shown that the players can achieve the value $p(v_0)$ applying the strategies of moves which do not depend on t. This means that the considered game can be formulated in the terms of stationary strategies. Such statement of the game in [2] is named cyclic game.

The strategies of players in cyclic game are defined as maps

$$s_A\colon\ u \to v \in V_G(u) \ \text{ for } \ u \in V_A; \ s_B\colon\ u \to v \in V_G(u) \ \text{ for } \ u \in V_B,$$

where $V_G(u)$ represents the set of extremities of edges $e = (u, v) \in E$, i.e. $V_G(u) = \{v \in V \mid e = (u, v) \in E\}$. Since G is a finite graph then the sets of strategies of players

$$\begin{array}{l} S_A = \{s_A\colon\ u \to v \in V_G(u) \ \text{ for } \ u \in V_A\}; \\ S_B = \{s_B\colon\ u \to v \in V_G(u) \ \text{ for } \ u \in V_B\} \end{array}$$

are finite sets. The payoff function $F_{v_0}\colon S_A \times S_B \to R$ in cyclic game is defined as follows.

Let $s_A \in S_A$ and $s_B \in S_B$ be fixed strategies of players. Denote by $G_s = (V, E_s)$ the subgraph of G generated by edges of form $(u, s_A(u))$

for $u \in V_A$ and $(u, s_B(u))$ for $u \in V_B$. Then G_s contains a unique directed cycle C_s which can be reached from v_0 through the edges $e \in E_s$. The value $F_{v_0}(s_A, s_B)$ we consider equal to mean edges cost of cycle C_s, i.e.

$$F_{v_0}(s_A, s_B) = \frac{1}{n(C_s)} \sum_{e \in E(C_s)} c(e),$$

where $E(C_s)$ represents the set of edges of cycle C_s and $n(C_s)$ is a number of the edges of C_s. So, the cyclic game is determined uniquely by the network (G, V_A, V_B, c) and starting position v_0. In [1,2] it is proved that there exist the strategies $s_A^* \in S_A$ and $s_B^* \in S_B$ such that

$$\begin{gathered}\overline{p}(v) = F_v(s_A^*, s_B^*) = \max_{s_A \in S_A} \min_{s_B \in S_B} F_v(s_A, s_B) = \\ = \min_{s_B \in S_B} \max_{s_A \in S_A} F_v(s_A, s_B), \ \forall\, v \in V.\end{gathered}$$

So, the optimal strategies s_A^*, s_B^* of players in cyclic games do not depend on starting position v although for different positions $u, v \in V$ the values $\overline{p}(u)$ and $\overline{p}(v)$ may be different. It means that the positions set V can be divided into several classes $V = V^1 \cup V^2 \cup \cdots \cup V^k$ according to values of positions $\overline{p}^1, \overline{p}^2, \ldots, \overline{p}^k$, i.e. $u, v \in V^i$ if and only if $\overline{p}^i = \overline{p}(u) = \overline{p}(v)$. In the case $k = 1$ the network (G, V_A, V_B, c) is named the ergodic network [2]. In [5, 6] it is shown that every cyclic game with arbitrary network (G, V_A, V_B, c) and given starting position v_0 can be reduced to an auxiliary cyclic game on auxiliary ergodic network (G', V_A', V_B', c').

4.2 Some Preliminary Results

In [2] the following theorem is formulated and proved.

Theorem 4. *Let (G, V_A, V_B, c) be an arbitrary network with the properties described in section 1. Then there exists the value $p(v)$, $v \in V$ and the function ε: $V \to R$ which determine a potential transformation $c'(u, v) = c(u, v) + \varepsilon(v) - \varepsilon(u)$ for costs on edges $e = (u, v) \in E$, such that the following properties hold*

a) $\overline{p}(u) = ext(c', u)$ *for* $v \in V$,

b) $\overline{p}(u) = \overline{p}(v)$ *for* $u \in V_A \cup V_B$ *and* $v \in \mathrm{VEXT}(c', u)$,

c) $\overline{p}(u) \geq \overline{p}(v)$ *for* $u \in V_A$ *and* $v \in V_G(u)$,
d) $\overline{p}(u) \leq \overline{p}(v)$ *for* $u \in V_B$ *and* $v \in V_G(u)$,
e) $\max\limits_{e\in E} |c'(e)| \leq 2|V| \max\limits_{e\in E} |c(e)|$.

The values $\overline{p}(v), v \in V$ *on network* (G, V_A, V_B, c) *are determined unequally and the optimal strategies of players can be found in the following way: fix the arbitrary strategies* $s_A^*\colon V_A \to V$ *and* $s_B^*\colon V_B \to V$ *such that* $s_A^*(u) \in \mathrm{VEXT}(c', u)$ *for* $u \in V_A$ *and* $s_B^*(u) \in \mathrm{VEXT}(c', u)$ *for* $u \in V_B$.

Further we shall use the theorem 4 in the case of the ergodic network (G, V_1, V_2, c), i.e. we shall use the following corollary.

Corollary 4. *Let* (G, V_A, V_B, c) *be an ergodic network. Then there exist the value* p *and the function* $\varepsilon\colon V \to R$ *which determines a potential transformation* $c'(u, v) = c(u, v) + \varepsilon(v) - \varepsilon(u)$ *for costs of edges* $e = (u, v) \in E$ *such that* $\overline{p} = ext(c', u)$ *for* $u \in V$. *The optimal strategies of players can be found as follows: fix arbitrary strategies* $s_A^*\colon V_A \to V$ *and* $s_B^*\colon V_B \to V$ *such that* $s_A^*(u) \in \mathrm{VEXT}(c', u)$ *for* $u \in V_A$ *and* $s_B^*(u) \in \mathrm{VEXT}(c', u)$ *for* $u \in V_B$.

4.3 The reduction of cyclic games to ergodic ones

Let us consider an arbitrary network (G, V_A, V_B, c) with given starting position $v_0 \in V$ which determines a cyclic game. In [5, 6] it is shown that this game can be reduced to a cyclic game on auxiliary ergodic network $(G', W_A, W_B, \overline{c})$, $G' = (W, F)$ in which the value $p(v_0)$ is preserving, $v_0 \in W = V \cup X \cup Y$.

The graph $G' = (W, F)$ is obtained from G if each edge $e = (u, v)$ is changed by a triple of edges $e^1 = (u, x), e^2 = (x, y), e^3 = (y, v)$ with the costs $\overline{c}(e^1) = \overline{c}(e^2) = \overline{c}(e^3) = c(e)$. Here $x \in X, y \in Y$ and $u, v \in V; W = V \cup X \cup Y$. In addition in G' each vertex x is connected with v_0 by edge (x, v_0) with the cost $\overline{c}(x, v_0) = M$ (M is a great value) and each edge (y, v_0) is connected with v_0 by edge (y, v_0) with the cost $\overline{c} = (y, v_0) = -M$. In $(G', W_A, W_B, \overline{c})$ the sets W_A and W_B are defined as follows: $W_A = V_A \cup Y$; $W_B = V_B \cup X$.

It is easy to observe that this reduction can be done in linear time.

4.4 The reduction of zero value ergodic cyclic games to max-min paths problem

Let us consider a zero value cyclic game on ergodic network (G, V_A, V_B, c), $G = (V, E)$. Then according to Theorem 8 there exists the function $\varepsilon : V \to R$ which determines the potential transformation $c'(u, v) = c(u, v) + \varepsilon(v) - \varepsilon(u)$ on edges $(u, v) \in E$ such that

$$\mathrm{ext}(c, u) = 0,\ \forall v \in V. \tag{6}$$

This means that if v_f is a vertex of the cycle C_{s^*} determined by optimal strategies s_A^* and s_B^* then the problem of finding the function $\varepsilon : V \to R$ which determines the canonic potential transformation is equivalent to the problem of finding the values $\varepsilon(v)$, $v \in V$ in max-min paths problem on G with sink vertex v_f where $\varepsilon(v_f) = 0$.

So, in order to solve zero value cyclic game we fix each time a vertex $v \in V$ as a sink vertex ($v_f = v$) and solve a max-min paths problem on G with sink vertex v_f. If for given $v_f = v$ the obtained function $\varepsilon : V \to R$ on the basis of Algorithm 2 determines the potential transformation which satisfies (6) then we fix s_A^* and s_B^* such that $s_A^*(u) \in \mathrm{VEXT}(c', u)$ for $u \in V_A$ and $s_B^*(u) \in \mathrm{VEXT}(c', u)$ for $u \in V_B$. If for given v the function $\varepsilon : V \to R$ does not satisfy (6) then we select another vertex $v \in V$ as a sink vertex and so on. This means that the optimal strategies of players in zero value ergodic cyclic games can be found in time $O(n^4)$.

References

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

[2] V.A. Gurvich, A.V. Karzanov, L.G. Khachiyan, *Cyclic games and an algorithm to find minmax cycle means in directed graphs.* USSR, Computational Mathematics and Mathematical Physics, (28), 1988, p. 85–91.

[3] U. Zwick, M. Paterson, *The complexity of mean payoff games on graphs.* TCS, (158), 1996, p. 344–359.

[4] D.D. Lozovanu, *Algorithms to solve some classes of network min-max problems and their applications.* Cybernetics (29), 1991, p. 93–100.

[5] D.D. Lozovanu, *Extremal-Combinatorial problems and algorithms for its solving.* Kishinev, Stiinta, 1991.

[6] D. Lozovanu, *Polynomial time algorithm for determining optimal strategies in cyclic games.* 10 International IPCO Conference, New York, 2004 (Proceedings), p. 74–85.

[7] D. Lozovanu, *Polynomial time algorithm for determining max-min paths in networks and solving zero value cyclic games.* Computer Science Journal of Moldova, (2), 2005, p. 13–28.

[8] E. Boros, V. Gurvich, *On Nash-solvability in pure stationary strategies of finite games with perfect information which may have cycles.* DIMACS Technical Report, (18), 2002, p. 1–32.

[9] R. Boliac, D. Lozovanu, D. Solomon, *Optimal paths in network games with p players,* Discrete Applied Mathematics, vol. 99, N 1–3 (2000), p. 339–348.

D.D. Lozovanu,

Received September 7, 2005

Institute of Mathematics and Computer Science,
5 Academiei str.
Chişinău, MD−2028, Moldova.
E–mail: *lozovanu@math.md*

Computer Science Journal of Moldova, vol.**13**, no.2(38), 2005

Note about the upper chromatic number of mixed hypertrees

Kenneth Roblee, Vitaly Voloshin

Abstract

A mixed hypergraph is a triple $\mathcal{H} = (X, \mathcal{C}, \mathcal{D})$, where X is the vertex set and each of $\mathcal{C}$, $\mathcal{D}$ is a family of subsets of X, the $\mathcal{C}$-edges and $\mathcal{D}$-edges, respectively. A proper k-coloring of $\mathcal{H}$ is a mapping $c : X \to [k]$ such that each $\mathcal{C}$-edge has two vertices with a common color and each $\mathcal{D}$-edge has two vertices with distinct colors. Upper chromatic number is the maximum number of colors that can be used in a proper coloring. A mixed hypergraph $\mathcal{H}$ is called *a mixed hypertree* if there exists a host tree on the vertex set X such that every edge ($\mathcal{C}$- or $\mathcal{D}$-) induces a connected subtree of this tree.

We show that if a mixed hypertree can be decomposed into interval mixed hypergraphs then the upper chromatic number can be computed using the same formula.

1 Introduction

In this paper, we use the terminology of [1, 2, 3, 4, 5, 6, 7]. A mixed hypergraph $\mathcal{H} = (X, \mathcal{C}, \mathcal{D})$ (each element of $\mathcal{C} \cup \mathcal{D}$ is of size at least 2) is said to be a *mixed hypertree* if there exists a host tree $T = (X, F)$ such that every $C \in \mathcal{C}$ and every $D \in \mathcal{D}$ induces a subtree in T [7]. An *interval mixed hypergraph* represents a special case of a mixed hypertree, namely, when the host graph is simply a path. In a mixed hypergraph $\mathcal{H} = (X, \mathcal{C}, \mathcal{D})$, a subfamily $\{C_i\}$ of $\mathcal{C}$-edges is said to be a *sieve*, if for any $x, y \in X$ and any j, k, $j \neq k$, the following implication holds (see [1, 7]):

$$\{x, y\} \subseteq C_j \cap C_k \Rightarrow \{x, y\} = D \in \mathcal{D} \text{ for some } D \in \mathcal{D}.$$

In other words, a sieve represents a subfamily of $\mathcal{C}$-edges with the property that if two $\mathcal{C}$-edges intersect, then every pair of vertices from the intersection forms a $\mathcal{D}$-edge. It appears that sieves play a role in estimating the upper chromatic number. The maximum cardinality of a sieve is the *sieve number* of $\mathcal{H}$ and is denoted by $s(\mathcal{H})$. It is proved (see [1, 7]) that if $\mathcal{H} = (X, \mathcal{C}, \mathcal{D})$ is a reduced interval mixed hypergraph (the size of each $\mathcal{C}$-edge is at least 3, the size of each $\mathcal{D}$-edge is at least 2, and no included edges of any type), then

$$\overline{\chi}(\mathcal{H}) = |X| - s(\mathcal{H}).$$

A $\mathcal{C}$-edge is called *redundant* if it contains no other $\mathcal{C}$-edges and after its removal no new coloring appears. This property never happens in classic graph or hypergraph coloring. We call a mixed hypertree $\mathcal{H} = (X, \mathcal{C}, \mathcal{D})$ *simple* if it is reduced and has no redundant $\mathcal{C}$-edges. In order to generalize the result for the upper chromatic number from interval mixed hypergraphs to mixed hypertrees, it is necessary to investigate the following question: if $\mathcal{H} = (X, \mathcal{C}, \mathcal{D})$ is a simple mixed hypertree with $|X| = n$ and sieve number s, when is $\overline{\chi}(\mathcal{H}) = n - s$?

The equality holds for many classes of mixed hypertrees. In this paper, we prove it for mixed hypertrees having uniquely colorable separator with the respective subgraphs being interval mixed hypergraphs. However, for general mixed hypertrees it is not true. We exhibit that the difference between $\overline{\chi}(\mathcal{H})$ and $n - s$ can actually be made arbitrarily large.

2 Results

Theorem 1. *If $\mathcal{H} = (X, \mathcal{C}, \mathcal{D})$ is a simple mixed hypertree that can be decomposed into the union of interval mixed hypergraphs $T_1, T_2, \ldots, T_k$, where $k \geq 1$, so that if any two of these hypergraphs meet, they meet only at a single vertex, then we have*

$$\overline{\chi}(\mathcal{H}) = |X(\mathcal{H})| - s(\mathcal{H}).$$

Before we prove the theorem, we establish a lemma regarding the sieve number for simple mixed hypertrees as described in the statement of the theorem.

Lemma 1. *If $\mathcal{H} = (X, \mathcal{C}, \mathcal{D})$ is a simple mixed hypertree that can be decomposed into the union of interval mixed hypergraphs $T_1, T_2, \dots, T_k$, where $k \geq 1$, so that if any two of these hypergraphs meet, they meet only at a single vertex, then*

$$s(\mathcal{H}) = s(\mathcal{T}_1) + s(\mathcal{T}_2) + \cdots + s(\mathcal{T}_k).$$

Proof. Note that hyperedges in different maximum sieves in (say) T_i, T_j, where $i \neq j$, will also be in a maximum sieve of $\mathcal{H}$. For, if they intersect, they may only intersect at a single point. □

Proof. We prove the theorem by induction on the number k of interval mixed hypergraphs. For $k = 1$, the mixed hypertree is an interval mixed hypergraph, in which case it is already known the result holds. Let us assume the result is true if there are $k = m$ interval mixed hypergraphs, and establish that it is true when there are $m + 1$ interval mixed hypergraphs. Remove any one of the $(m + 1)$-many interval mixed hypergraphs $\mathcal{T}_i$ from $\mathcal{H}$ (except for the intersecting vertex), and consider the resulting mixed hypertree $\mathcal{H}'$. Note that $\mathcal{H}'$ is the union of m-many interval mixed hypergraphs, and so by the inductive hypothesis and with obvious notation we have

$$\overline{\chi}(\mathcal{H}') = |X'| - s(\mathcal{H}').$$

Now, for the removed interval mixed hypergraph $\mathcal{T}_i$ we have

$$\overline{\chi}(\mathcal{T}_i) = |X_i| - s(\mathcal{T}_i),$$

since it is an interval mixed hypergraph.

Now, we note that $|X(\mathcal{H})| = |X(\mathcal{H}')| + |X(\mathcal{T}_i)| - 1$. Let v be the common vertex. Before we re-insert $\mathcal{T}_i$ back into $\mathcal{H}'$ to form the original mixed hypertree $\mathcal{H}$, we re-color v (in $\mathcal{T}_i$) and all vertices in $\mathcal{T}_i$ having that color (in the maximum coloring) in the same color that v has in a maximum coloring of $\mathcal{H}'$. Thus, we have the following:

$$\begin{aligned}\overline{\chi}(\mathcal{H}) &= \overline{\chi}(\mathcal{H}') + \overline{\chi}(\mathcal{T}_i) - 1\\ &= |X(\mathcal{H}')| - s(\mathcal{H}') + |X(\mathcal{T}_i)| - s(\mathcal{T}_i) - 1\\ &= (|X(\mathcal{H}')| + |X(\mathcal{T}_i)| - 1) - (s(\mathcal{H}') + s(\mathcal{T}_i))\\ &= |X(\mathcal{H})| - s(\mathcal{H}).\end{aligned}$$

In the displayed equations above, the only item we have not yet established is the first equality. To prove this, first note that giving both $\mathcal{H}'$ and $\mathcal{T}_i$ a maximum coloring, then merging back these mixed hypertrees with the necessary adjustment in the coloring common vertex v will give a proper coloring of $\mathcal{H}$; thus, $\overline{\chi}(\mathcal{H}) \geq \overline{\chi}(\mathcal{H}') + \overline{\chi}(\mathcal{T}_i)$.

On the other hand, suppose we have a maximum coloring of $\mathcal{H}$ with $\overline{\chi}(\mathcal{H})$ colors. Now decompose $\mathcal{H}$ into $\mathcal{H}'$ and $\mathcal{T}_i$, preserving the given maximum coloring. Then this is a coloring of $\mathcal{H}'$ and of $\mathcal{T}_i$; thus, we have $\overline{\chi}(\mathcal{H}) \leq \overline{\chi}(\mathcal{H}') + \overline{\chi}(\mathcal{T}_i) - 1$.

□

3 Examples

First, we exhibit a simple mixed hypertree $\mathcal{H} = (X, \mathcal{C}, \mathcal{D})$ in which the difference between $\overline{\chi}(\mathcal{H})$ and $n - s$ is 1, where $n = |X|$ and $s = s(\mathcal{H})$.

Example 1. Consider the mixed hypertree $\mathcal{H} = (X, \mathcal{C}, \mathcal{D})$, where $X = \{x_0, x_1, \ldots, x_5\}$,

$$\mathcal{C} = \{\{x_0, x_1, x_2\}, \{x_0, x_2, x_3\}, \{x_0, x_3, x_4\}, \{x_0, x_4, x_5\}, \{x_0, x_5, x_1\}\},$$

and $\mathcal{D} = \emptyset$. Here, $\bar{\chi}(\mathcal{H}) = 3$, $|X| = n = 6$, and the sieve number $s = 2$. Thus,

$$\bar{\chi}(\mathcal{H}) = 3 \neq 4 = n - s.$$

This example can also be generalized to similar mixed hypergraphs with an odd number of such "satellite vertices" (the vertices that are not the central vertex) as to make the difference between $\overline{\chi}(\mathcal{H})$ and $|X| - s$ as large as desired. Here is how: Start with the mixed hypertree $\mathcal{H}$ in example 1 above. Then, pick any of the satellite vertices and make it the satellite vertex of an added-on copy of $\mathcal{H}$. The resulting mixed hypertree – call it $\mathcal{H}_1$ – will have $\overline{\chi}(\mathcal{H})$ - ($|X| - s) = 2$. To make the difference equal to 3, one could pick one of the new satellite vertices from $\mathcal{H}_1$ and make it the satellite vertex of an added-on copy of $\mathcal{H}$. Continuing in this fashion, one can see how to make the difference arbitrarily large.

References

[1] E. Bulgaru, V.I. Voloshin. *Mixed interval hypergraphs*, Discrete Applied Math., 77(1) (1997), pp. 24–41.

[2] D. Kral, J. Kratochvil, A. Proskurowski, H.-J. Voss. *Coloring mixed hypertrees.* Proceedings, 26th Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science vol. 1928, pp. 279–289, Springer-Verlag, 2000.

[3] A. Niculitsa, V.I. Voloshin. *About uniquely colorable mixed hypertrees.* Discussione Mathematicae: Graph Theory. Vol. 20, No. 1, 2000, pp.81–91.

[4] Zs. Tuza, V.I. Voloshin. *Uncolorable mixed hypergraphs.* Discrete Applied Math., **99** (2000), pp. 209–227.

[5] V.I. Voloshin. *The mixed hypergraphs.* Computer Science J. Moldova **1** (1993), pp. 45–52.

[6] V.I. Voloshin. *On the upper chromatic number of a hypergraph,* Australasian J. Comb. **11** (1995), pp. 25–45.

[7] V.I. Voloshin. *Coloring Mixed Hypergraphs: Theory, Algorithms and Applications.* Fields Institute Monograph, American Mathematical Society, 2002.

K. Roblee, V. Voloshin,
Received July 25, 2005

Department of Mathematics and Physics,
Troy University
Troy, Alabama, USA
E–mail: *vvoloshin@troy.edu*

Nash equilibria sets in mixed extended 2×3 games

Valeriu Ungureanu, Ana Botnari

Abstract

We describe the Nash equilibria set as an intersection of best response graphs. The problem of Nash equilibria set construction for two-person mixed extended 2×3 games is studied.

Mathematics Subject Classification 2000: 91A05, 91A06, 91A10, 91A43, 91A44.

Keywords and phrases: Noncooperative game; Nash equilibrium, Nash equilibria set, best response graph.

1 Introduction

We construct the Nash equilibria set as an intersection of best response graphs [4, 5]. This paper may be considered a continuation of [5] and it has to illustrate the practical opportunity of a mentioned characteristic.

Consider a noncooperative game:

$$\Gamma = \langle N, \{X_i\}_{i \in N}, \{f_i(x)\}_{i \in N} \rangle,$$

where $N = \{1, 2, ..., n\}$ is a set of players, X_i is a set of strategies of player $i \in N$ and $f_i : X \to R$ is a player's $i \in N$ payoff function defined on the Cartesian product $X = \times_{i \in N} X_i$. Elements of X are named outcomes of the game (situations or strategy profiles).

The outcome $x^* \in X$ of the game is the Nash equilibrium [3] (shortly NE) of Γ if

$$f_i(x_i, x_{-i}^*) \leq f_i(x_i^*, x_{-i}^*), \forall x_i \in X_i, \ \forall i \in N,$$

where

$$x^*_{-i} = (x^*_1, x^*_2, ..., x^*_{i-1}, x^*_{i+1}, ..., x^*_n),$$

$$x^*_{-i} \in X_{-i} = X_1 \times X_2 \times ... \times X_{i-1} \times X_{i+1} \times ... \times X_n,$$

$$(x_i, x^*_{-i}) = (x^*_1, x^*_2, ..., x^*_{i-1}, x_i, x^*_{i+1}, ..., x^*_n) \in X.$$

There are diverse alternative formulations of a Nash equilibrium [1]: as a fixed point of the best response correspondence, as a fixed point of a function, as a solution of a non-linear complementarity problem, as a solution of a stationary point problem, as a minimum of a function on a polytope, as a semi-algebraic set. We study the Nash equilibria set as an intersection of best response graphs [4, 5], i.e. intersection of the sets:

$$Gr_i = \{(x_i, x_{-i}) \in X : x_{-i} \in X_{-i}, x_i \in \text{Arg} \max_{x_i \in X_i} f_i(x_i, x_{-i})\},\ i \in N.$$

From the players views not all Nash equilibria are equally attractive. They may be Pareto ranked. Therefore Nash equilibrium may dominate or it may be dominated. There are also different other criteria for Nash equilibria distinguishing such as perfect equilibria, proper equilibria, sequential equilibria, stable sets etc. Thus the methods that found only a sample of Nash equilibrium don't guarantee that determined Nash equilibrium complies all the players demands and refinement conditions. Evidently, a method for all Nash equilibria determination is useful and required. Other theoretical and practical factors that argue for NE set determination exist [1].

This paper as the continuation of [5] investigates the problems of NE set construction in the games that permit simple graphic illustrations and that elucidate the usefulness of the interpretation of NE as an intersection of best response graphs [4, 5].

2 Main results

Consider a two-person matrix game Γ with matrices:

$$A = (a_{ij}),\ B = (b_{ij}),\ i = \overline{1,2},\ j = \overline{1,3}.$$

The game $\Gamma_m = \langle\{1,2\}; X, Y; f_1, f_2\rangle$ is the mixed extension of Γ, where

$X = \{\mathbf{x} = (x_1, x_2) \in R^2 : x_1 + x_2 = 1, x_1 \geq 0, x_2 \geq 0\}$,

$Y = \{\mathbf{y} = (y_1, y_2, y_3) \in R^3 : y_1 + y_2 + y_3 = 1, y_1 \geq 0, y_2 \geq 0, y_3 \geq 0\}$,

$f_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{2} \sum_{j=1}^{3} a_{ij} x_i y_j$,

$f_2(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{2} \sum_{j=1}^{3} b_{ij} x_i y_j$.

This game is reduced to the game on the unit prism. For the reduced game the class partition of the strategy sets is considered and the NE set is determined for each possible "subgame" (see the following propositions).

2.1 Reduction to game on a prism

By substitutions:

$x_1 = x$, $x_2 = 1 - x$, $x \in [0,1]$,

$y_3 = 1 - y_1 - y_2, \quad y_3 \in [0,1]$,

the game Γ_m is reduced to the equivalent game:

$$\Gamma'_m = \langle\{1,2\}; [0,1], \triangle; \varphi_1, \varphi_2\rangle,$$

where

$\triangle = \{\mathbf{y} = (y_1, y_2) \in R^2 : y_1 + y_2 \leq 1, y_1 \geq 0, y_2 \geq 0\}$,

$$\begin{aligned}\varphi_1(x, \mathbf{y}) = {} & (a_{11}y_1 + a_{12}y_2 + a_{13}(1 - y_1 - y_2))x + \\ & (a_{21}y_1 + a_{22}y_2 + a_{23}(1 - y_1 - y_2))(1 - x) = \\ & ((a_{11} - a_{21} + a_{23} - a_{13})y_1 + (a_{12} - a_{22} + a_{23} - a_{13})y_2 + a_{13} - a_{23})x + \\ & (a_{21} - a_{23})y_1 + (a_{22} - a_{23})y_2 + a_{23} = \\ & ((a_{11} - a_{21})y_1 + (a_{12} - a_{22})y_2 + (a_{13} - a_{23})(1 - y_1 - y_2))\, x + \\ & nbb(a_{21} - a_{23})y_1 + (a_{22} - a_{23})y_2 + a_{23};\end{aligned}$$

$$\begin{aligned}\varphi_2(x, \mathbf{y}) = {} & (b_{11}y_1 + b_{12}y_2 + b_{13}(1 - y_1 - y_2))x + \\ & (b_{21}y_1 + b_{22}y_2 + b_{23}(1 - y_1 - y_2))(1 - x) = \\ & ((b_{11} - b_{13} + b_{23} - b_{21})x + b_{21} - b_{23})y_1 + \\ & ((b_{12} - b_{13} + b_{23} - b_{22})x + b_{22} - b_{23})y_2 + \\ & (b_{13} - b_{23})x + b_{23} = \\ & ((b_{11} - b_{13})x + (b_{21} - b_{23})(1 - x))y_1 + \\ & ((b_{12} - b_{13})x + (b_{22} - b_{23})(1 - x))y_2 + \\ & (b_{13} - b_{23})x + b_{23}.\end{aligned}$$

Thus, Γ_m is reduced to the game Γ'_m on the prism $\Pi = [0, 1] \times \triangle$. If $NE(\Gamma'_m)$ is known, then it is easy to construct the set $NE(\Gamma_m)$. Basing on properties of strategies of each player of the initial pure strategies game Γ, diverse classes of games are considered and for every class the sets $NE(\Gamma'_m)$ are determined.

For commodity, we use notation:

$$\triangle_= = \{\mathbf{y} = (y_1, y_2) \in R^2 : y_1 + y_2 = 1, y_1 \geq 0, y_2 \geq 0\}.$$

2.2 Both players have either equivalent strategies or dominant strategies

Proposition 1. *If all the players have equivalent strategies, then* $NE(\Gamma'_m) = \Pi$.

Proof. From the equivalence of strategies

$$\varphi_1(x, \mathbf{y}) = (a_{21} - a_{23})y_1 + (a_{22} - a_{23})y_2 + a_{23},$$

$$\varphi_2(x, \mathbf{y}) = (b_{13} - b_{23})x + b_{23}.$$

From this the truth of the proposition results. □

Proposition 2. *If all the players have dominant strategies in* Γ, *then:*

$$NE(\Gamma'_m) = \begin{cases} (0,0,0) & \textit{if strategies (2,3) are dominant,} \\ (0,0,1) & \textit{if strategies (2,2) are dominant,} \\ (0,1,0) & \textit{if strategies (2,1) are dominant,} \\ 0\times\Delta_= & \textit{if strategies (2,1} \sim \textit{2) are dominant,} \\ 0\times[0,1]\times 0 & \textit{if strategies (2,1} \sim \textit{3) are dominant,} \\ 0\times 0\times[0,1] & \textit{if strategies (2,2} \sim \textit{3) are dominant,} \\ (1,0,0) & \textit{if strategies (1,3) are dominant,} \\ (1,0,1) & \textit{if strategies (1,2) are dominant,} \\ (1,1,0) & \textit{if strategies (1,1) are dominant,} \\ 1\times\Delta_= & \textit{if strategies (1,1} \sim \textit{2) are dominant,} \\ 1\times[0,1]\times 0 & \textit{if strategies (1,1} \sim \textit{3) are dominant,} \\ 1\times 0\times[0,1] & \textit{if strategies (1,2} \sim \textit{3) are dominant.} \end{cases}$$

Proof. It is easy to observe that

$$\operatorname{Arg}\max_{x\in[0,1]} \varphi_1(x,\mathbf{y}) = \begin{cases} 1 & \text{if the 1-st strategy is dominant in } \Gamma, \\ 0 & \text{if the 2-nd strategy is dominant in } \Gamma, \end{cases}$$

$\forall \mathbf{y} \in \triangle$. Hence,

$$Gr_1 = \begin{cases} 1\times\triangle & \text{if the 1-st strategy is dominant,} \\ 0\times\triangle & \text{if the 2-nd strategy is dominant.} \end{cases}$$

For the second player:

$$\operatorname{Arg}\max_{\mathbf{y}\in\triangle} \varphi_2(x,\mathbf{y}) = \begin{cases} (1,0) & \text{if the 1-st strategy is dominant in } \Gamma, \\ (0,1) & \text{if the 2-nd strategy is dominant in } \Gamma, \\ (0,0) & \text{if the 3-rd strategy is dominant in } \Gamma, \\ \Delta_= & \text{if strategies } 1\sim 2 \text{ dominate } 3, \\ [0,1]\times 0 & \text{if strategies } 1\sim 3 \text{ dominate } 2, \\ 0\times[0,1] & \text{if strategies } 2\sim 3 \text{ dominate } 1, \end{cases}$$

$\forall \mathbf{x} \in [0,1]$. Hence,

$$Gr_2 = \begin{cases} [0,1]\times(1,0) & \text{if the 1-st strategy is dominant,} \\ [0,1]\times(0,1) & \text{if the 2-nd strategy is dominant,} \\ [0,1]\times(0,0) & \text{if the 3-rd strategy is dominant.} \\ [0,1]\times\Delta_= & \text{if strategies } 1 \sim 2 \text{ dominate } 3, \\ [0,1]\times[0,1]\times 0 & \text{if strategies } 1 \sim 3 \text{ dominate } 2, \\ [0,1]\times 0\times[0,1] & \text{if strategies } 2 \sim 3 \text{ dominate } 1. \end{cases}$$

Thus, the NE set contains either only one vertex of a unit prism $\triangle$ as an intersection of one facet Gr_1 with one edge Gr_2 or only one edge of a unit prism $\triangle$ as an intersection of one facet Gr_1 with one edge Gr_2. □

2.3 One player has dominant strategy

Proposition 3A. *If the 1-st strategy of the first player is dominant, then*

$$NE(\Gamma'_m) = \begin{cases} (1,1,0) & \textit{if } b_{11} > \max\{b_{12}, b_{13}\}, \\ (1,0,1) & \textit{if } b_{12} > \max\{b_{11}, b_{13}\}, \\ (1,0,0) & \textit{if } b_{12} > \max\{b_{11}, b_{13}\}, \\ 1\times\triangle_= & \textit{if } b_{11} = b_{12} > b_{13}, \\ 1\times[0,1]\times 0 & \textit{if } b_{11} = b_{13} > b_{12}, \\ 1\times 0\times[0,1] & \textit{if } b_{12} = b_{13} > b_{11}, \\ 1\times\triangle & \textit{if } b_{12} = b_{13} = b_{11}, \end{cases}$$

if the 2-nd strategy of the first player is dominant, then

$$NE(\Gamma'_m) = \begin{cases} (0,1,0) & \textit{if } b_{11} > \max\{b_{12}, b_{13}\}, \\ (0,0,1) & \textit{if } b_{12} > \max\{b_{11}, b_{13}\}, \\ (0,0,0) & \textit{if } b_{12} > \max\{b_{11}, b_{13}\}, \\ 0\times\triangle_= & \textit{if } b_{11} = b_{12} > b_{13}, \\ 0\times[0,1]\times 0 & \textit{if } b_{11} = b_{13} > b_{12}, \\ 0\times 0\times[0,1] & \textit{if } b_{12} = b_{13} > b_{11}, \\ 0\times\triangle & \textit{if } b_{12} = b_{13} = b_{11}. \end{cases}$$

Proof. If the first player has dominant strategy, then

$$Gr_1 = \begin{cases} 1 \times \triangle & \text{if the 1-st strategy is dominant,} \\ 0 \times \triangle & \text{if the 2-nd strategy is dominant} \end{cases}$$

is one triangle facet of the prism.

If the 1-st strategy of the first player is dominant, then

$$\varphi_2(1, \mathbf{y}) = (b_{11} - b_{13})y_1 + (b_{12} - b_{13})y_2 + b_{13} =$$

$$= b_{11}y_1 + b_{12}y_2 + b_{13}(1 - y_1 - y_2).$$

From this we obtain that

$$\operatorname{Arg}\max_{\mathbf{y}\in\triangle} \varphi_2(1, \mathbf{y}) = \begin{cases} (1,0) & \text{if } b_{11} > \max\{b_{12}, b_{13}\}, \\ (0,1) & \text{if } b_{12} > \max\{b_{11}, b_{13}\}, \\ (0,0) & \text{if } b_{12} > \max\{b_{11}, b_{13}\}, \\ \triangle_= & \text{if } b_{11} = b_{12} > b_{13}, \\ [0,1]\times 0 & \text{if } b_{11} = b_{13} > b_{12}, \\ 0 \times [0,1] & \text{if } b_{12} = b_{13} > b_{11}, \\ \triangle & \text{if } b_{12} = b_{13} = b_{11} \end{cases}$$

and

$$Gr_2 = 1 \times \operatorname{Arg}\max_{\mathbf{y}\in\triangle} \varphi_2(1, \mathbf{y})$$

is a vertex, edge or triangle facet of the prism Π. Hence, the truth of the first part of proposition follows.

Analogically the proposition can be proved when the second strategy is dominant. □

Proposition 3B. *If the second player has only one dominant strat-*

egy, then

$$NE(\Gamma'_m) = \begin{cases} (0,1,0) & \textit{if } (\cdot,1) \textit{ is dominant and } a_{11} < a_{21}, \\ (1,1,0) & \textit{if } (\cdot,1) \textit{ is dominant and } a_{11} > a_{21}, \\ [0,1]\times 1\times 0 & \textit{if } (\cdot,1) \textit{ is dominant and } a_{11} = a_{21}, \\ (0,0,1) & \textit{if } (\cdot,2) \textit{ is dominant and } a_{12} < a_{22}, \\ (1,0,1) & \textit{if } (\cdot,2) \textit{ is dominant and } a_{12} > a_{22}, \\ [0,1]\times 0\times 1 & \textit{if } (\cdot,2) \textit{ is dominant and } a_{12} = a_{22}, \\ (0,0,0) & \textit{if } (\cdot,3) \textit{ is dominant and } a_{13} < a_{23}, \\ (1,0,0) & \textit{if } (\cdot,3) \textit{ is dominant and } a_{13} > a_{23}, \\ [0,1]\times 0\times 0 & \textit{if } (\cdot,3) \textit{ is dominant and } a_{13} = a_{23}. \end{cases}$$

Proof. If the 3-rd strategy of the second player is dominant, then

$$\operatorname{Arg}\max_{\mathbf{y}\in\triangle} \varphi_2(x,\mathbf{y}) = \operatorname{Arg}\max_{\mathbf{y}\in\triangle}((b_{11}-b_{13})x + (b_{21}-b_{23})(1-x))y_1 +$$

$$((b_{12}-b_{13})x + (b_{22}-b_{23})(1-x))y_2 + (b_{13}-b_{23})x + b_{23} = (0,0),$$

and

$$Gr_2 = [0,1]\times(0,0)$$

is an edge of a prism Π.

For the first player, we obtain $\varphi_1(x,\mathbf{0}) = a_{13}x + a_{23}(1-x)$ and

$$Gr_1 = \begin{cases} (1,0,0) & \text{if } a_{13} > a_{13}, \\ (0,0,0) & \text{if } a_{13} < a_{13}, \\ [0,1]\times 0\times 0 & \text{if } a_{13} = a_{13}. \end{cases}$$

Consequently, NE set is a vertex or edge of the prism Π.

Similarly, the remained part of the proposition can be proved in the other two subcases. □

Proposition 3C. *If the second player has two dominant strategies, then*

$$NE(\Gamma'_m) = Gr_1 \cap Gr_2,$$

where:

$$Gr_1 = 0\times Y_{12}^{<} \cup 1\times Y_{12}^{>} \cup [0,1]\times Y_{12}^{=}$$

$$Gr_2 = [0,1] \times \Delta_=$$

if the 1-st and 2-nd strategies are equivalent and they dominate the 3-rd strategy;

$$Gr_1 = 0 \times Y_1^< \cup 1 \times Y_1^> \cup [0,1] \times Y_1^=$$

$$Gr_2 = [0,1] \times [0,1] \times 0$$

if the 1-st and 3-rd strategies are equivalent and they dominate the 2-nd strategy;

$$Gr_1 = 0 \times Y_2^< \cup 1 \times Y_2^> \cup [0,1] \times Y_2^=$$

$$Gr_2 = [0,1] \times 0 \times [0,1]$$

if the 2-nd and 3-rd strategies are equivalent and they dominate the 1-st strategy;

$$\begin{aligned}
Y_{12}^< &= \{\mathbf{y} \in \triangle_= : (a_{11} - a_{21})y_1 + (a_{12} - a_{22})y_2 < 0\},\\
Y_{12}^> &= \{\mathbf{y} \in \triangle_= : (a_{11} - a_{21})y_1 + (a_{12} - a_{22})y_2 > 0\},\\
Y_{12}^= &= \{\mathbf{y} \in \triangle_= : (a_{11} - a_{21})y_1 + (a_{12} - a_{22})y_2 = 0\},\\
Y_1^< &= \{\mathbf{y} \in R^2 : (a_{11} - a_{21} + a_{23} - a_{13})y_1 + a_{13} - a_{23} < 0,\\
&\qquad y_1 \in [0,1], y_2 = 0\},\\
Y_1^> &= \{\mathbf{y} \in R^2 : (a_{11} - a_{21} + a_{23} - a_{13})y_1 + a_{13} - a_{23} > 0,\\
&\qquad y_1 \in [0,1], y_2 = 0\},\\
Y_1^= &= \{\mathbf{y} \in R^2 : (a_{11} - a_{21} + a_{23} - a_{13})y_1 + a_{13} - a_{23} = 0,\\
&\qquad y_1 \in [0,1], y_2 = 0\},\\
Y_2^< &= \{\mathbf{y} \in R^2 : (a_{12} - a_{22} + a_{23} - a_{13})y_2 + a_{13} - a_{23} < 0,\\
&\qquad y_2 \in [0,1], y_1 = 0\},\\
Y_2^> &= \{\mathbf{y} \in R^2 : (a_{12} - a_{22} + a_{23} - a_{13})y_2 + a_{13} - a_{23} > 0,\\
&\qquad y_2 \in [0,1], y_1 = 0\},\\
Y_2^= &= \{\mathbf{y} \in R^2 : (a_{12} - a_{22} + a_{23} - a_{13})y_2 + a_{13} - a_{23} = 0,\\
&\qquad y_2 \in [0,1], y_1 = 0\}.
\end{aligned}$$

Proof. If the 1-st and 2-nd strategies of the second player are equivalent and they dominate the third strategy, then

$$\operatorname{Arg}\max_{\mathbf{y}\in\triangle} \varphi_2(x,\mathbf{y}) = \operatorname{Arg}\max_{\mathbf{y}\in\triangle}((b_{11}-b_{13})x + (b_{21}-b_{23})(1-x))(y_1+y_2)+$$

$$+(b_{13} - b_{23})x + b_{23} = \Delta_=$$

and

$$Gr_2 = [0,1] \times \Delta_=$$

is a facet of a prism Π.

For the first player, we obtain

$$\begin{aligned}\varphi_1(x, \mathbf{y}) = &((a_{11} - a_{21})y_1 + (a_{12} - a_{22})y_2)\, x + \\ &+ (a_{21} - a_{23})y_1 + (a_{22} - a_{23})y_2 + a_{23};\end{aligned}$$

and

$$Gr_1 = 0 \times Y_{12}^{<} \cup 1 \times Y_{12}^{>} \cup [0,1] \times Y_{12}^{=}$$

where:

$$\begin{aligned}Y_{12}^{<} &= \{\mathbf{y} \in \triangle_= : (a_{11} - a_{21})y_1 + (a_{12} - a_{22})y_2 < 0\},\\ Y_{12}^{>} &= \{\mathbf{y} \in \triangle_= : (a_{11} - a_{21})y_1 + (a_{12} - a_{22})y_2 > 0\},\\ Y_{12}^{=} &= \{\mathbf{y} \in \triangle_= : (a_{11} - a_{21})y_1 + (a_{12} - a_{22})y_2 = 0\}.\end{aligned}$$

Similarly, the remained part of the proposition can be proved in the other two subcases. $\square$

Evidently, propositions 3A, 3B and 3C elucidate the case when one player has dominant strategy (strategies) and the other player has equivalent strategies.

2.4 One player has equivalent strategies

Proposition 4A. *If the first player has equivalent strategies, then*

$$NE(\Gamma'_m) = Gr_2,$$

$$\begin{aligned}Gr_2 = &X_1 \times (1,0) \cup X_2 \times (0,1) \cup X_3 \times (0,0) \cup \\ &X_{12} \times \triangle_= \cup X_{13} \times [0,1] \times 0 \cup X_{23} \times 0 \times [0,1] \cup \\ &X_{123} \times \triangle,\end{aligned}$$

where:

$$X_1 = \left\{ x \in [0,1] : \begin{array}{l} (b_{11} - b_{21})x + b_{21} > (b_{12} - b_{22})x + b_{22} \\ (b_{11} - b_{21})x + b_{21} > (b_{13} - b_{23})x + b_{23} \end{array} \right\},$$

$$X_2 = \left\{ x \in [0,1] : \begin{array}{l} (b_{12} - b_{22})x + b_{22} > (b_{11} - b_{21})x + b_{21} \\ (b_{12} - b_{22})x + b_{22} > (b_{13} - b_{23})x + b_{23} \end{array} \right\},$$

$$X_3 = \left\{ x \in [0,1] : \begin{array}{l} (b_{13} - b_{23})x + b_{23} > (b_{11} - b_{21})x + b_{21} \\ (b_{13} - b_{23})x + b_{23} > (b_{12} - b_{22})x + b_{22} \end{array} \right\},$$

$$X_{12} = \left\{ x \in [0,1] : \begin{array}{l} (b_{11} - b_{21})x + b_{21} = (b_{12} - b_{22})x + b_{22} \\ (b_{11} - b_{21})x + b_{21} > (b_{13} - b_{23})x + b_{23} \end{array} \right\},$$

$$X_{13} = \left\{ x \in [0,1] : \begin{array}{l} (b_{11} - b_{21})x + b_{21} > (b_{12} - b_{22})x + b_{22} \\ (b_{11} - b_{21})x + b_{21} = (b_{13} - b_{23})x + b_{23} \end{array} \right\},$$

$$X_{23} = \left\{ x \in [0,1] : \begin{array}{l} (b_{12} - b_{22})x + b_{22} > (b_{11} - b_{21})x + b_{21} \\ (b_{12} - b_{22})x + b_{22} = (b_{13} - b_{23})x + b_{23} \end{array} \right\},$$

$$X_{123} = \left\{ x \in [0,1] : \begin{array}{l} (b_{11} - b_{21})x + b_{21} = (b_{12} - b_{22})x + b_{22} \\ (b_{11} - b_{21})x + b_{21} = (b_{13} - b_{23})x + b_{23} \end{array} \right\}.$$

Proof. If the strategies of the first player are equivalent, then $Gr_1 = \Pi$.

Suppose that $x \in [0,1]$ is fixed. The payoff function of the second player can be represented in the form

$$\begin{aligned} \varphi_2(x, \mathbf{y}) = {} & ((b_{11} - b_{21})x + b_{21})y_1 + ((b_{12} - b_{22})x + b_{22})y_2 + \\ & ((b_{13} - b_{23})x + b_{23})(1 - y_1 - y_2). \end{aligned}$$

It's evident that for:

$x \in X_1$ the minimum of the cost function is realized on $(1,0) \in \triangle$,

$x \in X_2$ the minimum is realized on $(0,1) \in \triangle$,

$x \in X_3$ the minimum is realized on $(0,0) \in \triangle$,

$x \in X_{12}$ the minimum is realized on $\triangle_=$,

$x \in X_{13}$ the minimum is realized on $[0,1] \times 0 \in \triangle$,

$x \in X_{23}$ the minimum is realized on $0 \times [0,1] \in \triangle$,

$x \in X_{123}$ the minimum is realized on $\triangle$.

From the above the truth of the proposition follows. $\square$

Proposition 4B. *If all three strategies of the second player are equivalent, then*

$$NE(\Gamma'_m) = Gr_1 = 1 \times Y_1 \cup 0 \times Y_2 \cup [0,1] \times Y_{12},$$

where

$$\begin{aligned}
&Y_1 = \{\mathbf{y} \in \triangle : \alpha_1 y_1 + \alpha_2 y_2 + \alpha_3 > 0\},\\
&Y_2 = \{\mathbf{y} \in \triangle : \alpha_1 y_1 + \alpha_2 y_2 + \alpha_3 < 0\},\\
&Y_{12} = \{\mathbf{y} \in \triangle : \alpha_1 y_1 + \alpha_2 y_2 + \alpha_3 = 0\},\\
&\alpha_1 = a_{11} - a_{13} + a_{23} - a_{21},\\
&\alpha_2 = a_{12} - a_{13} + a_{23} - a_{22},\\
&\alpha_3 = a_{13} - a_{23}.
\end{aligned}$$

Proof. If all three strategies of the second player are equivalent, then $b_{11} = b_{12} = b_{13}$, $b_{21} = b_{22} = b_{23}$ and $Gr_2 = \Pi$.

The cost function of the first player can be represented in the following form

$$\begin{aligned}
\varphi_1(x, \mathbf{y}) = ((a_{11} - a_{13})y_1 + (a_{12} - a_{13})y_2 + a_{13})x +\\
((a_{21} - a_{23})y_1 + (a_{22} - a_{23})y_2 + a_{23})(1 - x).
\end{aligned}$$

It's evident that for

$$\begin{aligned}
\mathbf{y} \in Y_1 = \{\mathbf{y} \in \triangle : a_{11}y_1 + a_{12}y_2 + a_{13}(1 - y_1 - y_2) >\\
> a_{21}y_1 + a_{22}y_2 + a_{23}(1 - y_1 - y_2)\ \} =\\
= \{\mathbf{y} \in \triangle : (a_{11} - a_{13} + a_{23} - a_{21})y_1 +\\
+ (a_{12} - a_{13} + a_{23} - a_{22})y_2 + a_{13} - a_{23} > 0\}
\end{aligned}$$

the 1-st strategy of the first player is optimal, for

$$\begin{aligned}
\mathbf{y} \in Y_2 = \{\mathbf{y} \in \triangle : a_{11}y_1 + a_{12}y_2 + a_{13}(1 - y_1 - y_2) <\\
< a_{21}y_1 + a_{22}y_2 + a_{23}(1 - y_1 - y_2)\ \} =\\
= \{\mathbf{y} \in \triangle : (a_{11} - a_{13} + a_{23} - a_{21})y_1 +\\
+ (a_{12} - a_{13} + a_{23} - a_{22})y_2 + a_{13} - a_{23} < 0\}
\end{aligned}$$

the 2-nd strategy of the first player is optimal, and for

$$\begin{aligned}\mathbf{y} \in Y_{12} = \{\mathbf{y} \in \triangle : a_{11}y_1 + a_{12}y_2 + a_{13}(1 - y_1 - y_2) = \\ = a_{21}y_1 + a_{22}y_2 + a_{23}(1 - y_1 - y_2)\ \} = \\ = \{\mathbf{y} \in \triangle : (a_{11} - a_{13} + a_{23} - a_{21})y_1 + \\ + (a_{12} - a_{13} + a_{23} - a_{22})y_2 + a_{13} - a_{23} = 0\}\end{aligned}$$

every strategy $x \in [0, 1]$ of the first player is optimal. From this, the truth of the proposition follows. □

2.5 The players don't have dominant strategies

Proposition 5. *If the both players don't have dominant strategies, then*

$$NE(\Gamma'_m) = Gr_1 \cap Gr_2,$$

where Gr_1, Gr_2 are defined as in propositions 4A, 4B.

The truth of the proposition follows from the above.

2.6 Algorithm

From the above a simple solving procedure follows. In this procedure only one step from **1°** to **5°** is executed.

0° The game Γ'_m is considered (see subsection 2.1);

1° If the both players have equivalent strategies in Γ, then the NE set in Γ_m is $X \times Y$ (see the proposition 1);

2° If the both players have dominant strategies in Γ, then the NE set in Γ_m is constructed in compliance with proposition 2 and substitutions of subsection 2.1;

3A° If only the first player has dominant strategy in Γ, then the NE set in Γ_m is constructed in conformity with proposition 3A and substitutions of subsection 2.1;

3B° If only the second player has only one dominant strategy in Γ, then the NE set in Γ_m is constructed in conformity with proposition 3B and substitutions of subsection 2.1;

3C° If the second player has two dominant strategies that dominate the other strategy in Γ, then the NE set in Γ_m is constructed in conformity with proposition 3B and substitutions of subsection 2.1;

4A° If only the first player has equivalent strategies in Γ, then the NE set in Γ_m is constructed in accordance with proposition 4A and substitutions of subsection 2.1;

4B° If only the second player has equivalent strategies in Γ, then the NE set in Γ_m is constructed in accordance with proposition 4B and substitutions of subsection 2.1;

5° If the both players don't have dominant strategies in Γ, then the NE set in Γ_m is constructed in compliance with proposition 5 and substitutions of subsection 2.1.

References

[1] McKelvey R.D., McLenan A., *Computation of equilibria in finite games,* Handbook of Computational Economics, 1, Amsterdam, Elsevier, 1996, pp. 87-142.

[2] Moulen H., *Théorie des jeux pour l'éqonomie et la politique,* Paris, 1981 (in Russian: Games Theory, Moscow, Mir, 1985, 200 p.).

[3] Nash J.F., *Noncooperative game,* Annals of Mathematics, 54, 1951, pp. 280-295.

[4] Sagaidac M., Ungureanu V., *Operational research,* Chişinău, CEP USM, 2004, 296 p. (in Romanian).

[5] Ungureanu V., Botnari A., *Nash equilibria sets in mixed extension of* $2 \times 2 \times 2$ *games*, Computer Science Journal of Moldova, 1, 2005, pp. 1-15.

V. Ungureanu, A. Botnari,

Received June 23, 2005

State University of Moldova,
60, A. Mateevici str.,
Chişinău, MD−2009, Moldova.
E–mail: *valungureanu@usm.md*

Computer Science Journal of Moldova, vol.**13**, no.2(38), 2005

Involving d-Convex Simple and Quasi-simple Planar Graphs in $\mathbb{R}^3$

Nadejda Sur Sergiu Cataranciuc

Abstract

The problem of finding dimension of d-convex simple and quasi-simple planar graphs is studied. Algorithms for involving these graphs in $\mathbb{R}^3$ are described.

1 Prelinimary Considerations

A connected graph can be considered as a metric space. Elements of this metric space are vertexes of the graph; distance between two vertexes is minimal length of chains between them. If we denote through $d(x,\, y)$ distance between two vertexes x, y of graph $G = (X;\, U)$, then this graph defines a discrete metric space $(X;\, d)$. Let X' be a finite space, where a metric of whole values ρ is defined. It is known that independently of metric ρ there always exists a graph $G = (X;\, U)$ such that $X' \subseteq X$ and distance among vertexes of subset X' in G coincides with distance ρ of a space X' (see [1]). In this case we say that metric ρ is involved in graph G or metric ρ is realized in graph G. A lot of results about realization of metric in special graphs are exposed in [2] and [3].

It has also practical and theoretical interest the mutual problem. We will say that a graph $G = (X;\, U)$ is involved in a metric space X', if there exists an application $\varphi : X \to X'$, such that any two adjacent vertexes x, y in G have the images $\varphi(x)$ and $\varphi(y)$ in X', that are of distance equal to 1. In other words, adjacent vertexes of graph G are transformed into elements of distance 1 of space X'. The minimal dimension of an Euclidean space, where a graph G can be involved is called *dimension of graph G* and denoted *dim G*. The problem itself can be formulated as follows: for any graph $G = (X;\, U)$ to find its dimension. Of course in this case we can talk about elaboration of an algorithm that will be allowed to compute dimension of a non-oriented graph.

In general case, the formulated problem is hard enough. In this article we will give some results that refer to finding dimension of some special graphs: *d-convex simple and d-convex quasi-simple graphs.*

2 Dimension of d-Convex Simple Planar Graphs

By definition, a subset of vertexes $A \subset X$ of a graph $G = (X;\, U)$ is called *d-convex* if for each two vertexes $x,\, y \in A$, the following relation is true:

$$< x,\, y >= \{z \in X;\, d(x,\, z) + d(z,\, y) = d(x,\, y)\} \subset A \ (see[4]).$$

Definition 1 [5]. *A non-oriented graph $G = (X, U)$ is called d-convex simple if any subset of vertexes $A \subset X$, $2 < |A| < |X|$ is not d-convex.*

Since in any graph the empty set, the set formed of 1 vertex, the set formed of two adjacent vertexes and the set of all vertexes are d-convex, the result is that d-convex graphs are graphs with minimal number of d-convex sets. If we denote the family of all d-convex sets of cardinal k by D_k, and the set of all d-convex sets of a graph with n vertexes and m edges by D, so that

$$D = \bigcup_{k=0}^{n} D_k,$$

then

$$|D| \geq |D_0| + |D_1| + |D_2| + |D_n| = 1 + n + m + 1 = n + m + 2.$$

In case of d-convex graph here we have always equality $|D| = n+m+2$. Let us denote by $\Gamma(x)$ the neighborhood of vertex x, i. e. $\Gamma(x) = \{y \in X | x \sim y\}$.

Definition 2 [5]. *A vertex y is called copy for vertex x ($x \neq y$), in graph $G = (X;\, U)$ if $\Gamma(x) = \Gamma(y)$.*

Let T be a tree with at least 3 vertexes and T_0 a sub-graph of T, that consists of all vertexes and edges of T, without those suspended. So, for each unsuspended vertex x from T, we have a uniquely correspondent vertex $\bar{x}$ from T_0, and for each vertex of T_0 we have a uniquely correspondent vertex from T. Let $L(T,\, T_0)$ be a graph obtained from T, T_0 and by adding the following edges: every vertex $\bar{x}$ of T_0 will be adjacent with all vertexes from $\Gamma(x)$ from T, where x and $\bar{x}$ are correspondent vertexes. It is easy to see that in graph $L(T,\, T_0)$ every vertex of degree at least 3 has a unique copy and there are no suspended vertexes.

The next theorem is true:

Theorem 1 [6]. *If T is a tree with at least 3 vertexes, then graph $G = L(T,\, T_0)$ is d-convex simple and planar.*

As follows from [6] the graph described above has deal with a class of d-convex simple planar graphs.

Theorem 2 [6]. *For any d-convex simple planar graph $G = (X,\, U)$, $|X| > 3$, there exists a tree T such that $G = L(T,\, T_0)$.*

The theorem 2 determines a structure for d-convex simple planar graphs, that we are going to use for finding the dimension of these graphs.

All d-convex simple graphs with number of vertexes $n < 5$ are in fig. 1.

It is easy to see that $dim\, K_1 = 0$, $dim\, K_2 = dim\, P_3 = 1$, $dim\, K_3 = dim\, C_4 = 2$ (see fig. 1.). Let now $G = (X;\, U)$ be a d-convex simple planar graph, with $|X| \geq 5$. In this case G has at least one vertex x with degree at least 3, which has also a copy $\bar{x}$. Let us suppose that $\Gamma(x) = \Gamma(\bar{x}) = \{y_1,\, y_2,\, y_3, \ldots, y_p\}$, $p \geq 3$. If we suppose that $dim\, G = 2$ then elements of set $\Gamma(x)$ are placed on circle of radius 1 with centre in one point that corresponds to vertex x. On the other

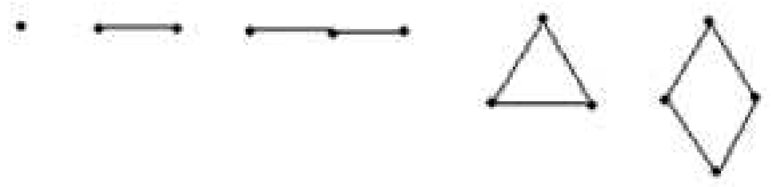

Figure 1. K_1, K_2, P_3, K_3 and C_4 respectively.

hand, the same vertexes should be on circle of radius 1 with centre in one point, that corresponds to vertex $\bar{x}$. This means that in $\mathbb{R}^2$ we can draw two different circles of radius 1, which intersect each other in $p,\ p \geq 3$ points, that is impossible. So we have $dim\, G \geq 3$. We will show now that $dim\, G = 3$.

Theorem 3 *If $G = (X, U),\ |X| \geq 5$ is a d-convex simple planar graph then $dim\, G = 3$.*

Proof: Let $G = (X,\ U),\ |X| \geq 5$ be a d-convex simple planar graph. According to the theorem 2 there exists a tree T such that $G = L(T;\ T_0)$. We will prove theorem using mathematical induction by number of vertexes $n = |X| \geq 5$. The unique d-convex simple planar graph with $n = 5$ is graph in fig. 2. Let us choose in $\mathbb{R}^3$ two points A, B at distance less than 2 and draw two spheres of radius 1, with centers in A and B. The intersection of these spheres is a circle $\mathcal{C}$. Let us fix on circle $\mathcal{C}$ three points $D_1,\ D_2,\ D_3$. If we place vertexes x and $\bar{x}$ of graph from fig. 2 in points A and B, and $y,\ y',\ y''$ in points $D_1,\ D_2,\ D_3$ and join them by segments of length one to x and $\bar{x}$, then we obtain an inclusion of this graph in $\mathbb{R}^3$.

Let us suppose that assertion of theorem is true for any d-convex simple planar graph G with $n \leq k,\ k \geq 5$ vertexes.

We will examine now case $n = k + 1$. According to the theorem 2, for graph G there exists a tree T such that $G = L(T;\ T_0)$. For any

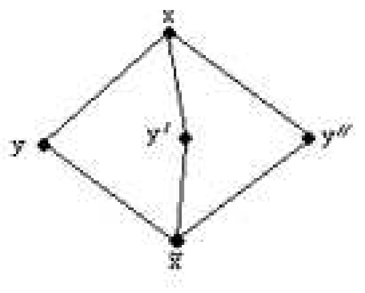

Figure 2. d-Convex simple planar graphs with 5 vetrtexes.

arbitrary vertex x from T we will denote by $deg_T\ x$ and $deg_G\ x$ the degree of x in tree T and graph G respectively.

Let x, y be two adjacent vertexes in tree T, such that $deg_T\ x = 1$. It is obvious that $deg_G\ x = 2$. Since according to the conditions of this theorem the graph G has at least 5 vertexes, we have the result that $deg_T\ y \geq 2$. We will analyze two cases.

I $deg_T\ y \geq 3$. If we eliminate the suspended vertex x from T, then we obtain a tree T', where y is not a suspended vertex. This implies that tree T'_0 coincides with T_0. According to the theorem 1 the graph $G' = L(T', T'_0)$ is d-convex simple and planar. Because G' has exact one vertex less than G, by mathematical induction we have $dim G' = 3$. Let $\bar{y}$ be copy of y and its image in T_0. In G' these two points are also copies one for another and theirs neighborhoods coincide. Let $\Gamma(y) = \{\omega_1,\ \omega_2, \ldots,\ \omega_p\} = \Gamma(\bar{y}),\ p \geq 3$. When we involve the graph G' in $\mathbb{R}^3$, then vertexes from $\Gamma(y)$ are placed on the circle that is at intersection of two spheres of radius 1, with centers in y and $\bar{y}$. If we place on this circle the vertex x, then distances from it to y and $\bar{y}$, as centers of spheres, are equal to 1. As the result, the graph G can be involved in $\mathbb{R}^3$, so $dim\ G = 3$.

II $deg_T\ y = 2$. So in T there exists a vertex $z \neq x$ and adjacent to y. Because G has at least 5 vertexes, then $deg_T\ z \geq 2$. This means that tree T_0 contains vertexes $\bar{y}$ and $\bar{z}$, which are images of vertexes y and z. So, in graph G there are true relations:

$$\Gamma(y) = \Gamma(\bar{y}),\ |\Gamma(y) \geq 3|$$

$$\Gamma(z) = \Gamma(\bar{z}),\ |\Gamma(z) \geq 3|$$

When we eliminate the vertex x from T, we obtain a new tree T', where y is a suspended vertex ($deg_{T'}\ y = 1$). In this case the tree T_0', which is obtained from T' by elimination of all suspended vertexes, differs from T_0 by one vertex $\bar{y}$. So $T' = T - x$, $T_0' = T_0 - \bar{y}$. We denote $G = L(T', T_0')$. By mathematical induction $dim\, G = 3$. Since $|\Gamma(z) \geq 3|$, then when we involve the graph G' in $\mathbb{R}^3$, the set of vertexes $\Gamma(z)$ is placed on the circle, that is at intersection of two spheres of radius 1, with centers in z and $\bar{z}$. On this circle we can place the vertex $\bar{y}$ (the vertex y is already on this circle as a vertex of graph G'). Now since y and $\bar{y}$ are placed on circle of diameter less than 2, then intersection of two unitary spheres with centers in y and $\bar{y}$ is a new circle, where we can place the vertex x. As the result, the graph G can be involved in $\mathbb{R}^3$, so $dim\, G = 3$.

By mathematical induction the assertion is true for any natural n. Theorem is proved. $\square$

Now we are going to study a class of graphs with a wider family of d-convex sets.

3 Dimension of d-Conves Quasi-Simple Planar Graphs

Definition 3 [6]. *A graph $G = (X, U)$ is called d-convex quasi-simple graph if any d-convex set $A \subset X$ forms in G a complete sub-graph.*

It is obvious that each d-convex simple graph is d-convex quasi-simple. D-convex quasi-simple graphs with $|X| < 5$ are graphs in fig. 1 and fig. 3.

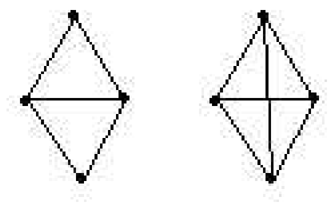

Figure 3. Graphs P and K_4.

Dimension of graphs in fig. 3 is 2 and 3 respectively.

Definition 4 *An edge $u = (x, y)$ of a graph G is called diagonal edge if in G there exist at least 2 vertexes s, t adjacent with x, y.*

Theorem 4 [6]. *If $G = (X, U)$ is d-convex quasi-simple planar graph then there exists a d-convex simple planar graph $G_b = (X, U_b)$ (called basis graph) such that G is obtained from G_b through adding some diagonal edges.*

In order to describe the structure of d-convex quasi-simple planar graphs we will describe next class of graphs. Let T be a tree with at least 3 vertexes. We will denote by $\mathcal{R}(T)$ the set of graphs that is obtained from T through adding some new edges, which will respect next conditions.

1. each new edge is incident to two old vertexes in T, the distance between which is equal to 2;
2. each new edge is incident to at least one suspended vertex;
3. in new graph the degree of each suspended vertex in T will be equal to at most 3;
4. if the new edge (x, y) join one suspended vertex with one unsuspended vertex from T and z is that vertex of T, that was between x and y, then $deg_T\ z = 2$;

5. a) if T is not a star, then in new graph there do not exist simple cycles, that are formed from suspended vertex of T;
   b) if T is a star and in new graph there exists a cycle that consists only from suspended vertexes, then this cycle passes through all suspended vertexes of T;

Like above we will denote by T_0 a sub-graph of T that consists of all vertexes and edges of T without those suspended. Let R be a graph from $\mathcal{R}(T)$. Let us construct graph $L(R, T_0)$ like above.

Theorem 5 [7]. *The graph $G = (X, U)$, $|X| > 4$, is d-convex quasi-simple planar graph if and only if there exists a tree T and a graph $R \in \mathcal{R}(T)$ such that $G = L(R, T_0)$.*

It is easy to see that for the d-convex quasi-simple planar graph $L(R, T_0)$ one basis graph is $L(T, T_0)$. The question is, on how many the dimension of d-convex quasi-simple planar graphs changes comparatively with the dimension of d-convex simple planar graphs. We are going to answer it.

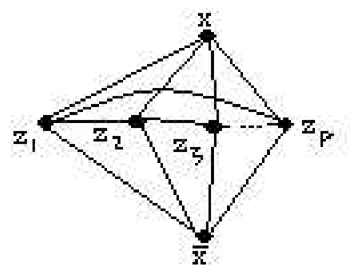

Figure 4. Class Q.

Condition 5. b) gives a special class of d-convex quasi-simple planar graphs. Any graph from this class is formed from two vertexes copies x and $\bar{x}$ with $\Gamma(x) = \Gamma(\bar{x}) = \{ y_1, y_2, \dots, y_p \}$, $p \geq 3$, such that $z_1 \sim z_2 \sim z_3 \sim \dots \sim z_p \sim z_1$(see fig. 4.). Let us denote this class by Q. First, we are going to examine the class of graphs Q. These graphs differ by number of suspended vertexes in the star T. Let us denote by Q_n, $n > 2$ a graph of this class with n suspended vertexes in T. The problem is to inscribe in a circle of radius less than 1 a regular, closed polygonal line with unit length of edges. If we would be able to do this then the graph Q_n will be placed in space like in the fig. 5., where there in the circle will be the closed polygonal line we are talking about.

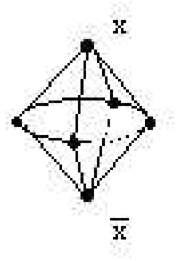

Figure 5. Q_n in 3-space.

For $n = 3,\ 4$ and 5 the circles are just those we inscribe a equilateral triangle, a square and a regular pentagon, because radiuses of these circles are less than 1. The circle where a unit regular hexagon can be inscribed has radius exactly equal to 1. That is why the graph Q_6 remains as a limit case. In order to study general case we need to prove first 3 lemmas.

Lemma 1 *The natural odd numbers $n = 2k + 1$ are relatively prime with k.*

Proof: Let us assume that the greater common divisor $(2k+1,\ k) = m$, then $\exists\ p,\ q \in \mathbb{N}$ such that:

$$2k + 1 = mp; \quad k = mq$$

or 2mq+1=mp, equivalent with m(p-2q)=1. The product of 2 natural numbers is 1 if and only if each of them is equal to 1, so $m = 1$. □

Lemma 2 *The natural even numbers of kind $n = 4k$ are relatively prime with $2k - 1$.*

Proof: Let us assume that the greater common divisor $(4k,\ 2k-1) = m$, then $\exists\ p,\ q \in \mathbb{N}$ such that:

$$4k = mp;\ \ 2k-1 = mq$$

or $2(mq+1) = mp$, we have $m(p-2q) = 2$. The product of 2 natural numbers is 2, so either $m = 1$, or $m = 2$. The number m could not be equal to 2 because it is divisor for an odd number $2k-1$, the result is $m = 1$. □

Lemma 3 *The natural even numbers of kind* $n = 4k+2$ *are relatively prime with* $2k-1$.

Proof: Let us assume that the greater common divisor $(4k+2,\ 2k-1) = m$, then $\exists\ p,\ q \in \mathbb{N}$ such that:

$$4k+2 = mp;\ \ 2k-1 = mq$$

or $2(mq+1)+2 = mp$, we have $m(p-2q) = 4$. The product of 2 natural numbers is 4, so either $m = 1$, or $m = 2$, or $m = 4$. The number m can not be equal to 2 because it is divisor for an odd number $2k-1$, m also can not be equal to 4, as divisor of the number $4k+2$, the result is $m = 1$. □

From algebra we know that every number g, that is relatively prime with n $(g < n)$, is a generator for the group $\mathbb{Z}_n$, i. e. we have $\mathbb{Z}_n =< \bar{g} >= \{\ \bar{g},\ \overline{g+g},\ \overline{g+g+g}, \ldots,\ \underbrace{\overline{g+g+\ldots+g}}_{n-times}\ \} = \{\bar{0},\ \bar{1},\ \bar{2}, \ldots, \overline{n-1}\ \}$. Now let us take a circle and place there all natural numbers from 0 till $n-1$, at equal distances one from another (see fig. 6.).

Let g be that number respectively prime with n from lemmas. If we will draw line from 0 to g, then from g to $g + g(mod n)$, and so on, then we will draw a polygonal line, that will be closed, because g is a cyclic generator of all these numbers and regular, because every line is based on the portion of circle of the same length. If we ask for the length between two numbers be equal to 1, then we obtain that

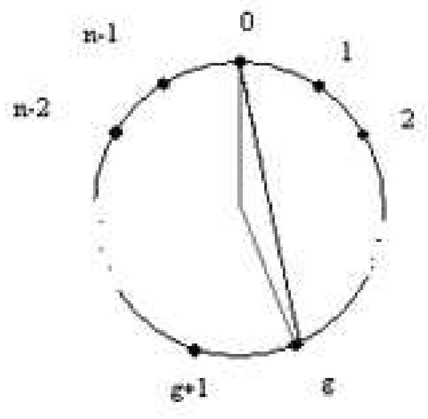

Figure 6. Closed polygonal regular line inscribed in circle.

radius of the circle in fig. 6. is $r = \frac{1}{2\sin(\alpha)}$, $\alpha = \frac{\pi g}{n}$. For $n = 2k+1$, $\alpha = \frac{\pi k}{2k+1} = \frac{\pi}{2}\frac{2k}{2k+1}$. For $n = 4k$, $\alpha = \frac{\pi(2k-1)}{4k} = \frac{\pi}{2}\frac{2k-1}{2k}$. For $n = 4k+2$, $\alpha = \frac{\pi(2k-1)}{4k+2} = \frac{\pi}{2}\frac{2k-1}{2k+1}$. Since we are interested in $n > 6$, then we have $\frac{\pi}{6} < \alpha < \frac{\pi}{2}$, or $\frac{1}{2} < \sin(\alpha) < 1$, finally $\frac{1}{2} < r < 1$. The last fact implies that we can place vertexes $x,\ \bar{x}$ at distance $d = 2\sqrt{1-r^2}$, $(0 < d < \sqrt{3})$ and in this case the circle, that is at intersection of unit spheres with centres in x and $\bar{x}$, will have radius r.

We have already proved next theorem:

Theorem 6 *If Q_n, $n > 2$, $n \neq 6$ is a d-convex simple planar graph from Q, then dim $Q_n = 3$.*

Theorem 7 *If $G = (X,\ U)$, $|X| > 4$, $G \neq Q_6$, is a d-convex quasi-simple planar graph, then dim $G = 3$.*

Proof: Let G be a graph that satisfies conditions of theorem. If $G \in Q$ then according to the last theorem $dim G = 3$. If $G \notin Q$ then according to the theorem 5 there are a tree T, a graph $R \in \mathcal{R}(T)$ and T_0 such that $G = L(R,\ T0)$. According to the theorem 1, the graph $G_b = L(T;\ T_0)$

is d-convex simple and planar, so according to the theorem 3, it has dimension equal to 3. Let us place in space the graph G_b like in the theorem 3.

There can be two cases.

1. All new edges in R join only suspended vertexes of T. Then according to condition 5. a), we have either some separated edges, that join some suspended vertexes at distance 2 and according to condition 1 they have the same predecessor, or some chains, that join also suspended vertexes of the same predecessor, according to conditions 1 and 3. So, if we have that y is a predecessor in T for suspended vertexes, that are adjacent in R, then we can replace vertexes y and $\bar{y}$ at distance d, $(d < \sqrt{(3)})$ for example $d \leq 0,02$ and try to inscribe our chains in the circle formed between unit spheres with centres in y, $\bar{y}$. If we can't do this, because some vertexes pretend at the same place, then try again to modify distance between y and $\bar{y}$. We are sure that we can find the distance for our copies, for which we can place the chains. We can place these copies at distance at least $d = \sqrt{(1-r^2)}$, where like above, $r = \frac{1}{2\sin(\frac{\pi g}{n})}$, for $n = |\Gamma(y)|$ and g-computed from lemmas. We will draw the chains here in the same order like we do it above for cycles. So, we have that this graph is 3-dimensional.

2. Through new edges there exist some, that join a suspended vertex with one unsuspended vertex, for example $(x,\, y)$ is a new edge with this property (see fig. 7. a)). Then in $L(R,\, T_0)$ this part of graph will be like in fig. 7. b).

   Now we will consider a new tree T^1, that will be identical with T, except the portion from fig. 7. a), where it will be like in the fig. 8. a), i. e. the vertexes y, z and $\bar{z}$ are suspended in T^1 and edges $(y,\, z)$, $(y,\, \bar{z})$ will be new edges, which will be added to R^1. Then the graph $L(R^1,\, T_0^1)$ looks like the graph $L(R,\, T_0)$, except the part of graph, where it looks like in the fig. 8. b). It is easy to see that graph $L(R,\, T_0)$ is a sub-graph of $L(R^1, T_0^1)$,

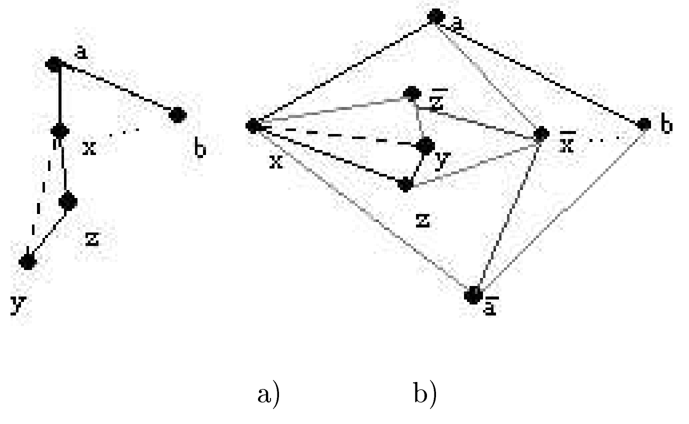

Figure 7. Portion of graphs R, G, where an edge joins a suspended vertex with the unsuspended one.

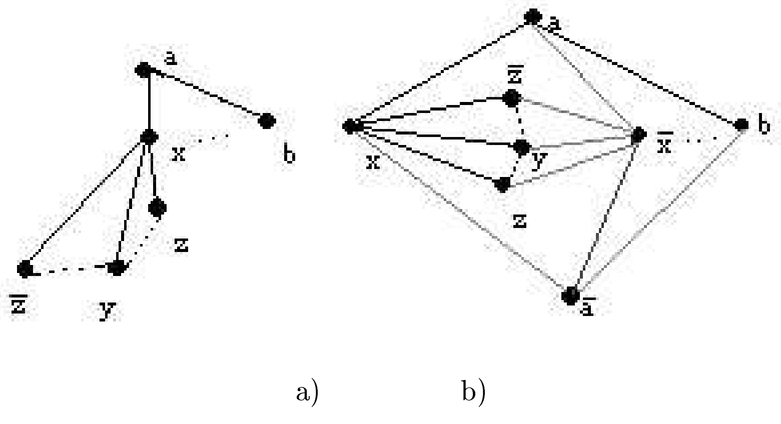

Figure 8. Portion of graphs R^1, G^1, after transformations.

because of edge $(\bar{x}, y)$. If we will do the same transformations for each edge that joins unsuspended vertexes, then we will obtain a graph, that is d-convex quasi-simple planar graph and all new edges join only suspended vertexes. According to the first part of the proof, this graph can be involved in 3-dimensional space, after that we can exclude the additional edges. So this graph is also 3- dimensional.

Theorem is proved. $\square$

4 Algorithm for Involving d-Convex Simple Planar Graphs in $\mathbb{R}^3$

Let $G = (X; U)$ be a d-convex simple planar graph with $|X| \geq 5$. According to the theorem 2, we can consider $G = L(T; T_0)$. Algorithm for involving G in $\mathbb{R}^3$ is an iterative algorithm, at each step of which the place of all vertexes from neighborhood of two copies-vertexes is found. In process of algorithm two special sets of vertexes are formed:

1. R - the set of all vertexes of the graph G, which we have not yet placed in $\mathbb{R}^3$. Of course, initially $R = X$.
2. S - the set of all pairs of copies-vertexes, which was already placed in $\mathbb{R}^3$, but their neighborhoods were not yet researched.

Description of algorithm:

1. The sets $R = X,\ S = \emptyset$ are formed;
2. Find any vertex $x \in R$, of degree greater than 2 and its copy $\bar{x}$. Include them in S, i. e. $S = S \cup \{(x,\ \bar{x})\}$. Place the vertexes $x,\ \bar{x}$ in $\mathbb{R}^3$, arbitrary at distance less than 2.
3. Take any pair of copies $(x;\ \bar{x}) \in S$ and change the sets:

$$S = S - \{(x;\ \bar{x})\}; \quad R = R - \{x;\ \bar{x}\}.$$

4. For every $y \in \Gamma(x) \cap R$:

   (a) if $degy = 2$, then place arbitrary the vertex y on circle that is at intersection of spheres with radius 1 and centres in x, $\bar{x}$. Modify the set R:
   $$R = R - \{y\};$$

   (b) if $deg\ y > 2$, then find $\bar{y} \in \Gamma(x) \cap R$, place arbitrary the vertex y, $\bar{y}$ on circle that is at intersection of spheres with radius 1 and centres in x, $\bar{x}$. Modify the sets S and R:
   $$S = S \cup \{(y;\ \bar{y})\}; \quad R = R - \{y;\ \bar{y}\};$$

5. If $S \neq \emptyset$ then go to 3, else STOP.

5 Algorithm for Involving d-Convex Quasi-simple Planar Graphs in $\mathbb{R}^3$

Let $G = (X; U)$ be a d-convex quasi-simple planar graph with $|X| \geq 5$, $G \neq Q_6$. According to the theorem 4 we have that any d-convex quasi-simple planar graph is obtained from the d-convex simple one by adding some new edges. These edges form either a family $\mathcal{L}$ of simple chains, or a simple cycle and then the graph is one from family Q. Algorithm for involving G in $\mathbb{R}^3$ is also an iterative algorithm, and in order to discribe it we will use the same sets of vertexes S and R.

Description of algorithm:

1. If $G \in Q$ then the pair of copies-vertexes is placed in $\mathbb{R}^3$ at distance
   $$d = 2\sqrt{1 - r^2}, \quad where\ \ r = \frac{1}{2\sin\frac{\pi g}{n-2}}$$
   and
   $$g = \begin{cases} k, & if\ n-2 = 2k+1 \\ 2k-1, & if\ n-2 = 4k, \quad or\ n-2 = 4k+2 \end{cases}$$
   The other vertexes of G are placed in $\mathbb{R}^3$ like it is described in proof of theorem 6. If $G \notin Q$ then go to 2.

2. The sets $R = X$, $S = \emptyset$ are formed and find a pair of copies-vertexes x, $\bar{x}$ from G, set d=0,01; Change the set:

$$R = R - \{x;\ \bar{x}\}.$$

3. Put d=d+0,01. If $d \geq \sqrt{3}$ then put $d = \sqrt{1-r^2}$, where r is computed like in 1. for $n = |\Gamma(x)|$. Place the vertexes x, $\bar{x}$ in $\mathbb{R}^3$ at distance d.

4. From vertexes $\Gamma(x) \cap R$ form two new sets W_1, W_2:

   $$W_1 = \{y \in \Gamma(x) \cap R \mid \Gamma(x) \cap \Gamma(y) = \emptyset\}; \quad W_2 = (\Gamma(x) \cap R) - W_1;$$

   If $W_2 \neq \emptyset$ then go to 5, else go to 6.

5. The vertexes from W_2 generate in G a family of simple chains. Try to place all vertexes from W_2 on the circle between x, $\bar{x}$, such that adjacent vertexes to be placed at distance 1. If we can do this then $R = R - W_2$ and go to 6, else go to 3.

6. If $W_1 = \emptyset$ then go to 7, else for every $y \in W_1$

   (a) if $deg y = 2$, then place arbitrary the vertex y on circle that is at intersection of spheres with radius 1 and centres in x, $\bar{x}$. Modify the set R:
   $$R = R - \{y\};$$

   (b) if $deg\ y > 2$, then find $\bar{y} \in \Gamma(x) \cap R$, place arbitrary the vertex y, $\bar{y}$ on circle that is at intersection of spheres with radius 1 and centres in x, $\bar{x}$. Modify the sets S and R:

   $$S = S \cup \{(y;\ \bar{y})\}; \quad R = R - \{y;\ \bar{y}\};$$

7. If $S \neq \emptyset$ then take any pair $(x;\ \bar{x})$, put $S = S - \{(x;\ \bar{x})\}$, $R = R - \{x,\ \bar{x}\}$, d=0,01 and go to 3, else STOP.

References

[1] E.D. Stotskii, *About Embanding Finite Metric in Graphs, Sibirian Journal of Mathematics*, nr. 5(1964) pp. 1203–1206. (in Russian)

[2] C.A. Zaretskii, *Construction of Tree by Set of Distances among Suspended Vertexes,* UUN 20, edition 6(1965), pp. 90–92. (in Russian)

[3] M.E. Talkin, *About Realizability of Matrices of Distances in Unit Cubes,* Problems of Cybernetics, edition 7(1962), pp. 31–42. (in Russian)

[4] V. Bolteanskii, P. Soltan, Combinatorial Geometry of Different Classes of Convex Sets, Kishinev, 1985, 279 p.(in Russian)

[5] S. Cataranciuc, *The d-Convex Simple Planar Graphs,* Research of Numerical Methods and Theoretical Cybernetics, Kishinev, 1985, pp. 68–75. (in Russian)

[6] S. Cataranciuc, *Structure and Isomorphism of d-Convex Simple Planar Graphs,* Research of General Algebra, Geometry and its Application, Kishinev, 1986, pp. 92–96. (in Russian)

N. Sur, S. Cataranciuc,

Received October 3, 2005

Faculty of Mathematics and Informatics,
Moldova State University,
A. Mateevici street, 60, MD-2009
Republic of Moldova
E–mails: *nadejda_sur@rambler.ru*,
caseg@usm.md

Computer Science Journal of Moldova, vol.**13**, no.2(38), 2005

Game-Theoretic Approach for Solving Multiobjective Flow Problems on Networks*

Maria A. Fonoberova, Dmitrii D. Lozovanu

Abstract

The game-theoretic formulation of the multiobjective multicommodity flow problem is considered. The dynamic version of this problem is studied and an algorithm for its solving, based on the concept of multiobjective games, is proposed.

Mathematics Subject Classification 2000: 90B10, 90C35, 90C27, 90C47.

Keywords and phrases: dynamic networks, multiobjective flows, dynamic flows, multiobjective games, optimal strategies.

1 Introduction

In this paper we consider the game-theoretic formulation of the multiobjective multicommodity flow problem. This problem consists of shipping a given set of commodities from their respective sources to their sinks through a network in order to optimize different criteria so that the total flow going through each edge does not exceed its capacity. The network is a collection of locations with directed edges identifying feasible transportation operations. The planning problem is to determine the amount to transport on each link in order to move all the cargo respecting fixed criteria.

If we associate to each commodity a player, we can regard this problem as a game problem, where players interact between them and the choices of one player influence the choices of the others. Each player has a vector utility function, components of which are such factors as

*Supported by CRDF-MRDA 008 award MOM2-3049-CS-03

transportation cost, speed of transit time, quality of service and others. Each player seeks to optimize his own utility function in response to the actions of the other players and all the players perform this optimization simultaneously and at the same time players are interested to preserve Nash optimality principle when they interact between them. The game theory fits perfectly in the realm of such a problem, and an equilibrium or stable operating point of the system has to be found. We study the dynamic version of the multiobjective multicommodity flow problem and use the concept of multiobjective games from [1, 2].

2 The game-theoretic approach and some preliminary results

In order to study our multiobjective multicommodity flow problem we will use the game-theoretic concept from [1, 2].

The multiobjective game with p players is denoted by $\overline{G} = (X_1, X_2, \dots, X_p, \overline{F}_1, \overline{F}_2, \dots, \overline{F}_p)$, where X_i is a set of strategies of player i, $i = \overline{1,p}$, and $\overline{F}_i = (F_i^1, F_i^2, \dots, F_i^r)$ is a vector payoff function of player i, defined on set of situations $X = X_1 \times X_2 \times \cdots \times X_p$:

$$\overline{F}_i \ : \ X_1 \times X_2 \times \cdots \times X_p \to R^r, \ i = \overline{1,p}.$$

Each component F_i^k of $\overline{F}_i$ corresponds to a partial criterion of player i and represents a real function defined on set of situations $X = X_1 \times X_2 \times \cdots \times X_p$:

$$F_i^k \ : \ X_1 \times X_2 \times \cdots \times X_p \to R^1, \ k = \overline{1,r}, \ i = \overline{1,p}.$$

We call the solution of the multiobjective game $\overline{G} = (X_1, X_2, \dots, X_p, \overline{F}_1, \overline{F}_2, \dots, \overline{F}_p)$ the Pareto-Nash equilibrium and define it in the following way.

Definition. The situation $x^* = (x_1^*, x_2^*, \dots, x_p^*) \in X$ is called Pareto-Nash equilibrium for the multiobjective game $\overline{G} = (X_1, X_2,$

$\ldots, X_p, \overline{F}_1, \overline{F}_2, \ldots, \overline{F}_p)$ if for every $i \in \{1, 2, \ldots, p\}$ the strategy x_i^* represents Pareto solution for the following multicriterion problem:

$$\max_{x_i \in X_i} \to \overline{f}^{\,i}_{x^*}(x_i) = (f^{i1}_{x^*}(x_i), f^{i2}_{x^*}(x_i), \ldots, f^{ir}_{x^*}(x_i)),\ i = \overline{1,p},$$

where

$$f^{ik}_{x^*}(x_i) = F_i^k(x_1^*, x_2^*, \ldots, x_{i-1}^*, x_i, x_{i+1}^*, \ldots, x_p^*),\ k = \overline{1,r}.$$

This definition generalizes well-known Nash equilibria notion for classical noncooperative games (single objective games) and Pareto optimum for multicriterion problems. If $r = 1$, then $\overline{G}$ becomes classical noncooperative game, where x^* represents Nash equilibria solution; in the case $p = 1$ the game $\overline{G}$ becomes Pareto multicriterion problem, where x^* is Pareto solution.

Further we formulate the main theorem which represents an extension of the Nash theorem for our multiobjective version of the game.

Theorem 1. Let $\overline{G} = (X_1, X_2, \ldots, X_p, \overline{F}_1, \overline{F}_2, \ldots, \overline{F}_p)$ be a multiobjective game, where $X_1, X_2, \ldots, X_p$ are convex compact sets and $\overline{F}_1, \overline{F}_2, \ldots, \overline{F}_p$ represent continuous vector payoff functions. Moreover, let us assume that for every $i \in \{1, 2, \ldots, p\}$ each component $F_i^k(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_p)$, $k \in \{1, 2, \ldots, r\}$, of the vector function $\overline{F}_i(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_p)$ represents a concave function with respect to x_i on X_i for fixed $x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p$. Then for multiobjective game $\overline{G} = (X_1, X_2, \ldots, X_p, \overline{F}_1, \overline{F}_2, \ldots, \overline{F}_p)$ there exists Pareto-Nash equilibria situation $x^* = (x_1^*, x_2^*, \ldots, x_p^*) \in X_1 \times X_2 \times \cdots \times X_p$.

The proof of Theorem 1 is given in [2].

So, if conditions of Theorem 1 are satisfied then Pareto-Nash equilibria solution for multiobjective game can be found by using the following algorithm.

Algorithm

1. Fix an arbitrary set of real numbers $\alpha_{11}, \alpha_{12}, \ldots, \alpha_{1r}, \alpha_{21}, \alpha_{22}, \ldots, \alpha_{2r}, \ldots, \alpha_{p1}, \alpha_{p2}, \ldots, \alpha_{pr}$, which satisfy condition

$$\begin{cases} \sum\limits_{k=1}^{r} \alpha_{ik} = 1, \quad i = \overline{1,p}; \\ \alpha_{ik} > 0, \qquad k = \overline{1,r},\ i = \overline{1,p}; \end{cases}$$

2. Form the single objective game $G = (X_1, X_2, \ldots, X_p, f_1, f_2, \ldots, f_p)$, where

$$f_i(x_1, x_2, \ldots, x_p) = \sum_{k=1}^{r} \alpha_{ik} F_i^k(x_1, x_2, \ldots, x_p),\ i = \overline{1,p};$$

3. Find Nash equilibria $x^* = (x_1^*, x_2^*, \ldots, x_p^*)$ for noncooperative game $G = (X_1, X_2, \ldots, X_p, f_1, f_2, \ldots, f_p)$ and fix x^* as Pareto-Nash equilibria solution for multiobjective game $\overline{G} = (X_1, X_2, \ldots, X_p, \overline{F}_1, \overline{F}_2, \ldots, \overline{F}_p)$.

3 The multiobjective multicommodity flow problem

3.1 The static model

We consider a network $N = (V, E, K, c, d, \varphi)$ that contains a directed graph $G = (V, E)$, where V is a set of vertexes, E is a set of edges, and $K = \{1, 2, \ldots, p\}$ is a set of commodities that must be routed through the same network. Each edge $e \in E$ has a nonnegative capacity c_i^e which bounds the amount of flow of commodity i allowed on arc e. There is a throughput demand d_i^v defined on vertexes for each commodity in the network. To model transit costs we define the cost function φ: $E \times R_+ \to R_+$. In such a way the following restrictions have to be verified for the flow x_i^e of commodity i sent on edge e:

$$\sum_{e \in E^+(v)} x_i^e - \sum_{e \in E^-(v)} x_i^e = d_i^v,\ \forall\, v \in V,\ \ \forall\, i \in K;$$

$$0 \le x_i^e \le c_i^e, \ \ \forall\, e \in E, \ \ \forall\, i \in K;$$

where $E^+(v) = \{(u,v)\,|\,(u,v) \in E\}$, $\ E^-(v) = \{(v,u)\,|\,(v,u) \in E\}$.

The flow has to be shipped through the network in such a way to optimize the vector utility function $\overline{F}_i = (F_i^1, F_i^2, \ldots, F_i^r)$ for every $i \in K$:

$$\overline{F}_i \ : \ X_1 \times X_2 \times \cdots \times X_p \to R^r,$$

$$F_i^k \ : \ X_1 \times X_2 \times \cdots \times X_p \to R^1, \ k = \overline{1,r},$$

where X_i is a set of flows of commodity i, r is a number of criteria.

3.2 The dynamic model

The static flow can not properly consider the evolution of the system under study over time. The time is an essential component, either because the flows take time to pass from one location to another, or because the structure of network changes over time. To tackle this problem, we use dynamic network flow models instead of the static ones.

We consider the discrete time model, in which all times are integral and bounded by time horizon T, which defines the makespan $\mathbb{T} = \{0, 1, \ldots, T\}$ of time moments we consider. Time is measured in discrete steps, so that if one unit of flow leaves node u at time t on arc $e = (u,v)$, one unit of flow arrives at node v at time $t + \tau^e$, where τ^e is the transit time of arc e. Each commodity has its own time-interval $\mathbb{T}_i \subset \mathbb{T}$.

A dynamic network $N = (V, E, K, c, \tau, d, \varphi)$ consists of a directed graph $G = (V, E)$, a set $K = \{1, 2, \ldots, p\}$ of commodities that must be routed through the same network within the makespan $\mathbb{T}$, capacity function c: $E \times K \times \mathbb{T} \to R_+$, transit time function τ: $E \to R_+$, demand function d: $V \times K \times \mathbb{T} \to R$ and cost function φ: $E \times R_+ \times \mathbb{T} \to R_+$. In such a way, the following restrictions have to be verified for the flow $x_i^e(t)$ of commodity i sent on link e at time $t \in \mathbb{T}$:

$$\sum_{\substack{e \in E^+(v) \\ t - \tau_e \ge 0}} x_i^e(t - \tau_e) - \sum_{e \in E^-(v)} x_i^e(t) = d_i^v(t), \ \forall\, t \in \mathbb{T}_i, \ \forall\, v \in V, \ \forall\, i \in K;$$

$$0 \le x_i^e(t) \le c_i^e(t), \quad \forall\, t \in \mathbb{T}_i, \ \forall\, e \in E, \ \forall\, i \in K;$$

$$x_i^e(t) = 0, \ \forall\, e \in E, \ t = \overline{T - \tau_e + 1, T}, \ \forall\, i \in K.$$

The flow has to be shipped through the network in such a way to optimize the vector utility function $\overline{F}_i = (F_i^1, F_i^2, \dots, F_i^r)$ for every $i \in K$, where

$$\overline{F}_i \ : \ (X_1 \times \mathbb{T}_i) \times (X_2 \times \mathbb{T}_i) \times \cdots \times (X_p \times \mathbb{T}_i) \to R^r,$$

$$F_i^k \ : \ (X_1 \times \mathbb{T}_i) \times (X_2 \times \mathbb{T}_i) \times \cdots \times (X_p \times \mathbb{T}_i) \to R^1, \ k = \overline{1, r}.$$

Using the apparatus from Section 2 we reduce the considered problem to single-objective multicommodity flow problem.

4 The game formulation of the multiobjective multicommodity flow problem

In the framework of the game theory each commodity in the formulated problem is associated with a player. We consider a general model with p agents each of which wishes to optimize its own vector utility function $\overline{F}_i$, $i = \overline{1, p}$, which is defined on the set of strategies of all players. Each component F_i^k, $k = \overline{1, r}$, of the vector utility function F_i of player i corresponds to a partial criterion of player i. Control decisions are made by each player according to its own individual performance objectives and depending on the choices of the other players.

Each player competes in a Nash equilibrium manner so as to optimize his own criteria in the task of transporting of flow from its origins to its destinations. Let x_i be strategy of user i and x_{-i} be strategies of all other agents. For fixed i we say that $x^* = (x_1^*, \dots, x_p^*)$ is a Nash equilibrium if no user can improve his utility by unilateral deviation. In our problem each player has several objectives, so we use the Pareto-Nash equilibrium concept extended to networks.

In such a way, players intend to optimize their utility functions in the sense of Pareto and at the same time players are interested to preserve Nash optimality principle when they interact between them.

The cost of transportation of a given resource, the time necessary to transport it to its destination as well as the quality of the transportation play the role of the components of the vector utility function of a player in the game-theoretic formulation of the problem. If payoff functions satisfy conditions of the above theorem then for solving such a problem we apply the algorithm proposed above.

5 Applications

In real-life problems users have to make decision concerning routing as well as type and amount of resources that they wish to transport. Different sets of parameters may suit the service requirements of a user. However, the performance measures depend not only on the user's choices, but also on the decisions of other connected users, where this dependence is often described as a function of some network "state". In this setting the game paradigm and the Pareto-Nash equilibrium concept become the natural choice at the user level.

Game theoretic models are widely employed in the context of flow control, routing, virtual path bandwidth allocation and pricing in modem networking. Flow problems in multimedia applications (teleconferencing, digital libraries) over high-speed broadband networks can serve a good example of this. In a multimedia network telecommunication companies carrying different traffic types (voice, data, and video) may share the limited common network resources such as buffers or transmission lines. These companies may have different objectives of maximizing packet throughput or minimizing packet blocking probability. A Pareto-Nash equilibrium may be reached when companies achieve their objectives in such a way that no company can improve its own performance by unilaterally changing its traffic load admission and routing strategies.

The problem of providing bandwidth which will be shared by many users ([3, 4]) is one of the most important problems. As it is typical for games in such a problem the interaction among the users on their individual strategies has to be imposed. This can be done using a utility function that depends on the availability of bandwidth and other factors in the network. The problem of consumer i consists in choosing which network resource to use and how much to use it.

The game theoretic approach can be applied in a problem of power control in radio systems ([5]). In cellular radio systems power is a valuable commodity for the users, so a mobile user prefers to use less power and at the same time to obtain better quality-of-service from assigned base stations. In such a way, each mobile user wishes to optimize his own personal objectives and choices of mobile user depend on choices of other users.

At the end we want to mention that the network performance is a very important factor for the improvement of the system work, which can be achieved both during the phase, when the network parameters are sized ([6]), and during the phase of the operation of the network. In such a way, the network performance is not completely determined by the technical characteristics of the network but also is a function of system state, therefore the issue of optimal user strategies is a very actual problem.

References

[1] E. Altman, T. Boulogne, R. El-Azouzi, T. Jimenez, L. Wynter. *A survey on networking games in telecommunications. Computers and Operations Research.* (to appear).

[2] D. Lozovanu, S. Pickl. *Pareto-Nash equilibrium for multiobjective games. European Journal of Operational Research.* (submitted for publication).

[3] Aurel A. Lazar, Ariel Orda, and Dimitrios E. *Virtual Path Bandwidth Allocation in Multiuser Networks.* IEEE/ACM transaction on networking, 5(6), (1997), pp. 861–871.

[4] Jeffrey K. MacKie-Mason and Hal R. Varian. *Pricing Congestible Network Resources.* IEEE Journal of selected areas in communications, 13(7), (1995), pp. 1141–1149.

[5] Hongbin Ji and Ching-Yao Huang. *Non-cooperative uplink power control in cellular radio systems.* Wireless Networks, 4, (1998), pp. 233-240.

[6] Yannk A. Korilis, Aurel A. Lazar, Fellow, and Ariel Orda. *Architecting Noncooperative Networks.* IEEE Journal of selected areas in communications, 13(7), (1995), pp. 1241–1251.

M.A. Fonoberova, D.D. Lozovanu,

Received June 24, 2005

Institute of Mathematics and Computer Science,
5 Academiei str.
Chişinău, MD2028, Moldova.
E–mails: *mashaf83@yahoo.com*, *lozovanu@math.md*

Computer Science Journal of Moldova, vol.**13**, no.2(38), 2005

Measure of stability of a Pareto optimal solution to a vector integer programming problem with fixed surcharges in the l_1 and l_∞ metrics*

Vladimir A. Emelichev, Olga V. Karelkina, Kirill G. Kuzmin

Abstract

In this paper we consider a vector integer programming problem with Pareto principle of optimality for the case where partial criteria belong to the class of separable piecewise linear functions. The limit level of the initial data's perturbations in the space of vector criteria parameters with norms l_1 and l_∞, preserved Pareto optimality of the solutions is investigated. Formulas of the quasistability radius and of strong quasistability radius of the considered problem are given as corollaries.

AMS Mathematics Subject Classification:90C08, 90C10.

Keywords and phrases: vector integer programming problem, Pareto set, stability radius, quasistability and strong quasistability radii.

1 Introduction

Because of the extensive application of discrete optimization models in economics, management and design during past decades, much attention of many specialists has been given to the study of diverse aspects of stability and other questions relating to parametric and postoptimal analisis of scalar (singlecriterion) and vector (multicriterion) discrete optimization problems. Under stability of a problem in the wide

*Supported by state program of fundamental research "Mathematical structures 29" and program of the Ministry of Education "Fundamental and application studies" of the Republic of Belarus.

sense we understand the existence of a neighborhood in the space of the problem parameters such that any "perturbed" problem with parameters from this neighborhood possesses a certain kind of invariance with respect to the initial problem. Under the stability of solution we understand the property of solution to keep corresponding efficiency (optimality) under mentioned perturbations.

In this paper we consider a vector integer programming problem consisted in finding Pareto set. We suppose that all partial criteria of the problem are separable piecewise linear functions with fixed surcharges. The stability of the problem defined as the semicontinuity by Hausdorff of the optimal mapping that assigns the Pareto function of choice, was investigated earlier [1]. Lower and upper bounds of stability radius of the problem in the l_∞ metrics were obtained.

The purpose of this work is to obtain the limit level of perturbation in the space of vector criteria parameters with l_1 and l_∞ metrics preserving Pareto optimality (efficiency) of a given solution. Formulas of the quasistability and strong quasistability radii of the problem were also obtained.

2 Base definitions and properties

Let m be the number of criteria, n be the number of variables, $C = [c_{ij}] \in \mathbf{R}^{m\times n}$, $D = [d_{ij}] \in \mathbf{R}^{m\times n}$, X be a finite subset of $\mathbf{Z}^n_+ = \{x \in \mathbf{Z}^n : x_j \geq 0,\ j \in N_n\}$, $N_n = \{1, 2, \ldots, n\}$, where $|X| > 1$.

We define the vector criterion on the set of (feasible) solutions X

$$f(x) = (f_1(x), f_2(x), \ldots, f_m(x)) \to \min_{x\in X},$$

The components (partial criteria) are piecewise linear discontinuous functions with fixed surcharges

$$f_i(x) = \sum_{j=1}^{n} c_{ij}(x_j), \qquad i \in N_m,$$

where

$$c_{ij}(x_j) = \begin{cases} c_{ij}x_j + d_{ij}, & \text{if } x_j > 0, \\ 0, & \text{if } x_j = 0. \end{cases}$$

For any integer vector $x \in X$ we define a boolean vector $\tilde{x} \in \mathbf{E}^n = \{0,1\}^n$ with the components

$$\tilde{x}_j = \begin{cases} 1, & \text{if } x_j > 0, \\ 0, & \text{if } x_j = 0. \end{cases}$$

Then partial criteria are linear functions:

$$f_i(x) = C_i x + D_i \tilde{x}, \qquad i \in N_m,$$

where $x = (x_1, x_2, \dots, x_n)^T$, $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n)^T$, and the subscript at the matrix points to the corresponding row of the matrix. For example, $C_i = (c_{i1}, c_{i2}, \dots, c_{in})$.

Changing the elements of the pair (C, D), we obtain different vector criteria. Therefore the pair (C, D) can be used for indexing a vector criterion. Its partial criteria are denoted by $f_i(x, C_i, D_i)$.

Further under the vector (m-criteria) problem $Z^m(C, D)$ with fixed surcharges we understand the problem of finding of the Pareto set (the set of efficient solutions)

$$P^m(C, D) = \{x \in X : P^m(x, C, D) = \emptyset\},$$

where

$$P^m(x, C, D) = \{x' \in X : g(x, x', C, D) \leq 0_{(m)},\ g(x, x', C, D) \neq 0_{(m)}\},$$

$$g(x, x', C, D) = (g_1, g_2, \dots, g_m),$$

$$g_i = g_i(x, x', C_i, D_i) = f_i(x', C_i, D_i) - f_i(x, C_i, D_i), \qquad i \in N_m,$$

$$0_{(m)} = (0, 0, \dots, 0) \in \mathbf{R}^m.$$

While the set X is finite, the Pareto set $P^m(C, D)$ is nonempty for any matrices $C, D \in \mathbf{R}^{m \times n}$ and for any natural number $m \geq 1$.

Further we need the following evident statements.

Property 1. *Let $x \in X$, $x' \in P^m(x, C, D)$. Then for any index $i \in N_m$ the following inequality is valid*

$$g_i(x, x', C_i, D_i) \leq 0.$$

Property 2. *The solution x is efficient if for any solution $x' \neq x$ there exists an index $i \in N_m$, such that*

$$g_i(x, x', C_i, D_i) > 0.$$

Note, that the problem in the scalar case ($m = 1$) can be understood as the problem of piecewise linear concave programming with separable discontinuous function [2, 3]. It is obvious that in another particular case, where D is the null matrix, the problem $Z^m(C, D)$ changes into the m-criteria integer linear programming problem, different stability types of which were investigated in [4].

For any natural number p we define two metrics l_1 and l_∞ in the space $\mathbf{R}^p$, i.e. under metrics of a vector $y = (y_1, y_2, \ldots, y_p)$ we understand correspondingly the numbers

$$||y||_1 = \sum_{i=1}^{p} |y_i|, \qquad ||y||_\infty = \max\{|y_i| : \ i \in N_p\}.$$

Under the norm of a matrix we understand the norm of the vector composed from its elements.

The following properties are obvious for any index $i \in N_m$.

Property 3. $g_i(x, x', C_i, D_i) \leq (||C_i||_1 + ||D_i||_1)||x - x'||_\infty$.

Property 4. $g_i(x, x', C_i, D_i) \leq ||C_i||_\infty ||x - x'||_1 + ||D_i||_\infty ||\tilde{x} - \tilde{x}'||_1$.

Let $\varepsilon > 0$. According to the selected metric (l_1 or l_∞) in the parameter space $\mathbf{R}^{m\times n} \times \mathbf{R}^{m\times n}$ we perturb the elements of the pair (C, D) by addition this pair with a pairs (C', D') from the set

$$\Omega_1(\varepsilon) = \{(C', D') \in \mathbf{R}^{m\times n} \times \mathbf{R}^{m\times n} : ||C'||_1 + ||D'||_1 < \varepsilon\},$$

if the metric is l_1, and from the set

$$\Omega_\infty(\varepsilon) = \{(C', D') \in \mathbf{R}^{m\times n} \times \mathbf{R}^{m\times n} : ||C'||_\infty < \varepsilon, ||D'||_\infty < \varepsilon\},$$

if the metric is l_∞.

The problem $Z^m(C+C', D+D')$, obtained from $Z^m(C,D)$ by such addition, is called perturbed. The pair (C', D') is called perturbing.

By analogy with [5–8], under stability radius of the efficient solution $x \in P^m(C,D)$ of the problem $Z^m(C,D)$ we understand the number

$$\rho^m(x,C,D) = \begin{cases} \sup \Xi, & \text{if} \quad \Xi \neq \emptyset, \\ 0, & \text{otherwise}, \end{cases}$$

where

$$\Xi = \{\varepsilon > 0 : \ \forall\ (C', D') \in \Omega_k(\varepsilon) \quad (x \in P^m(C+C', D+D'))\}.$$

Here $k=1$ or $k=\infty$ according to the above mentioned notation.

Hence, the stability radius of the efficient solution $x \in P^m(C,D)$ is the maximum level of perturbations of the vector criterion parameters in space $\mathbf{R}^{m\times n}$ (with one of the norms), which keep the efficiency of the solution x.

3 Lemmas

By definition, put

$$g_i^+(x,x',C_i,D_i) = \max\{0, g_i(x,x',C_i,D_i)\}.$$

Lemma 1. *If the inequality*

$$g_i(x,x',C_i+C_i', D_i+D_i') \leq 0, \tag{1}$$

holds for any index $i \in N_m$, then

$$g_i^+(x,x',C_i,D_i) \leq (||C_i'||_1 + ||D_i'||_1)||x-x'||_\infty. \tag{2}$$

Actually, when $g_i(x,x',C_i,D_i) \leq 0$ inequality (2) is evident. If $g_i(x,x',C_i,D_i) > 0$, then taking into account condition (1), linearity of the function $g_i(x,x',C_i,D_i)$ and property 3 we deduce

$$g_i^+(x,x',C_i,D_i) = g_i(x,x',C_i,D_i) =$$

$$= g_i(x, x', C_i + C'_i, D_i + D'_i) - g_i(x, x', C'_i, D'_i) \leq$$

$$\leq -g_i(x, x', C'_i, D'_i) \leq (||C'_i||_1 + ||D'_i||_1)||x - x'||_\infty.$$

Lemma 1 is proved.

Lemma 2. *Let $x, x' \in X$, $x \neq x'$. For any φ, satisfying the inequalities*

$$0 < \varphi||x - x'||_\infty \leq \sum_{i \in N_m} g_i^+(x, x', C_i, D_i), \qquad (3)$$

and for any perturbing pair $(C', D') \in \Omega_1(\varphi)$ the following ratio is valid

$$x' \notin P^m(x, C + C', D + D').$$

Proof. Suppose the opposite, i.e. there exists perturbing pair $(C', D') \in \Omega_1(\varphi)$, such that $x' \in P^m(x, C+C', D+D')$. Then according to property 1 for any index $i \in N_m$ the inequality (1) is true. Therefore based on lemma 1 the inequality (2) is true. Hence, using inclusion $(C', D') \in \Omega_1(\varphi)$, we conclude

$$\sum_{i \in N_m} g_i^+(x, x', C_i, D_i) \leq \sum_{i \in N_m} (||C'_i||_1 + ||D'_i||_1)||x - x'||_\infty \leq$$

$$\leq (||C'||_1 + ||D'||_1)||x - x'||_\infty < \varphi||x - x'||_\infty,$$

that contradicts to condition (3).

Lemma 2 is proved.

Lemma 3. *Let $x, x' \in X$, $x \neq x'$. For any number ε such that*

$$\varepsilon > \sum_{i \in N_m} \eta_i,$$

where

$$\eta_i||x - x'||_\infty > g_i^+(x, x', C_i, D_i), \qquad i \in N_m, \qquad (4)$$

there exists perturbing pair $(C', D') \in \Omega_1(\varepsilon)$, such that $x' \in P^m(x, C + C', D + D')$.

Proof. At first note, that any number ε, under conditions of the lemma, is positive since all the numbers $\eta_i, i \in N_m$. Obviously that for proving our lemma it suffices to show the perturbing pair $(C', D') \in \Omega_1(\varepsilon)$, such that following inequalities

$$g_i(x, x', C_i + C'_i, D_i + D'_i) < 0, \qquad i \in N_m \tag{5}$$

are fulfilled. Let

$$q = \arg\max\{|x_j - x'_j| : \ j \in N_n\}$$

and define the elements of the perturbing pair (C', D') by formulas

$$c'_{ij} = \begin{cases} \eta_i \cdot \operatorname{sign}(x_q - x'_q), & \text{if } i \in N_m, \ j = q, \\ 0 & \text{otherwise,} \end{cases}$$

$$d'_{ij} = 0, \qquad i \in N_m, \ \ j \in N_n.$$

It is easy to see that $(C', D') \in \Omega_1(\varepsilon)$. By virtue of construction of the pair (C', D') the equalities

$$g_i(x, x', C'_i, D'_i) = -\eta_i ||x - x'||_\infty, \qquad i \in N_m$$

are true. Thus, taking into account the linearity of $g_i(x, x', C_i, D_i)$ and ratios (4), we make sure of correctness of inequality (5):

$$g_i(x, x', C_i + C'_i, D_i + D'_i) = g_i(x, x', C_i, D_i) + g_i(x, x', C'_i, D'_i) =$$

$$= g_i(x, x', C_i, D_i) - \eta_i ||x - x'||_\infty \leq$$

$$\leq g_i^+(x, x', C_i, D_i) - \eta_i ||x - x'||_\infty < 0, \qquad i \in N_m.$$

Lemma 3 is proved.

Lemma 4. *If for any index $i \in N_m$ the inequality*

$$g_i(x, x', C_i, D_i) > \max\{||C'_i||_\infty, ||D'_i||_\infty\}(||x - x'||_1 + ||\tilde{x} - \tilde{x}'||_1),$$

holds, then

$$g_i(x, x', C_i + C'_i, D_i + D'_i) > 0.$$

Proof, from property 4, combining it with the linearity of the function $g_i(x, x', C_i, D_i)$, and condition of the lemma we obtain

$$g_i(x, x', C_i + C'_i, D_i + D'_i) = g_i(x, x', C_i, D_i) + g_i(x, x', C'_i, D'_i) \geq$$
$$\geq g_i(x, x', C_i, D_i) - ||C'_i||_\infty ||x - x'||_1 - ||D'_i||_\infty ||\tilde{x} - \tilde{x}'||_1 \geq$$
$$\geq g_i(x, x', C_i, D_i) - \max\{||C'_i||_\infty, ||D'_i||_\infty\}(||x - x'||_1 + ||\tilde{x} - \tilde{x}'||_1) > 0.$$

Lemma 4 is proved.

Lemma 5. *Let $x, x' \in X$, $x \neq x'$. For any number ε such that*

$$\varepsilon(||x-x'||_1+||\tilde{x}-\tilde{x}'||_1) > \max\{g_i(x, x', C_i, D_i) : \ i \in N_m\}, \quad \varepsilon > 0 \quad (6)$$

there exists a perturbing pair $(C', D') \in \Omega_\infty(\varepsilon)$, such that $x' \in P^m(x, C + C', D + D')$.

Proof. Obviously that for proving our lemma it suffices to show the perturbing pair $(C', D') \in \Omega_\infty(\varepsilon)$, such that following inequalities

$$g_i(x, x', C_i + C'_i, D_i + D'_i) < 0, \qquad i \in N_m \qquad (7)$$

are true.

By virtue of (6) there exists a number α, such that

$$0 < \alpha < \varepsilon,$$
$$\alpha(||x - x'||_1 + ||\tilde{x} - \tilde{x}'||_1) > \max\{g_i(x, x', C_i, D_i) : \ i \in N_m\}. \qquad (8)$$

We assign the elements of the perturbing pair $(C', D') \in \Omega_\infty(\varepsilon)$ by the rule:

$$c'_{ij} = \alpha \cdot \operatorname{sign}(x_j - x'_j), \quad d'_{ij} = \alpha \cdot \operatorname{sign}(\tilde{x}_j - \tilde{x}'_j), \qquad i \in N_m, j \in N_n.$$

Then taking into account the linearity of the function $g_i(x, x', C_i, D_i)$, combining it with evident inequalities

$$g_i(x, x', C'_i, D'_i) = -\alpha(||x - x'||_1 + ||\tilde{x} - \tilde{x}'||_1), \qquad i \in N_m,$$

and inequality (8), we obtain:

$$g_i(x, x', C_i + C'_i, D_i + D'_i) = g_i(x, x', C_i, D_i) + g_i(x, x', C'_i, D'_i) =$$
$$= g_i(x, x', C_i, D_i) - \alpha(||x - x'||_1 + ||\tilde{x} - \tilde{x}'||_1) < 0, \qquad i \in N_m.$$

Lemma 5 is proved.

4 Formulas of the stability radius of an efficient solution

Theorem. *For any number $m \geq 1$ the stability radius $\rho^m(x, C, D)$ of any efficient solution $x \in P^m(C, D)$ of the problem $Z^m(C, D)$ is expressed by the formula*

$$\rho^m(x, C, D) =$$

$$= \min_{x' \in X \setminus \{x\}} \begin{cases} \sum\limits_{i \in N_m} \dfrac{g_i^+(x, x', C_i, D_i)}{||x - x'||_\infty}, & \textit{if the metric is } l_1, \\ \max\limits_{i \in N_m} \dfrac{g_i(x, x', C_i, D_i)}{||x - x'||_1 + ||\tilde{x} - \tilde{x}'||_1}, & \textit{if the metric is } l_\infty. \end{cases} \tag{9}$$

Proof. It is evident that in the right part of the equation (9) we have non-negative numbers.

1. The case of l_1 metric. First let us prove the inequality $\rho^m(x, C, D) \geq \varphi_1$. We see that it suffices to consider the case $\varphi_1 > 0$. By definition of the number φ_1, for any solution $x' \neq x$ inequalities (3) are correct. Hence based on lemma 2 for any perturbing pair $(C', D') \in \Omega_1(\varphi_1)$ solution $x' \notin P^m(x, C + C', D + D')$. Therefore the set $P^m(x, C + C', D + D') = \emptyset$. Thus for any perturbing pair $(C', D') \in \Omega_1(\varphi_1)$ solution $x \in P^m(C + C', D + D')$. Consequently $\rho^m(x, C, D) \geq \varphi_1$.

Now we show that $\rho^m(x, C, D) \leq \varphi_1$. Let $\varepsilon > \varphi_1$. According to definition of the number φ_1 there exists a solution $x^* \neq x$, such that

$$\varphi_1 ||x - x^*||_\infty = \sum_{i \in N_m} g_i^+(x, x^*, C_i, D_i).$$

Therefore there exists such positive numbers η_i that

$$\eta_i ||x - x^*||_\infty > g_i^+(x, x^*, C_i, D_i), \qquad i \in N_m,$$

$$\varepsilon > \sum_{i \in N_m} \eta_i > \varphi_1.$$

Hence by lemma 3 there exists a perturbing pair $(C', D') \in \Omega_1(\varepsilon)$ such that $x^* \in P^m(x, C + C', D + D')$, i.e. the solution $x \notin P^m(C + C', D +$

$D')$. Hence for any number $\varepsilon > \varphi_1$ the inequality $\rho^m(x, C, D) < \varepsilon$ holds. Thus $\rho^m(x, C, D) \leq \varphi_1$.

2. The case of l_∞ metric. First prove the inequality $\rho^m(x, C, D) \geq \varphi_\infty$. Without loss of generality it can be assumed that $\varphi_\infty > 0$. By definition of value φ_∞ for any perturbing pair $(C', D') \in \Omega_\infty(\varphi_\infty)$ and any solution $x' \neq x$ there exists index $i \in N_m$, such that

$$\frac{g_i(x, x', C_i, D_i)}{||x - x'||_1 + ||\tilde{x} - \tilde{x}'||_1} \geq \varphi_\infty > \max\{||C'||_\infty, ||D'||_\infty\}.$$

From this according to lemma 4 we have

$$g_i(x, x', C_i + C'_i, D_i + D'_i) > 0.$$

Thus, taking into account property 2, solution x belongs to the Pareto set of the perturbed problem $Z^m(C + C', D + D')$. Consequently $\rho^m(x, C, D) \geq \varphi_\infty$.

Further we prove that $\rho^m(x, C, D) \leq \varphi_\infty$. According to the definition of number φ_∞ there exists a solution $x^* \neq x$, such that

$$\varphi_\infty(||x - x^*||_1 + ||\tilde{x} - \tilde{x}^*||_1) = \max\{g_i(x, x^*, C_i, D_i) : \ i \in N_m\}.$$

Then for $\varepsilon > \varphi_\infty$ inequality (6) is fulfilled. Hence based on lemma 5 there exists a perturbing pair $(C', D') \in \Omega_\infty(\varepsilon)$, such that $x^* \in P^m(x, C + C', D + D')$, i.e. the solution $x \notin P^m(C + C', D + D')$. Thereby it is proved that for any number $\varepsilon > \varphi_\infty$ the inequality $\rho^m(x, C, D) < \varepsilon$ is valid. Hence $\rho^m(x, C, D) \leq \varphi_\infty$.

Theorem is proved.

5 Corollaries

Under the quasistability of a vector problem of discrete optimization we usually understand [4, 9–12] the Hausdorff lower semicontinuity of the set-valued (point-to-set) mapping determining the Pareto choice function. In other words, the problem $Z^m(C, D)$ is called quasistability if there exists a number $\varepsilon > 0$, such that for any perturbing pair $(C', D') \in \Omega(\varepsilon)$ the following conclusion is valid

$$P^m(C, D) \subseteq P^m(C + C', D + D').$$

Therefore quasistability radius of the problem is determined as follows:

$$\rho_1^m(C,D) = \begin{cases} \sup \Phi_1, & \text{if } \Phi_1 \neq \emptyset, \\ 0, & \text{otherwise}, \end{cases}$$

where

$$\Phi_1 = \{\varepsilon > 0 : \ \forall\, (C',D') \in \Omega_k(\varepsilon) \quad (P^m(C,D) \subseteq P^m(C+C',D+D'))\},$$

$\Omega_k(\varepsilon)$ is the above mentioned set of perturbing pairs (C',D'). In other words, quasistability radius of the problem $Z^m(C,D)$ is the maximum level of perturbations of matrices C and D in the space of parameters of the vector criterion with the corresponding norm such that Pareto set can only expand.

Directly from the theorem we obtain

Corollary 1. *For any $m \geq 1$ for quasistability radius $\rho_1^m(C,D)$ of the problem $Z^m(C,D)$ the following formula is valid*

$$\rho_1^m(C,D) =$$

$$= \min_{x \in P^m(C,D)} \min_{x' \in X \setminus \{x\}} \begin{cases} \sum\limits_{i \in N_m} \dfrac{g_i^+(x,x',C_i,D_i)}{||x-x'||_\infty}, & \textit{if the metric is } l_1, \\ \max\limits_{i \in N_m} \dfrac{g_i(x,x',C_i,D_i)}{||x-x'||_1 + ||\tilde{x}-\tilde{x}'||_1}, & \textit{if the metric is } l_\infty. \end{cases}$$

Therefore next corollary is true.

Corollary 2. *For any $m \geq 1$ the problem $Z^m(C,D)$ is quasistable if and only if $P^m(C,D) = S^m(C,D)$.*

Here $S^m(C,D)$ is the traditional Smale set, i.e. the set of strictly efficient solutions of the problem $Z^m(C,D)$, which is a subset of the Pareto set and defined in the following way [13]:

$$S^m(C,D) = \{x \in X : \ S^m(x,C,D) = \emptyset\},$$

where

$$S^m(x,C,D) = \{x' \in X \setminus \{x\} : \quad g(x,x',C,D) \leq 0_{(m)}\}.$$

When we relax the demand of preservation of all the Pareto set in definition of the quasistability of problem $Z^m(C,D)$, we get the concept of the strong quasistability. This type of the stability means the existence of neighborhood of vector criterion parameters such that although disappearance of the old effective solutions is possible but there exists at least one pareto-optimal solution of initial problem, that preserves its efficiency under small perturbations of parameters. In other words there exists at least one stable Pareto optimum. Thus under the strong quasistability radius of the problem $Z^m(C,D)$ we understand the number

$$\rho_2^m(C,D) = \begin{cases} \sup\Phi_2, & \text{if } \Phi_2 \neq \emptyset, \\ 0, & \text{otherwise,} \end{cases}$$

where

$$\Phi_2 = \{\varepsilon > 0 :\ \exists\, x \in P^m(C,D)\ \ \forall\, (C',D') \in \Omega_k(\varepsilon)$$

$$(x \in P^m(C+C', D+D'))\}.$$

Directly from the theorem we obtain

Corollary 3. *For strong quasistability radius* $\rho_2^m(C,D)$, $m \geq 1$ *of the problem* $Z^m(C,D)$ *the following formula is valid*

$$\rho_2^m(C,D) =$$

$$= \max_{x \in P^m(C,D)} \min_{x' \in X\setminus\{x\}} \begin{cases} \sum\limits_{i \in N_m} \dfrac{g_i^+(x,x',C_i,D_i)}{||x-x'||_\infty}, & \textit{if the metric is } l_1, \\ \max\limits_{i \in N_m} \dfrac{g_i(x,x',C_i,D_i)}{||x-x'||_1 + ||\tilde{x}-\tilde{x}'||_1}, & \textit{if the metric is } l_\infty. \end{cases}$$

Hence we obtain

Corollary 4. *For any* $m \geq 1$ *the problem* $Z^m(C,D)$ *is strongly quasistable problem if and only if* $S^m(C,D) \neq \emptyset$.

The next two statements follow from corollaries 3 and 4.

Corollary 5. *Any quasistable problem* $Z^m(C, D), m \geq 1$, *is strongly quasistable.*

Corollary 6. *For any scalar problem* $Z^1(C, D), C \in \mathbf{R}^n, D \in \mathbf{R}^n$ *the next statements are equivalent:*
a) $Z^1(C, D)$ *is quasistable,*
b) $Z^1(C, D)$ *is strongly quasistable,*
c) $Z^1(C, D)$ *has a unique optimal solution.*

6 Example

Let us give a simple example which illustrates stated results. Let us consider a two-criterion problem. $X = \{x^1, x^2, x^3, x^4, x^5\}$, where

$$x^1 = (1, 0, 0, 0)^T, \quad x^2 = (0, 1, 0, 0)^T, \quad x^3 = (0, 0, 1, 0)^T,$$

$$x^4 = (2, 3, 1, 1)^T, \quad x^5 = (2, 4, 1, 1)^T,$$

the matrices C and D are

$$C = \begin{bmatrix} 2 & 5 & 5 & 3 \\ 7 & 1 & 1 & 2 \end{bmatrix}, \qquad D = \begin{bmatrix} 1 & 3 & 3 & 0 \\ 2 & 1 & 1 & 4 \end{bmatrix}.$$

Then

$$f(x^1) = (3, 9), \qquad f(x^2) = f(x^3) = (8, 2),$$

$$f(x^4) = (34, 28), \qquad f(x^5) = (39, 29),$$

Pareto set consists of three solutions x^1, x^2, x^3, Smale set contains just one solution x^1. Therefore the problem is quasistable (in virtue of corollary 2), but is not strongly quasistable (in virtue of corollary 4). By the theorem it is easy to calculate

$$\rho^2(x^1, C, D) = \begin{cases} 5, & \text{if the metric is } l_1, \\ \frac{5}{4}, & \text{if the metric is } l_\infty, \end{cases} \tag{10}$$

$$\rho^2(x^2, C, D) = \rho^2(x^3, C, D) = 0 \quad \text{for any metric } (l_1 \text{ or } l_\infty).$$

Hence in virtue of corollaries 1 and 3 the quasistability and strong quasistability radii take on the forms correspondingly

$$\rho_1^2(C,D) = 0 \quad \text{for any metric } (l_1 \text{ or } l_\infty),$$

$$\rho_2^2(C,D) = \begin{cases} 5, & \text{if the metric is } l_1, \\ \frac{5}{4}, & \text{if the metric is } l_\infty. \end{cases}$$

Now consider two new another variants of the problem changing just the set of the admissible solutions X.

Variant 1. Let $X' = X \setminus \{x^3\} = \{x^1, x^2, x^4, x^5\}$. Then the Pareto set $\{x^1, x^2\}$ and the Smale set are congruent. Therefore the problem is quasistable and strong quasistable simultaneously. Using formula (9), we make sure that the value $\rho^2(x^1, C, D)$ is defined by formula (10) too.

$$\rho^2(x^2,C,D) = \begin{cases} 7, & \text{if the metric is } l_1, \\ \frac{7}{4}, & \text{if the metric is } l_\infty. \end{cases}$$

Taking into account corollaries 1 and 3, we get the quasistability and strong quasistability radii:

$$\rho_1^2(C,D) = \begin{cases} 5, & \text{if the metric is } l_1, \\ \frac{5}{4}, & \text{if the metric is } l_\infty, \end{cases}$$

$$\rho_2^2(C,D) = \begin{cases} 7, & \text{if the metric is } l_1, \\ \frac{7}{4}, & \text{if the metric is } l_\infty. \end{cases}$$

Variant 2. Let $X'' = X \setminus \{x^1\} = \{x^2, x^3, x^4, x^5\}$. Then $\{x^2, x^3\}$ is the Pareto set. The Smale set is empty. By the formula (9) we have

$$\rho^2(x^2, C, D) = \rho^2(x^3, C, D) = 0 \quad \text{for any metric.}$$

Therefore in virtue of corollaries 2 and 4, the problem is neither quasistable nor strongly quasistable.

References

[1] Bukhtoyarov S. E., Emelichev V. A. *Stability radius of a vector integer programming problem with fixed surcharges,* Studii în analiză numerică şi optimizare, Chişinău, 2001, vol. 2, no. 2(6), pp. 152–156.

[2] Lozovanu D. D. *Extremely-combinatoric problems and algorithms of their solutions,* Kishinev, Ştiinţa, 1991, 222 p. (in Russian)

[3] Lozovanu D. D., Solomon D. I. *Nonlinear optimization problem. Algorithms and complexity*, Kishinev, Evrika, 1996, 208 p. (in Russian)

[4] Emelichev V. A., Girlich E., Nikulin Yu. V., Podkopaev D. P. *Stability and regularization of vector problems of integer linear programming*, Optimization, 2002, vol. 51, no. 4, pp. 645–676.

[5] Bukhtoyarov S. E., Emelichev V. A., Stepanishina Y. V. *Stability of discrete vector problems with the parametric principle of optimality*, Cybernetics and System Analysis, 2003, vol. 39, no. 4, pp. 604–614.

[6] Emelichev V. A., Kuz'min K. G., Leonovich A. M. *Stability in the combinatorial vector optimization problems*, Automatic and Remote Control, 2004, vol. 65, no. 2, pp. 227–240.

[7] Emelichev V. A., Krichko V. N., Nikulin Y. V. *The stability radius of an efficient solution in minimax Boolean programming problem*, Control and Cybernetics, Warszawa, 2004, vol. 33, no. 1, pp. 127–132.

[8] Emelichev V. A., Kuz'min K. G., *The stability radius of efficient solution to a vector problem of boolean programming in the l_1 metric*, Doclady Mathematics, 2005, vol. 71, no. 2, pp. 266–268.

[9] Emelichev V. A., Podkopaev D. P. *Condition of stability, pseudostability and quasi-stability of the Pareto set in a vector trajectorial problem*, Revue d'Analyse Numérique et de Theorie de l'Approximation, Cluj-Napoca, Romania, 1998, t. XXVII, no. 1, pp. 91–97.

[10] Emelichev V. A., Kravtsov M. K. Podkopaev D. P. *On the quasistability of trajectory problems of vector discrete optimization*, Mathematical Notes, 1998, vol. 63, no. 1, pp. 19–24.

[11] Emelichev V. A., Podkopaev D. P. *On a quantitative measure of stability for a vector problem in integer programming*, Comp. Math. and Math. Physics, 1998, vol. 38, no. 11, pp. 1727–1731.

[12] Emelichev V. A., Leonovich A. M. *Quasistability of the vector l_∞-extreme combinatorial problem with Pareto principle of optimality*, Buletinul Acad. de St. a Republicii Moldova, Matematica, 2000, no. 1, pp. 33–44.

[13] Smale S. *Global analysis and economics, V. Pareto theory with constraints*, J. Math. Econ. 1974. no 1. pp. 213–221.

[14] Emelichev V. A., Nikulin Y. V. *Numerical measure of strong stability and strong quasistability in the vector problem of integer linear programming*, Computer Science Journal of Moldova, 1999, vol. 7, no. 1, pp. 105–117.

[15] Emelichev V. A., Pokhilka V. G. *On strong quasistability of a vector problem on substitutions*, Computer Science Journal of Moldova, 2001, vol. 9, no. 1, pp. 71–85.

[16] Berdysheva R. A., Emelichev V. A. *Strong stability and strong quasistability of vector trejectorial problem of lexicographic optimization*, Computer Science Journal of Moldova, 1998, vol. 6, no. 2, pp. 119–136.

Received June 24, 2005

V.A. Emelichev, O.V. Karelkina,
Kirill G. Kuzmin,

Belorussian State University,
ave. F. Skoriny, 4,
Minsk, 220050, Belarus.
E–mail: *emelichev@bsu.by*, *okarel@mail.ru*, *kuzminkg@mail.ru.*

Computer Science Journal of Moldova, vol.**13**, no.3(39), 2005

Approaches to Software Based Fault Tolerance – A Review

Goutam Kumar Saha

Abstract

This paper presents a review work on various approaches to software based fault tolerance. The aim of this paper is to cover past and present approaches to software implemented fault tolerance that rely on both software design diversity and on single but enhanced design.

1 Introduction

Fault tolerance is the ability of a system to perform its function correctly even in the presence of internal faults. The purpose of fault tolerance is to increase the dependability of a system. The objective of a fault-tolerant system is to mask faults (or to detect errors to switch to an alternate module) and continue to provide service despite faults. Fault tolerant systems must provide their specified services despite the occurrence of faults in the systems's components [1]. A failure occurs when an actual running system deviates from the specified behaviour. The cause of a failure is called an error. A fault is the root cause of a failure. In other words, an error is merely the symptom of a fault. A fault may not necessarily result in an error, but the same fault may result in multiple errors. Similarly, a single error may lead to multiple failures [116]. All fault tolerance techniques must use some form of redundancy to tolerate faults. Depending on the class of faults [76] redundant devices, networks, data or applications are used. Software fault tolerance relies either on design diversity or on single design using robust data structure. The only type of fault possible in software is a design fault introduced during the software development. Software faults are what we commonly call “bugs”. According to [74], software faults are the root cause in a high percentage of operational system failures. The consequences of these failures depend on the application and the particular characteristic of the faults. The immediate effects can range from minor inconveniences (e.g., having to restart a hung personal computer) to catastrophic events (e.g., software in an aircraft that prevents the pilot from recovering from an input error) [75]. From a business perspective, operational failures caused by software faults can translate into loss of potential customers, lower sales, higher warranty repair costs, and losses due to legal actions from the people affected by the failures. There are four ways of dealing with software faults: prevention, removal, fault tolerance, and input sequence workarounds. Fault prevention is concerned with the use of design methodologies, techniques, and technologies aimed at preventing the introduction of faults into the design. Fault removal considers the use of techniques like reviews, analyses, and testing to check an implementation and remove any faults thereby exposed. The proper use of software engineering during the development processes is a way of realizing fault prevention and fault removal (i.e., fault avoidance). The use of fault avoidance is the standard approach for dealing with software faults and the many developments in the software field target the improvement of the fault avoidance techniques. [74,76] states that the software development process usually removes most of the deterministic design faults. This type of fault is activated by the inputs independently of the internal state of the software. A large number of the faults in operational software are state-dependent faults activated by particular input sequences. Given the lack of techniques that can guarantee that complex software designs are free of design fault, fault tolerance is sometimes used as an extra layer of protection. Software fault tolerance is the use of techniques to enable the continued delivery of services at an acceptable level of performance and safety after a design fault becomes active. Single version technique aims to improve the fault tolerance of a single piece of software by adding extra measures into the design aiming the error detection, containment, and tolerating errors caused by the design faults. Multi-version fault tolerance technique uses multiple versions (or variants) of a piece of software in a structured way to ensure that design faults in one version do not cause system failures. This is based on masking the design bugs. A characteristic of the software fault tolerance techniques is that they can be applied at any level in a software system: procedure, process, full application program, or the whole system including the operating system [4]. Such technique can also be applied selectively to those components that are deemed to have design faults due to their complexity [5]. Whatever measure we may take to remove software design bugs, software cannot be made free of design bugs. Though at present software errors have been attributed to be the main cause of most system failures. Transient errors often disrupt proper functioning (e.g., program hanging, wrong answer, false branching, data and code errors etc.) of an application program during the execution of an application system. But recent studies [2,3] have suggested that soft errors or transient bit-errors are increasingly responsible for system malfunctioning. Nowadays, computer systems are becoming more complex and are optimized for price and performance and not for availability. This makes soft errors an even more common case. Move towards denser, smaller, and low voltage transistors has the potential to increase these transient errors. Most system software architectures assume faith in underlying hardware, and software make no provisions to deal with hardware faults. [101] predicts that soft error rate (SER) per chip of logic circuits will increase nine orders of magnitude from 1992 to 2011 and at that point magnitude will be comparable to the SER per chip of unprotected memory elements. Researches in [104] have analyzed the effect of soft errors on system software. Software fault tolerance techniques have also been proposed in [102,103]. In this survey paper, we not only investigate various approaches to tolerate software design bugs but also we investigate the consequence of soft error on the system as a whole and current research into proposed recovery mechanisms along with transient fault tolerance. Multi version approach is basically on the assumption that software built differently might fail differently and thus, if one of the redundant versions fails, at least one of the others might provide an acceptable output. Recovery blocks, N-version programming, N self-checking programming, consensus recovery blocks, and $t/(n-1)$ techniques are also reviewed. Current research in software engineering focuses on establishing patterns in the software structure and trying to understand the practice of software engineering. We expect that software based fault tolerance research will be benefited by this research on enabling greater predictability of the dependability of software.

2 Various Approaches to Software Fault Tolerance

In this section, we investigate various software based fault tolerant approaches that rely on design diversity (multiple version) as well as on single design.

2.1 Design Diversity Based Software Fault Tolerance

Design Diversity Based or Multiple version based software fault tolerance is based on the use of at least two versions (or "variants") of a piece of software, executed either in sequence or in parallel. The versions are used as alternatives (with a separate means of error detection), in pairs (to implement detection by replication checks) or in larger groups (to enable masking through voting). The rationale for the use of multiple versions is the expectation that components built differently (i.e, different designers, different algorithms, different design tools, etc) should fail differently [6,7]. Therefore, if one version fails on a particular input, at least one of the alternate versions should be able to provide an appropriate output. This section covers some of these "design diversity" approaches to software reliability and safety. Basically multiple-version approach is to mask software design bugs. Two critical issues in the use of multi-version software fault tolerance techniques are the guaranteeing of independence of failure of the multiple versions and the development of the output selection algorithms.

Design diversity is "protection against uncertainty" [16]. In the case of software design, the uncertainty is in the presence of design faults and the failure modes due to those faults. The goal of design diversity techniques applied to software design is to build program versions that fail independently and with low probability of coincidental failures. If this goal is achieved, the probability of not being able to select a good output at a particular point during program execution is greatly reduced or eliminated. Due to the complexity of software, the use of design diversity for software fault tolerance is today more of an art rather than a science. The methodology of multiple-version software design was carried out by Algirdas Avizienis and his colleagues at UCLA starting in the 1970s [6,17,18,19,20,21,22,23,24]. Although focused mainly on software, their research considered the use of design diversity concepts for other aspects of systems like the operating system, the hardware, and the user interfaces. Assuming that the development is rigorous and design diversity is adequately applied to the product, there is still the common error source of the identical input profile. [25] points out that experiments (e,g, [26,27,28]) have shown that the probability of error manifestations are not equally distributed over the input space and the probability of coincident errors is impacted by the chosen inputs. Certainly data diversity techniques could be used to reduce the impact of this error source, but the problem of quantifying the effectiveness of the approach still remains. The cost of using multi-version software is also an important issue. A direct replication of the full development effort, including testing, would certainly be an expensive proposition. In some applications where only a small part of the functionality is safety critical, development and production cost can be reduced by applying design diversity only to those critical parts [16].

- **The Recovery Block Scheme**

The Recovery Block Scheme (RBS) technique [8,9] combines the basics of both the checkpoint and restart approach with multiple versions of a software component such that a different version is tried after an error is detected. Checkpoints are created before a version executes. Checkpoints are needed to recover the state after a version fails to provide a valid operational starting point for the next version if an error is detected. The acceptance test need not be an output-only test and can be implemented by various embedded checks to increase the effectiveness of the error detection. Also, because the primary version will be executed successfully most of the time, the alternates could be designed to provide degraded performance in some sense (e.g., by computing values to a lesser accuracy). Actual execution of the multiple versions can be sequential or in parallel depending on the available processing capability and performance requirements. If all the alternates are tried unsuccessfully, the component must raise an exception to communicate to the rest of the system its failure (or crash) to complete its function. Note that such a failure occurrence does not imply a permanent failure of the component, which may be reusable after changes in its inputs or state. The much possibility of coincident faults is the source of much controversy concerning all the multi-version software fault tolerance techniques.

- **The N-Version Programming Scheme**

The N-Version programming Scheme (NVPS) [7] is a multiple-version technique in which all the versions are designed to satisfy the same basic requirements and the decision of output correctness is based on the comparison of all the outputs. The use of a generic decision algorithm (usually a voter) to select the correct output is the fundamental difference of this approach from the Recovery Blocks approach, which requires an application dependent acceptance test. Since all the versions are built to satisfy the same requirements, the use of N-version programming requires considerable development effort but the complexity (i.e., development difficulty) is not necessarily much greater than the inherent complexity of building a single version. Design of the voter can be complicated by the need to perform inexact voting. [43] presents a generic two step structure for the output selection process. The first step is a filtering process where individual version outputs are analyzed by acceptance tests for likelihood of correctness, timing, completeness, and other characteristics. [44] presents four generalized voters for use in redundant systems: Formalized Majority Voter, Generalized Median Voter, Formalized Plurality Voter, and Weighted Averaging Techniques. The Weighted Averaging Technique combines the version outputs in a weighted average to produce a new output. The weights can be selected a-priori based on the characteristics of the individual versions and the application. When all the weights are equal this technique becomes a mean selection technique. The weights can also be selected dynamically based on the pair-wise distances of the version outputs [48] or the success history of the versions measured by some performance metric [44,47]. Other voting techniques have been proposed. For example, [45] proposed a selection function that always produces an acceptable output through the use of artificial intelligence techniques. [49, 50, 51] discuss the voting or majority output problem in some detail.

- **The N Self-Checking Programming Scheme**

The N Self-Checking Programming Scheme (NSCPS) [10,11,12] is the use of multiple software versions combined with structural variations of the Recovery Blocks and N-Version Programming. N Self-Checking programming uses acceptance tests. Here the versions and the acceptance tests are developed independently from common requirements. This use of separate acceptance tests for each version is the main difference of this N Self-Checking model from the Recovery Blocks approach. Similar to N-Version Programming, this model has the advantage of using an application independent decision algorithm to select a correct output.

- **The Consensus Recovery Blocks Scheme**

The Consensus Recovery Blocks Scheme (CRBS) [13] approach combines N-Version Programming and Recovery Blocks to improve the reliability over that achievable by using just one of the approaches. The acceptance tests in the Recovery Blocks suffer from lack of guidelines for their development and a general proneness to design faults due to the inherent difficulty in creating effective tests. The use of voters as in N-Version Programming may not be appropriate in all situations, especially when multiple correct outputs are possible. In that case a voter, for example, would declare a failure in selecting an appropriate output. Consensus Recovery Blocks uses a decision algorithm similar to N-Version Programming as a first layer of decision. If this first layer declares a failure, a second layer using acceptance tests similar to those used in the Recovery Blocks approach is invoked. Although obviously much more complex than either of the individual techniques, the reliability models indicate that this combined approach has the potential of producing a more reliable piece of software.

- **The $t/(n-1)$-Variant Programming Scheme**

The $t/(n-1)$-Variant Programming Scheme (VPS) was proposed in [14]. The main difference between this approach and the ones mentioned above is in the mechanism used to select the output from among the multiple variants. The design of the selection logic is based on the theory of system-level fault diagnosis [15], which is beyond the scope of this paper. Basically, a $t/(n-1)$-VPS architecture consists of n variants and uses the $t/(n-1)$ diagnosibility measure to isolate the faulty units to a subset of size at most $(n-1)$ assuming there are at most t faulty units [14]. Thus, at least one non-faulty unit exists such that its output is correct and can be used as the result of computation for the module. $t/(n-1)$-VPS compares favorably with other approaches in that the complexity of the selection mechanism grows with order $O(n)$ and it can potentially tolerate multiple dependent faults among the versions. It also has a lower probability of failure than N Self-Checking Programming and N-Version Programming when they use a simple voter as selection logic.

2.2 Single-Design Software Fault Tolerance Approach

Single-design fault tolerance is based on the use of redundancy applied to a single version of a piece of software to detect and recover from faults. Among others, single-version software fault tolerance techniques include considerations on program structure and actions, error detection, exception handling, checkpoint and restart, process pairs, and data diversity [29].

- **Software Engineering Aspects**

Software architecture gives us the basis for implementation of fault tolerance. The use of modularizing techniques to decompose a problem into manageable components is as important to the efficient application of fault tolerance as it is to the design of a system. The modular decomposition of a design should consider built-in protections to keep aberrant component behavior in one module from propagating to other modules. Control hierarchy issues like visibility (i.e., the set of components that may be invoked directly and indirectly by a particular component) and connectivity (i.e., the set of components that may be invoked directly or used by a given component) should be considered in the context of error propagation for their potential to enable uncontrolled corruption of the system state. Partitioning is a technique for providing isolation between functionally independent modules [31]. Advantages of using partitioning in a design include simplified testing, easier maintenance, and lower propagation of side effects [30]. System closure is a fault tolerance principle stating that no action is permissible unless explicitly authorized [32]. An atomic action among a group of components is an activity in which the components interact exclusively with each other and there is no interaction with the rest of the system for the duration of the activity [33]. The advantage of using atomic actions in defining the interaction between system components is that they provide a framework for error confinement and recovery. There are only two possible outcomes of an atomic action: either it terminates normally or it is aborted upon error detection. If an atomic action terminates normally, its results are complete and committed. If a failure is detected during an atomic action [76], it is known beforehand that only the participating components can be affected.

- **Error Detection Mechanisms**

Effective application of fault tolerance techniques in single version systems requires that the structural modules have two basic properties: self-protection and self-checking [34]. The self-protection property means that a component must be able to protect itself from external contamination by detecting errors in the information passed to it by other interacting components. Self-checking means that a component must be able to detect internal errors and take appropriate actions to prevent the propagation of those errors to other components. The degree (and coverage) to which error detection mechanisms are used in a design is determined by the cost of the additional redundancy and the run-time overhead. Note that the fault tolerance redundancy is not intended to contribute to system functionality but rather to the quality of the product. Similarly, detection mechanisms detract from system performance. Actual usage of fault tolerance in a design is based on trade-offs of functionality, performance, complexity, and safety. Anderson [33] has proposed a classification of error detection checks, some of which can be chosen for the implementation of the module properties mentioned above. The location of the checks can be within the modules or at their outputs, as needed. The checks include replication, timing, reversal, coding, reasonableness, and structural checks.

- The use of **Assertions** [105] that is logic statements inserted at different points in the program that reflects invariant relationships between the variables of the program can also be used for fault tolerance. However it can lead to different problems, since assertions are not transparent to the programmer and their effectiveness largely depends on the nature of the application and on the programmers ability.

- **Control Flow Checking** [106] is to partition the application program in basic blocks (that is, branch-free parts of code). For each block a deterministic signature is computed and faults can be detected by comparing the run-time signature with a precomputed one. In most control-flow checking techniques one of the main problems is to tune the test granularity that should be used.

- **Replication checks** make use of matching components with error detection based on comparison of their outputs. This is applicable to multi-version software fault tolerance.

- **Timing checks** are applicable to systems and modules whose specifications include timing constraints, including deadlines. Based on these constraints, checks can be developed to look for deviations from the acceptable module behavior. Watchdog timers are a type of timing check with general applicability that can be used to monitor for satisfactory behavior and detect "lost or locked out" components.

- **Reversal checks** use the output of a module to compute the corresponding inputs based on the function of the module. An error is detected if the computed inputs do not match the actual inputs. Reversal checks are applicable to modules whose inverse computation is relatively straightforward.

- **Coding checks** use redundancy in the representation of information with fixed relationships between the actual and the redundant information. Error detection [109, 110] is based on checking those relationships before and after operations. Checksums are a type of coding check. Similarly, many techniques developed for hardware (e.g., Hamming, M-out-of-N, cyclic codes) can be used in software, especially in cases where the information is supposed to be merely referenced or transported by a module from one point to another without changing its contents. Many arithmetic operations preserve some particular properties between the actual and redundant information, and can thus enable the use of this type of check to detect errors in their execution.

- **Reasonableness checks** use known semantic properties of data (e.g., range, rate of change, and sequence) to detect errors. These properties can be based on the requirements or the particular design of a module.

- The **Data Structural checks** use known properties of data structures. For example, queues, lists, and trees can be inspected for number of elements in the structure, their links and pointers, and any other particular information that could be articulated. Structural checks could be made more effective by augmenting data structures with redundant structural data like extra pointers, embedded counts of the number of items on a particular structure, and individual identifiers for all the items [34, 35, 36, 37, 38]. Another fault detection tool is run-time checks [15]. These are provided as standard error detection mechanisms in hardware systems (e.g., divide by zero, overflow, underflow). Although they are not application specific, they do represent an effective means of detecting design errors. Error detection strategies can be developed in an ad-hoc fashion or using structured methodologies. Fault trees have been proposed as a design aid in the development of fault detection strategies [39]. Fault trees can be used to identify general classes of failures and conditions that can trigger those failures. Fault trees represent a top-down approach which, although not guaranteeing complete coverage, is very helpful in documenting assumptions, simplifying design reviews, identifying omissions, and allowing the designer to visualize component interactions and their consequences through structured graphical means. Fault trees enable the designer to perform qualitative analysis of the complexity and degree of independence in the error checks of a proposed fault tolerance strategy.

- **The Exception Handling**

The task exception handling is the interruption of normal operation to handle abnormal responses. Exceptions are signaled by the implemented error detection mechanisms as a request for initiation of an appropriate recovery. The design of exception handlers requires that consideration be given to the possible events triggering the exceptions, the effects of those events on the system, and the selection of appropriate mitigating actions [15]. [8] lists three classes of exception triggering events for a software component: interface exceptions, internal local exceptions, and failure exceptions. This knowledge of error containment is essential to the design of effective exception handlers.

- **The Checkpoint and Restart**

For single-design software there are few recovery mechanisms. The most often mentioned is the checkpoint and restart mechanism (e.g., [15]). As mentioned in previous sections, most of the software faults remaining after development are unanticipated, state-dependent faults [76]. This type of fault behaves similarly to transient hardware faults: they appear, do the damage, and then apparently just go away, leaving behind no obvious reason for their activation in the first place [40]. Because of these characteristics, simply restarting a module is usually enough to allow successful completion of its execution [40]. A restart, or backward error recovery has the advantages of being independent of the damage caused by a fault, applicable to unanticipated faults, general enough that it can be used at multiple levels in a system, and conceptually simple [33]. There exist two kinds of restart recovery: static and dynamic. A static restart is based on returning the module to a predetermined state. This can be a direct return to the initial reset state, or to one of a set of possible states, with the selection being made based on the operational situation at the moment the error detection occurred. Dynamic restart uses dynamically created checkpoints that are snapshots of the state at various points during the execution.

- **The Process Pairs**

A process pair uses two identical versions of the software that run on separate processors [15]. The recovery mechanism is checkpoint and restart. Here the processors are labeled as primary and secondary. At first the primary processor is actively processing the input and creating the output while generating checkpoint information that is sent to the backup or secondary processor. Upon error detection, the secondary processor loads the last checkpoint as its starting state and takes over the role of primary processor. As this happens, the faulty processor goes offline and executes diagnostic checks. The main advantage of this recovery technique is that the delivery of services continues uninterrupted after the occurrence of a failure in the system.

- **The Data Diversity**

The last line of defense against design faults is to use "input sequence workarounds". Data diversity can be seen as the automatic implementation of "input sequence workarounds" combined with checkpoint and restart [76]. Again, the rationale for this technique is that faults in deployed software are usually input sequence dependent. Data diversity has the potential of increasing the effectiveness of the checkpoint and restart by using different input re-expressions on each retry [41]. The goal of each retry is to generate output results that are either exactly the same or semantically equivalent in some way. In general, the notion of equivalence is application dependent. [41, 42] presents three basic data diversity models: (i) Input Data Re-Expression, where only the input is changed; (ii) Input Re-Expression with Post-Execution Adjustment, where the output is also processed as necessary to achieve the required output value or format; (iii) Re-Expression via Decomposition and Recombination, where the input is broken down into smaller elements and then recombined after processing to form the desired output. Data diversity is compatible with the Process Pairs technique using different re-expressions of the input in the primary and secondary.

- **Fault Tolerance in Operating Systems**

Any application level software relies on the correct behavior of the operating system. Software fault tolerance can be applied to the design of operating systems [32,76]. However, in general, designing and building operating systems tends to be a rather complex, lengthy and costly endeavor. For safety critical applications it may be necessary to develop custom operating systems through highly structured design processes [31] including highly experienced programmers and advanced verification techniques in order to gain a high degree of confidence on the correctness of the software. Another approach to the development of fault tolerant operating systems for mission critical applications is the use of wrappers on off-the-shelf operating systems to boost their robustness to faults. A problem with the use of off-the-shelf software on dependable systems is that the system developers are not sure if the off-the-shelf components are reliable enough for the application [52]. It is known that the development process for commercial off-the-shelf software does not consider de facto standards for safety or mission critical applications and the available documentation for the design and validation activities tend to be rather weak [53]. A point in favor of using commercial operating systems is that they often include the latest developments in operating system technology. Also, widely deployed commercial operating systems could have fewer bugs overall than custom developed software due to the corrective actions performed in response to bug complaints from the users [54]. Because modifications to the internals of the operating system could increase the risk of introducing design faults, it is preferred to apply techniques that use the software as is. A wrapper is a piece of software put around another component to limit what that component can do without modifying the component's source code [52]. Wrappers monitor the flow of information into and out of the component and try to keep undesirable values from being propagated. In this manner, the wrapper limits the component's input and output spaces. Wrappers have been used as middleware located between the operating system and the application software [55, 56, 57]. The wrappers (called "sentries" in the referenced work) encapsulate operating system services to provide application-transparent fault tolerant functionality and can augment or change the characteristics of the services as seen by the application layer. In this design the sentries provide the mechanism to implement fault tolerance policies that can be dynamically assigned to particular applications based on the individual fault tolerance, cost and performance needs. [53] proposed the use of wrappers at the microkernel level for off-the-shelf operating systems. The wrappers proposed by these researchers aim at verifying consistency constraints at a semantic level by utilizing information beyond what is available at the interface of the wrapped component. Their approach uses abstractions [76] (i.e., models) of the expected component functionality.

3 Assessment of Fault Tolerance by Fault Injection

Software fault injection (SFI) [76] is the process of testing software under anomalous circumstances involving erroneous external inputs or internal state information. The main reason for using software fault injection is to assess the goodness of a design [65]. Basically, SFI tries to measure the degree of confidence that can be placed on the proper delivery of services. Since it is very hard to produce correct software, SFI tries to show what could happen when faults are activated. The collected information can be used to make code less likely to hide faults and also less likely to propagate faults to the outputs either by reworking the existing code or by augmenting its capabilities with additional code as done with wrappers [65]. SFI can be used to target both objectives of the dependability validation process: fault removal and fault forecasting [62]. In the context of fault removal, SFI can be used as part of the testing strategy during the software development process to see if the designed algorithms and mechanisms work as intended. In fault forecasting, SFI is used to assess the fault tolerance robustness of a piece of software (e.g., an off-the-shelf operating system). The use of SFI has two important advantages over the traditional input sequence test cases [59]. First, by actively injecting faults into the software we are in effect accelerating the failure rate and this allows a thorough testing in a controlled environment within a limited time frame. Second, by systematically injecting faults to target particular mechanisms we are able to better understand the behavior of that mechanism including error propagation and output response characteristics. There exist two basic models of software injection: fault injection and error injection. Fault injection simulates software design faults by targeting the code. Here the injection considers the syntax of the software to modify it in various ways with the goal of replacing existing code with new code that is semantically different [65]. This “code mutation” can be performed at the source code level before compilation if the source code is available. The mutation can also be done by modifying the text segment of a program’s object code after compilation. Error injection, called “data-state mutation” in [65], targets the state of the program to simulate fault manifestations. Actual state injection can be performed by modifying the data of a program using any of various available mechanisms: high priority processes that modify lower priority processes with the support of the operating system; debuggers that directly change the program state; message-based mechanisms where one component corrupts the messages received by another component; storage-based mechanisms by using storage (e.g., cache, primary, or secondary memory) manipulation tools; or command-based approaches that change the state by means of the system administration and maintenance interface commands [59]. An important aspect of both types of fault injection is the operational profile of the software [65]. Fault injection is a dynamic-type testing because it must be used in the context of running software following a particular input sequence and internal state profile. A large amount of work has been done in the area of assessing software robustness by many researchers. Examples of reported works include [54, 58, 60, 61, 63, 64].

4 Software and Hardware Fault Tolerance

System fault tolerance is a vast area of knowledge well beyond what can be covered in a single paper. The concepts presented in this section are purposely treated at a high level with details considered only where regarded as appropriate. Readers interested in a more thorough treatment of the concepts of computer system fault tolerance should consult additional reference material [15, 66, 67].

- **Fault Tolerance in Computer System**

Computer fault tolerance is one of the means available to increase dependability of delivered computational services. Dependability is a quality measure encompassing the concepts of reliability, availability, safety, performability, maintainability and testability [68]. (i) Reliability is the probability that a system continues to operate correctly during a particular time interval given that it was operational at the beginning of the interval. (ii) Availability is the probability that a system is operating correctly at a given time instant. (iii) Safety is the probability that the system will perform in a non-hazardous way. A hazard is defined as "a state or condition of a system that, together with other conditions in the environment of the system, will lead inevitably to an accident" [69]. (iv) Performability is the probability that the system performance will be equal to or greater than some particular level at a given instant of time. (v) Maintainability is the probability that a failed system will be returned to operation within a particular time period. Maintainability measures the ease with which a system can be repaired. (vi) Testability is a measure of the ability to characterize a system through testing. Testability includes the ease of test development (i.e., controllability) and effect observation (i.e., observability).

The primary concern for fault tolerant designs is the ability to continue delivery of services in the presence of faults in the system. A fault is an anomalous condition occurring in the system hardware or software. [66,70] presents a general fault classification which is excellent for understanding the types of faults that fault tolerant designs are called upon to handle. A *latent fault* is a fault that is present in the system but has not caused errors; after errors occur, the fault is said to be active. Permanent faults are present in the system until they are removed; transient faults appear and disappear on their own with no explicit intervention from the system. *Symmetric faults* are those perceived identically by all good subsystems; asymmetric faults are perceived differently by the good subsystems. A *random fault* is caused by the environment (e.g., heat, humidity, vibration, etc.) or by component degradation; generic faults are built-in faults accidentally introduced during design or manufacturing of the system. *Benign faults* are detectable by all good subsystems; malicious faults are not directly detectable by all good subsystems. The fault count classification is relative to the modularity of the system. A *single fault* is a fault in a single system module; a group of multiple faults affects more than one module.

The *time classification* is relative to the time granularity. Coincident multiple faults appear during the same time interval; distinct-time faults appear in different time intervals. Independent faults are faults originating from different causes or nature. Common mode faults, in the context of multiple faults, are faults that have the same cause and are present in multiple components.

The main use of fault tolerance in these systems is to provide added value and prevent nuisance faults from affecting the perceived dependability from a user perspective. The design of systems with fault tolerance capabilities to satisfy particular application requirements is a complex process loaded with theoretical and experimental analysis in order to find the most appropriate tradeoffs within the design space. [66] offers a high-level design paradigm extracted from the more detailed description presented in [70]. System properties to be considered include dependability (i.e., reliability, availability, maintainability, etc), performance, failure modes, environmental resilience, weight, cost, volume, power, design effort, and verification effort.

Every fault tolerant design must deal with one or more of the following aspects [33, 71,76]:

- Detection: A basic element of a fault tolerant design is error detection. Error detection [96, 97, 98, 99, 100] is a critical prerequisite for other fault tolerant mechanisms.

- Containment: In order to be able to deal with the large number of possible effects of faults in a complex computer system it is necessary to define confinement boundaries for the propagation of errors. Containment regions are usually arranged hierarchically throughout the modular structure of the system. Each boundary protects the rest of the system from errors occurred within it and enables the designer to count on a certain number of correctly operating components by means of which the system can continue to perform its function.

- Masking: For some applications, the timely flow of information is a critical design issue. In such cases, it is not possible to just stop the information processing to deal with detected errors. Masking is the dynamic correction of errors. In general, masking errors is difficult to perform inline with a complex component. Masking, however, is much simpler when redundant copies of the data in question are available.

- Diagnosis: After an error is detected, the system must assess its health in order to decide how to proceed. If the containment boundaries are highly secure, diagnosis is reduced to just identifying the enclosed components. If the established boundaries are not completely secure, then more involved diagnosis is required to identify which other areas are affected by propagated errors.

- Repair/reconfiguration: In general, systems do not actually try to repair component-level faults in order to continue operating. Because faults are either physical or design-related, repair techniques are based on finding ways to work around faults by either effectively removing from operation the affected components or by rearranging the activity within the system in order to prevent the activation of the faults.

- Recovery and Continued Service: After an error is detected, a system must be returned to proper service by ensuring an error-free state. This usually involves the restoration to a previous or predefined state, or rebuilding the state by means of known-good external information.

Redundancy in computer systems is the use of resources beyond the minimum needed to deliver the specified services. Fault tolerance is achieved through the use of redundancy in the hardware, software, information, or time domain [68,71,73]. In what follows we present some basic concepts of hardware redundancy to achieve hardware fault tolerance. Good examples of information domain redundancy for hardware fault tolerance are error detecting and correcting codes [72]. Time redundancy is the repetition of computations in ways that allow faults to be detected [68, 76].

5 Implemented Structures

Here, we present few examples of fault tolerant architectures that are implemented in various important applications.

Fault-tolerance in *Maintainable Real-Time System* (MARS) [82,83] is based on fail-silent components running in dual active redundancy and on sending each message twice on the two actively redundant real-time busses. MARS is a fault –tolerant distributed real-time architecture for hard real-time application.

CRAK [84, 85, 86] is a kernel module that implements checkpoint / restart for Linux. Checkpoint / restart is an operating system feature that creates a file describing a running process. Checkpoint / restart is a mechanism for fault tolerance. Applications may be checkpointed periodically. Once the application state has been committed to stable storage, the application may be restarted and reconfigured to work around the fault.

Safety critical fault tolerant architectures are used on the flight control computers of the fly-by-wire systems of two types of commercial jet transport aircraft. The first computer is used on the *Boeing 777* airplane [76]. The second computer is used on the AIRBUS A320/A330/A340 series aircraft. The fly-by-wire system of the Boeing 777 airplane departs from old-style mechanical systems that directly connect the pilot's control instruments to the external control surfaces. A fly-by-wire system enables the creation of artificial airplane flight characteristics that allow crew workload alleviation and flight safety enhancement, as well as simplifying maintenance procedures through modularization and automatic periodic self-inspection [77,78, 79, 80, 81]. Software diversity was to be achieved through the use of different programming languages targeting different lane processors. The final and current implementation uses only one programming language with the executable code being generated by three different compilers still targeting dissimilar lane processors. The lane processors are dissimilar because they are the single most complex hardware devices, and thus there is a perceived risk of design faults associated with their use. The requirements for the flight control computer on the Airbus A320/A330/A340 [76] include many of the same considerations as in the B777 fly-by-wire system [87, 88]. The selected architecture, however, is much different. The basic building block is the fail-stop control and monitor module.

Active-stream / Redundancy-stream Simultaneous Multithreading (AR_SMT) [89, 90, 91] exploits several recent microarchitectural trends (e.g., simultaneous multithreading [94, 95], control flow and data flow prediction and hierarchical processors) to provide low-overhead, broad coverage of transient faults and restricted coverage of some permanent faults. Program-level time redundancy is used here. Time redundancy [92, 93, 107, 108] is a fault tolerant technique in which a computation is performed multiple times.

Self-stabilization [115] is an *optimistic* way of looking at system fault tolerance, because it provides a built-in safeguard against transient failures that might corrupt the data in a distributed system. Although the concept was introduced by Dijkstra in 1974 [111], and Lamport [112] showed its relevance to fault tolerance in distributed systems in 1983, serious work only began in the late nineteen-eighties. A good survey of self-stabilizing algorithms can be found in [113]. Herman's bibliography [114] also provides a fairly comprehensive listing of most papers in this field. Because of the size and nature of many ad hoc and geographically distributed systems, communication links are unreliable. The system must therefore be able to adjust when faults occur. But 100% fault tolerance is not warranted. The promise of self-stabilization, as opposed to fault masking, is to recover from failure in a reasonable amount of time and without intervention by any external agency. Since the faults are transient (eventual repair is assumed), it is no longer necessary to assume a bound on the number of failures. A fundamental idea of self-stabilizing algorithms is that the distributed system may be started from an arbitrary global state. After a finite amount of time the system reaches a correct global state, called a *legitimate* or *stable* state. An algorithm is self-stabilizing if (i) for any initial illegitimate state it reaches a legitimate state after a finite number of node moves, and (ii) for any legitimate state and for any move allowed by that state, the next state is a legitimate state. A self-stabilizing system does not guarantee that the system is able to operate properly when a node continuously injects faults in the system (Byzantine fault that generates wrong and random answer) or when communication errors occur so frequently that the new legitimate state cannot be reached. While the system services are unavailable when the self-stabilizing system is in an illegitimate state, the repair of a self-stabilizing system is simple; once the offending equipment is removed or repaired the system provides its service after a reasonable time.

[117] have proposed a *transient fault tolerant design* that takes good advantage of the resource for parallel executions found in a superscalar processor. The design in [117] delivers fault tolerance by carrying out multiple executions of the same instruction in lower performance.

The approaches in [118-140] are all low-cost but effective software fault tolerance tool that do not rely on software design diversity. These approaches are all based on enhanced single-version programming (ESVP) schemes for fault tolerance against operational faults, transient and permanent errors etc. Replicated or transformed code [118,119,125] and data have also been used. [120,121,126,127,129] propose other important single-version fault tolerance approaches to design reliable applications using robust data structure for tolerating erroneous control flow and program hanging. [122,136] propose various low-cost software based fault tolerance approaches for designing microprocessor based commodity applications on inserting NO-Operation (NOP) instructions and run time consistency checking for those extra NO-Operation codes. The approaches in [123,124,128] describe various self-checking and assertion based approaches for designing reliable computing by enhancing basic computing logic. The approaches in [125,130] rely on code and data redundancy for detecting run-time error detection and recovery thereof. [138] proposes a single version software implemented transient fault tolerant approach for designing a reliable application using a multiprocessor with lower overhead on execution time. [131, 132, 133, 134, 135, 137, 139,140] propose various enhanced single-version software implemented transient fault tolerant computing schemes using fault masking with an affordable time redundancy (< 3) and program state verification. Specific knowledge of the application allows the use of a suite of math, logic and heuristic checks on the data, the data processing flow and the results. These techniques also provide more efficient error handling, and system recovery.

The approaches in [118-140] are useful specifically for designing low cost reliable computing applications against operational, transient and permanent errors that might occur during the execution time of applications. ESVP does not aim to tolerate software design bugs. ESVP needs only one reliable machine to execute an application with an enhanced processing logic. It does not rely on multiple versions of software and machines. The designers also feel comfortable while implementing these simple approaches to their applications without any extra cost and hardware.

6 Conclusion

A review on software fault tolerance is presented in this paper. It covers fault tolerant computing schemes that rely on the single-design as well as on the multiple-design. Single version software fault tolerance techniques discussed include system structuring and closure, atomic actions, inline fault detection, exception handling, assertion, and checkpoint and restart. Process pairs exploit the state dependence characteristic of most software faults to allow uninterrupted delivery of services despite the activation of faults. Similarly, data diversity aims at preventing the activation of design faults by trying multiple alternate input sequences. Multiple-version techniques are based on the assumption that software built differently should fail differently and thus, if one of the redundant versions fails, at least one of the others should provide an acceptable output. Because of our present inability to produce error-free software, software fault tolerance is and will continue to be an important consideration in software systems.

The root cause of software design errors is the complexity of the systems. Compounding the problems in building correct software is the difficulty in assessing the correctness of software for highly complex systems. Current research in software engineering focuses on establishing patterns in the software structure and trying to understand the practice of software engineering aiming at better predicting the software dependability. Multiple-version schemes are costlier (O(2.67)) than a single-version software fault tolerance approach because of lower software development cost.

References

[1] L. Spainhower and T.A. Gregg, *IBM S/390 Parallel Enterprise Server G5 Fault Tolerance: a Historical Perspective*, IBM Journal of Research & Development, Vol.43, No. 5/6, 1999.

[2] A. Messer, P. Bernadat, G. Fu, D. Chen, Z. Dimitrijevic, D. Lie, D.D. Mannaru, A. Riska and D. Milojicic, *Susceptibility of Commodity Systems and Software to Memory Soft Errors*, IEEE Transactions on Computers, Vol. 53, No. 12, December 2004, pp. 1557–1568.

[3] H. Kopetz, H. Kantz, G. Grilnsteidl, P. Puschner and J. Reisinger, *Tolerating Transient Faults in MARS*, Proc. of the 20^{th} Symposium on Fault Tolerant Computing, June 1990, UK.

[4] Brian Randell, *System Structure for Software Fault Tolerance*, IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, June 1975, pp. 220–232.

[5] Michael R. Lyu, editor, *Software Fault Tolerance*, John Wiley & Sons, 1995.

[6] Algirdas Avizienis, *The N-Version Approach to Fault-Tolerant Software*, IEEE Transactions on Software Engineering, Vol. SE-11, No. 12, December 1985, pp. 290–300.

[7] Algirdas Avizienis, *The Methodology of N-Version Programming*, in R. Lyu, editor, Software Fault Tolerance, John Wiley & Sons, 1995.

[8] Brian Randell and Jie Xu, *The Evolution of the Recovery Block Concept*, in Software Fault Tolerance, Michael R. Lyu, editor, Wiley, 1995, pp. 1–21.

[9] Brian Randell, et al, editors, *Predictably Dependable Computing Systems*, Springer, 1995.

[10] J.C. Laprie, et al, *Hardware- and Software-Fault Tolerance: Definition and Analysis of Architectural Solutions*, Digest of Papers FTCS-17: The Seventeenth International Symposium on Fault-Tolerant Computing, July 1987, pp. 116–121.

[11] Jean-Claude Laprie, et al, *Definition and Analysis of Hardware- and Software-Fault- Tolerance Architectures*, IEEE Computer, July 1990, pp. 39–51.

[12] J.C. Laprie, et al, *Architectural Issues in Software Fault Tolerance*, in Software Fault Tolerance, Michael R. Lyu, editor, Wiley, 1995, pp. 47–80.

[13] R. Keith Scott, James W. Gault, and David F. McAllister, *Fault-Tolerant Software Reliability Modeling*, IEEE Transactions on Software Engineering, Vol. SE-13, No. 5, May 1987, pp. 582–592.

[14] Jie Xu and Brian Randell, *Software Fault Tolerance: t/(n-1)-VariantProgramming*, IEEE Transactions on Reliability, Vol. 46, No. 1, March 1997, pp. 60–68.

[15] Dhiraj K. Pradhan, *Fault-Tolerant Computer System Design*, Prentice-Hall, Inc., 1996.

[16] Peter Bishop, *Software Fault Tolerance by Design Diversity*, in R. Lyu, editor, Software Fault Tolerance, John Wiley & Sons, 1995.

[17] A. Avizienis, et al, *The UCLA DEDIX System: A Distributed Testbed for Multiple-Version Software*, Digest of Papers: The Fifteenth Annual International Symposium on Fault-Tolerant Computing (FTCS 15), Ann Arbor, Michigan, June 19–21, 1985, pp. 126–134.

[18] Algirdas Avizienis and Jean-Claude Laprie, *Dependable Computing: From Concepts to Design Diversity*, Proceedings of the IEEE, Vol. 74, No. 5, May 1986, pp. 629–638.

[19] Algirdas Avizienis, *In Search of Effective Diversity: A Six-Language Study of Fault-Tolerant Flight Control Software*, Digest of Papers FTCS-18: The Eighteenth International Symposium on Fault-Tolerant Computing, June 27–30, 1988, pp. 15–22.

[20] Algirdas Avizienis, *Software Fault Tolerance*, Information Processing 89, Proceedings of the IFIP 11 th World Computer Congress, 1989, pp. 491–98.

[21] Algirdas Avizienis, *Dependable Computing Depends on Structured Fault Tolerance*, Proceedings of the 1995 6^{th} International Symposium on Software Reliability Engineering, Toulouse, France, 1995, pp. 158–168.

[22] Algirdas Avizienis, *The Methodology of N-Version Programming*, in R. Lyu, editor, Software Fault Tolerance, John Wiley & Sons, 1995.

[23] Algirdas Avizienis, *Toward Systematic Design of Fault-Tolerant Systems*, Computer, April 1997, pp. 51–58.

[24] Thomas C. Bressoud, *TFT: A Software System for Application-Transparent Fault Tolerance*, Digest of Papers: Twenty-Eight Annual International Symposium on Fault-Tolerant Computing, Munich, Germany, June 23–25, 1998, pp. 128–137.

[25] F. Saglietti, *Strategies for the Achievement and Assessment of Software Fault-Tolerance*, IFAC 1990 World Congress, Automatic

Control. Vol. IV, IFAC Symposia Series, Number 4, 1991, pp. 303–308.

[26] J. C. Knight, et al, *A Large Scale Experiment in N-Version Programming*, Digest of Papers FTCS-15: The 15th Annual International Conference on Fault Tolerant Computing, June 1985, pp. 135–139.

[27] J. C. Knight and Nancy G. Leveson, *An Experimental Evaluation of the Assumption of Independence in Multiversion Programming*, IEEE Transactions on Software Engineering, Vol. SE-12, No. 1, January 1986, pp. 96–109.

[28] Dave E. Eckhardt, et al, *An Experimental Evaluation of Software Redundancy as a Strategy for Improving Reliability*, IEEE Transactions on Software Engineering, Vol. 17, No. 7, July 1991, pp. 692–702.

[29] Michael R. Lyu, editor, *Software Fault Tolerance*, John Wiley & Sons, 1995.

[30] Roger S. Pressman, *Software Engineering: A Practitioner's Approach*, The McGraw-Hill Companies, Inc., 1997

[31] *Software Considerations in Airborne Systems and Equipment Certification*, RTCA/DO-178B, RTCA, Inc, 1992.

[32] Peter J. Denning, *Fault Tolerant Operating Systems*, ACM Computing Surveys, Vol. 8, No. 4, December 1976, pp. 359–389.

[33] T. Anderson and P.A. Lee, *Fault Tolerance: Principles and Practice*, Prentice/Hall, 1981.

[34] Russell J. Abbott, *Resourceful Systems for Fault Tolerance, Reliability, and Safety*, ACM Computing Surveys, Vol. 22, No. 1, March 1990, pp. 35–68.

[35] David J. Taylor, et al, *Redundancy in Data Structures: Improving Software Fault Tolerance*, IEEE Transactions on Software Engineering, Vol. SE-6, No. 6, November 1980, pp. 585–594.

[36] David J. Taylor, et al, *Redundancy in Data Structures: Some Theoretical Results*, IEEE Transactions on Software Engineering, Vol. SE-6, No. 6, November 1980, pp. 595–602.

[37] J. P. Black, et al, *Introduction to Robust Data Structures*, Digest of Papers FTCS-10: The Eleventh Annual International Symposium on Fault-Tolerant Computing, October 1–3, 1980, pp. 110–112.

[38] J. P. Black, et al, *A Compendium of Robust Data Structures*, Digest of Papers FTCS-11: The Eleventh Annual International Symposium on Fault-Tolerant Computing, June 24–26, 1981, pp. 129–131.

[39] Herbert Hecht and Myron Hecht, *Fault-Tolerance in Software, in Fault-Tolerant Computer System Design*, Dhiraj K. Pradhan, Prentice Hall, 1996.

[40] Jim Gray, *Why Do Computers Stop and What Can Be Done About It?* Proceedings of the Fifth Symposium On Reliability in Distributed Software and Database Systems, January 13–15, 1986, pp. 3–12.

[41] Paul E. Ammann and John C. Knight, *Data Diversity: An Approach to Software Fault Tolerance*, IEEE Transactions on Computers, Vol. 37, No. 4, April 1988, pp. 418–425.

[42] Victor F. Nicola, *Checkpointing and the Modeling of Program Execution Time*, in Software Fault Tolerance, Michael R. Lyu, Ed, Wiley, 1995, pp. 167–188.

[43] Tom Anderson, *A Structured Mechanism for Diverse Software*, Proceedings of the Fifth Symposium on Reliability in Distributed Software and Database Systems, January 1986, pp. 125–129.

[44] Paul R. Lorczack, et al, *A Theoretical Investigation of Generalized Voters for Redundant Systems*, Digest of Papers FTCS-19: The Nineteenth International Symposium on Fault-Tolerant Computing, 1989, pp. 444–451.

[45] P. R. Croll, et al, *Dependable, Intelligent Voting for Real-Time Control Software*, Engineering Applications of Artificial Intelligence, vol. 8, no. 6, December 1995, pp. 615–623.

[46] Judith Gersting, et al, *A Comparison of Voting Algorithms for N-Version Programming*, Proceedings of the 24^{th} Annual Hawaii International Conference on System Sciences, Volume II, January 1991, pp. 253–262.

[47] J. M. Bass, *Voting in Real-Time Distributed Computer Control Systems*, PhD Thesis, University of Sheffield, October 1995.

[48] R. B. Broen, *New Voters for Redundant Systems*, Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control, March 1975, pp. 41–45.

[49] John P. J. Kelly, et al, *Multi-Version Software Development*, Proceeding of the Fifth IFAC Workshop, Safety of Computer Control Systems, October 1986, pp. 43–49.

[50] Kam Sing Tso and Algirdas Avizienis, *Community Error Recovery in N-Version Software: A Design Study with Experimentation*, Digest of Papers FTCS-17: The Seventeenth International Symposium on Fault-Tolerant Computing, July 6–8, 1987, pp. 127–133.

[51] F. Saglietti, *Software Diversity Metrics: Quantifying Dissimilarity in the Input Partition*, Software Engineering Journal, January 1990, pp. 59–63.

[52] Jeffrey M. Voas, *Certifying Off-the-Shelf Software Components*, IEEE Computer, Vol. 31, June 1998, pp. 53–59.

[53] Frédéric Salles, et al, *MetaKernels and Fault Containment Wrappers*, Digest of Papers: Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, Madison, Wisconsin, June 15–18, 1999, pp. 22–29.

[54] Philip Koopman, et al, *Comparing Operating Systems Using Robustness Benchmarks*, Proceedings of the 1997 16^{th} IEEE Symposium on Reliable Distributed Systems, October 1997, pp. 72–79.

[55] Mark Russinovich, et al, *Application Transparent Fault Management in Fault Tolerant Mach*, Digest of Papers: The Twenty-Third International Symposium on Fault-Tolerant Computing (FTCS-23), Toulouse, France, June 22–24, 1993, pp. 10–19.

[56] Mark Russinovich, et al, *Application Transparent Fault Managemet in Fault Tolerant Mach*, in Foundations of Dependable Computing System Implementation, Gary M. Koob and Clifford G. Lau, editors, Kluwer Academic Publishers, 1994, pp. 215–241.

[57] Mark Russinovich and Zary Segall, *Fault-Tolerance for Off-The-Shelf Applications and Hardware*, Digest of Papers: The Twenty-Fifth International Symposium on Fault-Tolerant Computing, Pasadena, CA, June 27–30, 1995, pp. 67–71.

[58] Jean Arlat, et al, *Fault Injection for Dependability Validation: A Methodology and Some Applications*, IEEE Transactions on Software Engineering, Vol. 16, No. 2, February 1990, pp. 166–182.

[59] Ming-Yee Lai and Steve Y. Wang, *Software Fault Insertion Testing for Fault Tolerance*, in Software Fault Tolerance, Michael R. Lyu, editor, John Wiley & Sons, 1995, pp. 315–333.

[60] Wei-lun Kao, et al, *FINE: A Fault Injection and Monitoring Environment for Tracing the UNIX System Behavior Under Faults*, IEEE Transactions on Software Engineering, Vol. 19, No. 11, November 1993, pp. 1105–1118.

[61] J. C. Fabre, et al, *Assessment of COTS Microkernels by Fault Injection*, Proceedings IFIP DCCA-7, 1999, pp. 19–38.

[62] Dimitri Avresky, et al, *Fault Injection for the Formal Testing of Fault Tolerance*, Digest of Papers of the Twenty-Second In-

ternational Symposium on Fault-Tolerant Computing, Boston, Massachusetts, July 8–10, 1992, pp. 345–354.

[63] Ravishankar K. Iyer and Dong Tang, *Experimental Analysis of Computer System Dependability*, in Fault Tolerant Computer System Design, Dhiraj K. Pradhan, Prentice Hall, 1996, pp. 282–392.

[64] Inhwan Lee and Ravishankar K. Iyer, *Software Dependability in the Tandem GUARDIAN System*, IEEE Transactions on Software Engineering, Vol. 21, No. 5, May 1995, pp. 455–467.

[65] Jeffrey M. Voas and Gary McGraw, *Software Fault Injection: Inoculating Programs Against Errors*, John Wiley & Sons, Inc., 1998.

[66] N. Suri, et al, *Advances in Ultra-dependable Distributed Systems*, IEEE Computer Society Press, 1995.

[67] Brian Randell, et al, editors, *Predictably Dependable Computing Systems*, Springer, 1995.

[68] Barry W. Johnson, *An Introduction to the Design and Analysis of Fault-Tolerant Systems*, in Fault-Tolerant Computer System Design, Dhiraj K. Pradhan, Prentice Hall, Inc., 1996, pp. 1–87.

[69] Nancy G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley, 1995.

[70] Algirdas Avizienis, *A Design Paradigm for Fault Tolerant Systems*, Proceedings of the AIAA/IEEE Digital Avionics Systems Conference (DASC), Washington, D.C., 1987.

[71] Victor P. Nelson, *Fault-Tolerant Computing: Fundamental Concepts*, IEEE Computer, July 1990, pp. 19–25.

[72] Stephen B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, 1995.

[73] Jaynarayan H. Lala and Richard E. Harper, *Architectural Principles for Safety-Critical Real-Time Applications*, Proceedings of the IEEE, Vol. 82, No. 1, January 1994, pp. 25–40.

[74] Timothy C. K. Chou, *Beyond Fault Tolerance*, IEEE Computer, April 1997, pp. 47–49.

[75] Charles B. Weinstock and David P. Gluch, *A Perspective on the State of Research in Fault-Tolerant Systems*, Software Engineering Institute, Special Report CMU/SEI-97-SR-008, June 1997.

[76] Wilfredo Torres-Pomales, NASA Report (No. L-18034) on Software Fault Tolerance, 2000.

[77] Brian D. Aleska and Joseph P. Carter, *Boeing 777 Airplane Information Management System Operational Experience*, AIAA/IEEE Digital Avionics Systems Conference, Vol. II, 1997, pp. 3.1-21 – 3.1-27.

[78] Robert J. Bleeg, *Commercial Jet Transport Fly-By-Wire Architecture Considerations*, AIAA/IEEE 8th Digital Avionics Systems Conference, October 1988, pp. 399–406.

[79] Andy D. Hills and Dr. Nisar A. Mirza, *Fault Tolerant Avionics*, AIAA/IEEE 8th Digital Avionics Systems Conference, October 1988, pp. 407–414.

[80] Gordon McKinzie, *Summing Up the 777's First Year: Is This a Great Airplane, or What?*, Airliner, July – September 1996, pp. 22–25.

[81] Y.C. Yeh, *Triple-Triple Redundant 777 Primary Flight Computer*, Proceedings of the 1996 IEEE Aerospace Applications Conference, Vol. 1, 1996, pp. 293–307.

[82] A. Damm, J. Reisinger, W. Schwabl, and H. Kopetz, *The Real-Time Operating System of MARS*, ACM SIGOPS Operating Systems Review, Vol. 23, No. 3, July 1989, pp. 141–157.

[83] H. Kopetz, A. Damm, Ch. Koza, M. Mulazzani, W. Schwabl, Ch. Senft and R. Zainlinger, *Distributed Fault-Tolerant Real-Time Systems: The MARS Approach*, IEEE Micro, Vol. 9, No. 1, February 1989, pp. 25–40.

[84] Hua Zhong and Jason Nieh, *CRAK: Linux Checkpoint / Restart As a Kernel Module*, Technical Report CUCS-014-01, Department of Computer Science, Columbia University, November 2002.

[85] E. Roman, *A Survey of Checkpoint/ Restart Implementations*, Lawrence Berkeley National Laboratory – Checkpoint Survey Report, July 2002.

[86] J. Duell, P. Hargrove and E. Roman, *Requirements for Linux Checkpoint/ Restart*, Report of the Lawrence Berkeley National Laboratory, 2002.

[87] Dominique Briere and Pascal Traverse, *AIRBUS A320/A330/A340 Electrical Flight Controls: A Family of Fault-Tolerant Systems*, Digest of Papers FTCS-23: The Twenty-Third International Symposium on Fault-Tolerant Computing, June 1993, pp. 616–623.

[88] Pascal Traverse, *Dependability of Digital Computers on Board Airplanes*, Dependable Computing for Critical Applications, Volume 4, A. Avizienis, J.C. Laprie, editors, 1991, pp. 134–152.

[89] E. Rotenberg, *Ar-smt: Coarse-grain time redundancy for high performance general purpose processors*, Univ. of Wisc. Course Project (ECE753), May 1998.

[90] E. Rotenberg, Q. Jacobson, Y. Sazeides and J. Smith, *Trace Processors*, Proc. 30th Intl. Symp. On Microarchitecture, December 1997.

[91] E. Rotenberg, *AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors*, Technical Report of the Univ. of Wisconsin – Madison, 1998.

[92] J.H. Patel and L.Y. Fung, *Concurrent error detection in ALU's by recomputing with shifted operands*, IEEE Transactions on Computers, Vol. C-31, No. 7, July 1982, pp. 589–595.

[93] B.W. Johnson, *Fault-Tolerant Microprocessor–Based Systems*, IEEE Micro, December 1984, pp. 1609–1624.

[94] D. Tullsen, S. Eggers and H. Levy, *Simultaneous Multithreading: Maximizing on-chip parallelism*, Proc. of the 22^{nd} Intl. Symp. On Computer Architecture, June 1995, pp. 392–403.

[95] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo and R. Stamm, *Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor*, Proc. of the 23^{rd} Intl. Symp. On Computer Architecture, May 1996, pp. 191–202.

[96] A. Damm, *The Effectiveness of Software Error-Detection Mechanisms in Real-Time Operating Systems*, Proc of the 16^{th} Intl. Symp. On Fault Tolerant Computing, Vienna, July 1986, pp.171–176.

[97] A. Damm, *Experimental Evaluation of Error-Detection and Self-Checking Coverage of Components of a Distributed Real-Time System*, PhD Thesis, Technisch Naturwissenschaftliche Fakultat, Technische Universitat Wien, Vienna, Austria, 1988.

[98] A. Damm, *Self-Checking Coverage of Components of a Distributed Real-Time System*, Proc. of the 4^{th} Intl. Conf. On Fault-Tolerant Computing Systems, Germany, 1989, pp. 308–319.

[99] T. Sato and I.Arita, *Tolerating Transient Faults in Microprocessors*, Proc. of the 13^{th} Symposium on Parallel Processing, 2001, Japan.

[100] J. Reisinger, *Failure Modes and Failure Characteristics of a TDMA driven Ethernet*, Research Report 8/89, Institut fur Technische Informatik, Technische Universtat Wien, Vienna, Austria, 1989.

[101] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger and L. Alvisi, *Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic*, Proc. of the Intl. Conf. On Dependable Systems and Networks, Maryland, June 2002, pp. 389–398.

[102] D. Milojicic, A. Messer, J. Shau, G. Fu, P. Alto and A. Munoz, *Increasing Relevance of Memory Hardware Errors: a Case for recoverable Programming Models*, Proc. of the ACM SIGOPS European Workshop, Denmark, Sept. 2000, pp. 97–102.

[103] M. Rebaudengo, M.S. Reorda, M. Torchiano and M. Violante, *Soft-Error Detection Through Software Fault Tolerance Techniques*, Proc. of the IEEE Intl. Symp. On Defect and Fault Tolerance in VLSI Systems, New Mexico, Nov. 1999, pp. 210–218.

[104] C. da Lu and D.A. Reed, *Assessing Fault Sensitivity in MPI Applications*, In Supercomputing, Pittsburg, Nov. 2004, pp. 37.

[105] M.Z. Rela, H. Madeira and J.G. Silva, *Experimental Evaluation of the Fail Silent Behavior in Programs with Consistency Checks*, Proc. of the Intl. Symp. On Fault Tolerant Computing, Japan, June 1996, pp. 394–403.

[106] S. Yau and F. Chen, *An Approach to Concurrent Control Flow Checking*, IEEE Transactions on Software Engineering, Vol. 6, No. 2, March 1980, pp. 126–137.

[107] Y. Tamir and E. Gafni, *A Sofware-Based Hardware Fault Tolerance Scheme for Multicomputers*, Proc. of the Intl. Conf. on Parallel Processing, Illinois, August 1987, pp.117–120.

[108] Hoang Pham (Ed), *Fault- Tolerant Software Systems*, IEEE Computer Society Press, 1992.

[109] Kuang-Hua Huang and Jacob A. Abraham, *Algorithm-Based Fault Tolerancefor Matrix Operations*, IEEE Transactions on Computers, Vol. C-33, No. 6, June 1986, pp. 518–528.

[110] Gam D. Nguyen, *Error- Detection Codes: Algorithms and Fast Implementation*, IEEE Transanctions on Computers, Vol. 54, No.1, January 2005, pp. 1–11.

[111] E. W. Dijkstra, *Self-stabilizing systems in spite of distributed control*, Communications of the ACM, Vol.17, No.11, November 1974, pp.643–644.

[112] L. Lamport, *Solved problems, unsolved problems, and non-problems in concurrency*, In Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing, 1984, pages 1–11.

[113] M. Schneider, *Self-stabilization*, ACM Computing Surveys, Vol.25, No.1, March 1993, pp.45–67.

[114] T. Herman, *A comprehensive bibliograph on self-stabilization, a working paper*, Chicago J. Theoretical Comput. Sci., http://www.cs.uiowa.edu/ftp/selfstab/bibliography.

[115] W. Goddard, S.T. Hedetmiemi, D.P. Jacobs and P.K. Srimani, *Self-Stabilizing Distributed Algorithm for Strong Matching in a System Graph*, Proc. of the High Performance Computing, Hyderabad, 2003.

[116] P. Jalote, *Fault Tolerance in Distributed Systems*, Prentice Hall, 1994.

[117] J. Ray, J.C. Hoe and B. Faisafi, *Dual Use of Superscalar Datapath for Transient-Fault Detection and Recovery*, IEEE Micro, 2001.

[118] Goutam Kumar Saha, *Algorithm Based EFT Errors Detection in Matrix Arrays*, Internatinal Journal System Analysis Modelling Simulation, Gordon and Breach, Vol.36, No. 1, USA, 1999, pp.117–135.

[119] G.K. Saha, *Fault Tolerant Computing: A Self-Detection & Recovery Technique*, Internatinal Journal Computers & Electrical Engineering, Elsevier Science Pub., UK, Vol.24, No.5, 1998.

[120] Goutam K Saha, *Transient- Fault Tolerant Processing in a RF Application*, Internatinal Journal System Analysis Modelling Simulation, Gordon and Breach, USA, Vol.38, 2000, pp.81–93.

[121] Goutam K Saha, *EMI Control by Software for a RF Communication System*, in Book: Electromagnetic Environments and Consequences, Part II, Edited by D.J. Serafin, EUROEM, France, 1995, pp.1870–1875.

[122] Goutam Kumar Saha, *A Software Tool for Fault Tolerance*, accepted & in press, Internatinal Journal – Information Science & Engineering, 2005.

[123] Goutam K Saha, *Algorithm Based Fault Tolerant Computing for a Scientific Application*, Internatinal Journal System Analysis Modelling Simulation, Gordon and Breach, USA, Vol.34, No.4, 1999, pp.509–523.

[124] Goutam K Saha, *EMP- Fault Tolerant Computing: A New Approach*, Internatinal Journal of Microelectronic Systems Integration, Plenum Publishing Corporation, USA, Vol.5, No.3, 1997, pp.183–193.

[125] G.K. Saha, *Designing an EMI Immune Software for Microprocessor Based Traffic Control System*, Proc. 11^{th} IEEE Internatinal Symposium EMC'95, Zurich, 1995, pp.401–404.

[126] Goutam Kumar Saha, *Fault Tolerant Processing Technique for a Temperature Measurement System*, in Desk Book: Industrial Measurements & Automation, TIMA'96- American Society of Instrumentation, HCK Publication, Madras, pp.56–58.

[127] G.K. Saha, *Virtual N-Versions Programming for Fault Tolerant Computing*, Internatinal Journal Computers & Electrical Engineering, Elsevier Science Pub., UK, Vol.24, No.4, 1999.

[128] G.K. Saha, *Using Software to Control ESD/EMP in Microprocessor-Based System*, RF Design (EMC Test & Design), Argus Inc., USA, November 1995, pp. 8-11.

[129] G.K. Saha, *EMI Protection by Software for a Microcomputer Based Process Controller*, Proc. IEEE sponsored Symp. ISEMC'94, SP Brazil, 1994.

[130] Goutam Kumar Saha, *Transient Software Fault Tolerance Through Recovery*, ACM Ubiquity, ACM Press, Vol. 4, No. 29, USA, 2003.

[131] Goutam Kumar Saha, *Fault Tolerance in Distributed System*, CSI HardCopy, Vol. 40, December 2003, Kolkata, Computer Society of India.

[132] Goutam Kumar Saha, *Single-Version Software Fault Tolerance in Process Control System*, Proc of the Modelling & Simulation, MS'2004, AMSE Press, France, 2004.

[133] Goutam Kumar Saha, *Beyond the Conventional Techniques of Software Fault Tolerance*, ACM Ubiquity, ACM Press, Vol. 4, No. 47, USA, 2004.

[134] Goutam Kumar Saha, *Fault Management in Mobile Computing*, ACM Ubiquity, ACM Press, Vol. 4, No. 32, USA, 2003.

[135] Goutam Kumar Saha, *Software Fault Tolerance in Industrial Automation*, Journal Industrial Automation, IED, Mumbai, April 2004.

[136] Goutam Kumar Saha, *A Software Fix Towards Fault Tolerant Computing*, ACM Ubiquity, ACM Press, Vol. 6, No. 16, USA, May 2005.

[137] Goutam Kumar Saha, *A Technique of Designing in Application System with Fault Tolerance*, Journal of the AMSE, France, December 2004.

[138] Goutam Kumar Saha, *Fault Tolerance Application Using a Multiprocessor*, to appear in IEEE Potentials, 2005.

[139] Goutam Kumar Saha, *Transient Software Fault Tolerance Using Single Version Algorithm*, accepted paper, ACM Ubiquity, ACM Press, 2005.

[140] Goutam Kumar Saha, *Software Implemented Fault Tolerance – The ESVP Approach*, to appear in IEEE Potentials, 2005.

G.K. Saha,

Received July 12, 2005

Centre for Development of Advanced Computing,
Kolkata
Mailing Address: CA- 2 / 4B, Baguitai, Deshbandhu Nagar,
Kolkata-700059, INDIA
E–mail: *gksaha@rediffmail.com*

Interfaces to symbolic computation systems: reconsidering experience of Bergman

Svetlana Cojocaru Ludmila Malahova
Alexandru Colesnicov

Abstract

The article is based on experience of implementation of computer algebra system Bergman and on analysis of other symbolic computation systems. It is noted that many symbolic computation systems meet similar interface problems. Necessity of strict separation of calculation engine and interface shell, functions of these components, and requirements to them are motivated. The described approach will be used at future development of Bergman.

1 Introduction

Bergman [8] is a computer algebra system for symbolic calculations in non-commutative and commutative algebra. It calculates Gröbner basis and related information. Bergman is written in Lisp, and the users were to be communicate with the system through underlying Lisp console. This was found unsuitable for most users, and a graphical shell was developed in Java. The system and its interface shell were described elsewhere [1, 4–7].

We present below some inferences from our experience with Bergman and its shell. The article is organized as follows:

- General problems of interfaces to symbolic calculation systems are discussed in Sec. 2. We refer to several existing systems to illustrate our observations.

- Some aspects of Bergman shell implementation and used techniques are discussed in Sec. 3. We permitted here some technical details.
- We try to describe and motivate desired features of interface to a symbolic calculation system in Sec. 4.

2 General problem of interfaces to symbolic calculation systems

We suppose the existence of a program executing symbolic computations (a symbolic computation engine, or, simply, an engine). Symbolic computations are widely used in many areas, including pure and applied mathematics, theoretical physics, etc. Multitude of solved problems makes investigators to create specialized engines for symbolic computations in the cases when use of general purpose systems is inefficient, or the necessary functionality is not implemented even in commercial systems. As a rule, the creator of such system has not enough time, resources, and qualification to develop the interface for it. It is not unusual that rich mathematical ideas implemented in an engine are enveloped in poorly designed interface. The absence of the user-friendly standard interface do not permit the extensive usage of such system because of requiring special knowledge and skills, e.g., in programming, to use it. Another problem of symbolic computation engines is multitude of data formats and the implied difficulty in communication between different engines.

We will divide all functionality of a symbolic computation system (SCS) in two parts: the engine features (the computations that it can execute) and the interface features. It is obvious that the latter are external relative to the former and are almost independent of them.

Development of interfaces for SCS was and remains an object of long-time investigations [2,3].

The problem of SCS interface has the following aspects:

1. Interaction with text editors;
2. Graphics;

3. Interaction with numerical calculation systems;
4. Interaction between different symbolic computation systems (including interaction through computer networks);
5. Testing support;
6. Interaction with end users.

Investigations show that SCS interface development should solve the following problems:

- 2-D presentation of mathematical expressions,
- Editing of mathematical expressions that includes sub-expression manipulation,
- Windows that model sheets of paper and combine texts, formulas, and graphics,
- Processing and presentation of long expressions,
- Simultaneous use of several SCS, which implies the necessity to solve problems of data conversion, configuration management, and communication protocols,
- Satisfaction of special needs for teaching systems, (in particular, the possibility to show intermediate results and explications of processes applied to obtain them; elaboration of electronic manuals, and especially interactive ones),
- Interface extensibility providing additions of new menus, new fragments of on-line documentation, etc.,
- Guiding of the user during the whole period of his/her problem solving,
- The system should be self-explanatory; its operational mode should be understandable directly from the experience of interaction with the system,
- Control over problem formulation correctness and over information necessary to solve it.

The primary scope of an interface is creation of a comfortable environment for a mathematician or another specialist that uses mathematical apparatus. It would be preferable for these users to input data and to obtain mathematical results in their natural 2-dimensional form. The linear form of input can be used also as the input is faster but it imposes additional conventions to enter powers, indices, fractions, etc., or uses additional characters. It is necessary also to provide possibilities to edit expressions, integrate them with a usual text, and obtain results in a form suitable for publication of an article (e.g., LaTeX) or in Internet (e.g., MathML).

The syntactic check of entered mathematical expressions and the spelling check of accompanying text would be also desired features.

The following three categories of SCS can be found analyzing SCS interfaces:

1. Systems or packages that do not have a special interface,
2. Interfaces based on a (specialized) programming language,
3. Graphical interfaces.

This division is not strict: e.g., most systems with graphical interface possess their own programming language also.

There are many systems that have command line interface only, e.g., Singular [22], Bergman, and Yacas [24].

Absence of graphical interfaces is compensated partially by integration in an existing editor like Emacs or its derivatives (e.g., such are Macaulay [13] and Singular), in Scientific Workplace [21], Scientific Word, or Scientific Notebook (e.g., Maple [14] and MuPAD [17]). MuPAD has the interface to Java but their approach is opposite to ours: MuPAD itself is regarding as the shell, and Java programs are treated as applets or plug-ins expanding MuPAD itself. Services provided from these editors may be not too sophisticated but create much more comfortable environment than operating in ASCII from the command line.

We can mention also specialized editors developed to serve as interfaces to SCS. One such editor is TeXmacs by Joris van der Hoeven [23]. It combines elements of TeX and Emacs and was successfully applied for Macaulay 2, Reduce [20], MuPAD, Maxima [16]. Another front-end product with graphical interface is FrontMan [11]. It offers a small but useful set of possibilities (transparent SSH sessions, syntactic coloring of input information and results, export of sessions in HTML format, integrated document visualization through a Web browser, multiple simultaneous sessions). An important fact is that these editors permit creation of users own style. It is a kind of personalization created by a user.

Most of features mentioned above can be found in systems with graphical interfaces. Examples of such systems are Derive [10], Mathematica [15], etc. In general, most of these systems provide:

- Visualization of mathematical formulae in 2-dimensional format,
- Sub-expression manipulation,
- Separate windows for data input and results output,
- Separate windows for graphical operations,
- Export of results in a printable format (RTF, PDF, LaTeX, etc.)
- Comfortable navigation with on-line help,
- Integration with an existing or specially developed editor that facilitates editing of mathematical texts,
- Demonstration of intermediate steps to explain processes of expression transformation.

In addition, so-called "notebooks" support operations over text, mathematical formulae, and graphics. Most of them can be adapted to user's preferences individualizing menus and toolbars, and assigning hot keys to actions.

Communication between different systems is to be supported by use of specially developed unified formats for mathematical formulae, like OpenMath [18] or OpenXM [19].

3 Inferences from development of Bergman and its shell

Here we analyze and motivate several solutions taken by us in Bergman implementations. We selected only less trivial aspects; big part of design (e.g., main menu structure, toolbar, sliders, etc.) is usual for graphical shells of any kind.

3.1 Implementation language

Bergman was designed under commercial PSL (Portable Standard Lisp). It works also under Reduce [20] that is in its turn based on PSL. For better dissemination of Bergman, we ported it to free CLISP [9] implementation of ANSI Common Lisp. We needed therefore for Bergman shell a programming system that was free and as portable as CLISP.

We selected Java [12] by obvious reasons. We developed the Bergman shell in Intel PC under Linux and Windows, and in Sun Sparc under Solaris, and tested CLISP port of Bergman and the shell in all these machines.

The selection of Java should be classified as very successful. We found only several small problems in porting the shell:

1. Under Windows, the execution thread with Bergman do not terminates simultaneously with calculations as under Unix. It seems to happen due to implementation of program run in terminal window: at first, the terminal itself is started in the thread, and then CLISP Bergman is started under the terminal. The termination of Bergman does not mean termination of terminal. We catch the terminal output and stop the process after the corresponding message. This solution seems to be not very perfect, and we need some additional tuning of running Bergman from the shell under Windows.
2. The screen font sizes and proportions are different in all three systems: Linux, Solaris, and Windows. The design of screen forms is to be made taking this into account. We were to redesign several forms as we tested the shell under Solaris at the first time. The problem is solved by careful form design and obligatory testing under all available systems.

3.2 Adaptable user interface: sessions and environments

A session is a set of parameters that fully defines the problem to be solved. Session is implemented as a directory where all data are saved, Bergman input files are generated, and Bergman output files are produced. Sessions serve to return to previous Bergman calculations, modify them, and experiment with them.

Informally, sessions give to a mathematician the possibility to use the previous experience of Bergman's users (own or others) and to save the current setup for the future calculations.

With sessions, it is possible:

- select a session and load its data to panels, i.e., switch to the saved problem;
- create a new session;
- save data in the selected session;
- save data in a different session (save as. . .);
- delete a session.

One can see also the session directory, comment, and statistics of its usage.

The session mechanism is usual for dialog shells and IDEs (Integrated Desktop Environments). For Bergman, we found it necessary to generalize the notion of session. We called the new feature "environments".

An environment is a partial set of data common for several sessions. It corresponds to the group of mathematical problems the user investigates during different sessions. E.g., after the installation the environments directory contains an environment called "commutative" that fixes a single parameter, the commutativity.

All new sessions are created using the current default environment. Inversely, when a new environment is created, it is based on the parameters of the current session. To save a session as an environment, the user selects parameters that are to be fixed, and drops other parameters.

3.3 Dialog data input

Main data for Gröbner basis calculations are list of variables and list of polynomials. They are entered in usual text areas. Other data may be represented as switches (flags) and selections. There are approx. 30 input fields the user should set.

It was a serious problem to design these fields in comfortable and reviewable manner. The first idea was to enter the session parameters step by step (in wizard mode) but it was rejected. One of reasons for it was the absence of suitable wizard implementation in Java, but the main reason was that such mode is quite tedious. Next, we tried to position input fields in five tabs but this was badly reviewable.

The current solution permitted us to concentrate all information in a relatively small panel. We use drop-down menus and labels that show the current selection. The drop-down menu is activated when the corresponding label is clicked. Let us revise an example to make the situation more clear.

There is a possibility to select the field for polynomial coefficients. The field characteristic can be 0, 2, or any odd prime number. In characteristic 0, we use integers instead of rational numbers by reducing all coefficients to a common denominator. The drop-down menu for coefficient field contains the following items:

- Machine integers (16-bit)
- Machine integers (32-bit)
- Machine integers (64-bit)
- Lisp integers (arbitrary precision)
- Characteristic 2
- Odd prime characteristic

Suppose the user selects odd prime characteristic. Then an additional small dialog window appears and the user is asked to enter an odd prime. The entered number is checked. Then the label shows: "Odd prime characteristic = p", where p is the entered odd prime. Here we do not need the additional input field on the form for the odd prime number, and all information is always visible.

We use menus and not combo boxes because of pure technical reason. We can assign a program action to each menu item that is not the case with combo boxes.Using actions, we can treat mutually dependent parameters in a natural manner.

The equivalent set of radio buttons plus the input field (as in our previous implementation) occupies, obviously, the bigger form surface.

3.4 Expanded consoles

When we start a calculation under the shell, Bergman run in a console window we programmed for the case. Except of this, we have a possibility to start a console Bergman session. We found that the console features should be expanded. E.g., the calculation console has additional buttons “Stop” and “Continue” and its input and output are caught and come from/to files, but the session console has the feature of keyboard input imitation from an arbitrary file selected by the user.

3.5 Internal Bergman hooks

We inserted into Lisp coded Bergman modules so-called “hooks” where we call external programs supplied by the shell. “Stop” button can not stop calculations in an arbitrary place; the process continues up to some suitable point where the hook is inserted. The hook checks if the stop button was pressed, and stops calculation, with the possibility to continue it from the same point. There are many possible uses of such hooks, e.g., progress indicator that moves forward in some point of calculation.

4 Conclusion: desired features of interface to symbolic calculation system

We can conclude from all aforementioned that SCS interfaces have a lot of features and functions in common.

Discussion in Sec. 2 and 3 shows that all discussed problems can be reviewed as general problems of graphical shells to SCSs. We can look at Bergman as at the **calculation engine** which is called from the dialog shell. Inversely, we can look from the shell side and treat Bergman as the **calculation plug-in** for the shell.

The separation of calculation engine and interface shell means that a system can be developed for semi-automatic generation of interface shell for a SCS. Such system should contain ready made adaptable interface modules in Java and an interface generator.

In implementation of such system, the following objectives must be reached:

- To implement selected features of SCS interface at the generalized level permitting automated production of interface, taking into account specifics of this area. To implement cross-process interaction with a symbolic computation engine.

- To elaborate, inside the interface, tools for smart user personalization and intellectual adaptation to his/her preferences.
- To elaborate an interface generator that will produce interface for a target symbolic computation engine.
- To elaborate programs and filters for interaction with other existing SCS.
- To elaborate and/or adapt and integrate tools for auxiliary development tasks, e.g., help and documentation tools, extensibility support, testing support, etc.
- To elaborate and/or adapt and integrate tools for visual presentation of data and results.
- To produce, as the result, Java packages to be used in interface generation, and Java application(s) for interface development.

Some preliminary research is necessary:

- Classification and technological description of features and functions specific to interfaces for SCS;
- Clarification and classification of cross-process interactions between interface modules and symbolic computation engines;
- Generalization of selected features to the level permitting automated interface generation;
- Description of adaptive behavior of interface elements, and techniques to implement it;
- As the result, the necessary scientific basis will be created to automated development of interfaces for SCS.

Interaction between different systems can be provided also.

SCS is a very useful tool that simplifies formula manipulation and handling of mathematical models for engineering applications, for mathematical research, for education and for many other areas. Our approach can be applied in all these cases and these areas will gain time and efforts for interface development. Universality of the proposed solution will be guaranteed if we will use Java as the technology for its implementation. The developed packages and applications will permit investigators in different areas to concentrate efforts on symbolic computation engines and to use ready-made interface solutions.

References

[1] A. Colesnicov. *Implementation and usage of the Bergman package shell.* / Computer Science Journal of Moldova, **vol. 4,** nr. 2 (11), 1996, pp. 260–276.

[2] N. Kajler, N. Soiffer. *A Survey of User Interfaces for Computer Algebra Systems.* / Journal of Symbolic Computation, **vol. 25,** issue 2, February 1998, pp. 127-159.

[3] N. Kajler (ed.). *Computer-Human Interaction in Symbolic Computation.* - Springer-Verlag: Wien, 1998. - ISBN 3–211–82843–5.

[4] J. Backelin, S. Cojocaru, V. Ufnarovski. *BERGMAN.* In: Computer Algebra Handbook. J. Grabmeier, E. Kaltofen, V. Weispfenning (eds.). – Springer-Verlag: 2003, pp. 349–352.

[5] J. Backelin, S. Cojocaru, V. Ufnarovski. *The Computer Algebra Project Bergman: Current State.* In: "Commutative algebra, Singularities and Computer Algebra", eds. J. Herzog and V. Vuletescu, 2003, pp. 75–101. – Series II. Mathematics, Physics and Chemistry. **Vol. 115,** Kluwer Academic Publishers.

[6] J. Backelin, S. Cojocaru, A. Colesnicov, L. Malahova, V. Ufnarovski. *Problems in interaction with the Computer Algebra*

System Bergman. In: "Computational Commutative and Non-Commutative Algebraic Geometry", **vol. 196,** NATO Science Series: Computer & Systems Sciences. S. Cojocaru et al. (eds.). – IOS Press, 2005, pp. 185–198.

[7] J. Backelin, S. Cojocaru, V. Ufnarovski. *Mathematical Computations Using Bergman.* - Lund University, Centre for Mathematical Science, ISBN 91–631–7203–8, 2005, 206 p.

Web references

[8] Bergman: *http://www.math.su.se/bergman/*

[9] CLISP: *http://clisp.sourceforge.net/*

[10] Derive: *http://www.chartwellyorke.com/derive.html*

[11] FrontMan:
http://rpmfind.net/linux/RPM/sourceforge/r/rp/rpmsforsuse/frontman-0.3.4-1.i386.html
http://www.eleceng.ohio-state.edu/ ravi/kde/frontman.html

[12] Java: *http://java.sun.com/*

[13] Macaulay: *http://www.math.uiuc.edu/Macaulay2/*

[14] Maple: *http://www.maplesoft.com/*

[15] Mathematica:
http://www.wolfram.com/products/mathematica/index.html

[16] Maxima: *http://maxima.sourceforge.net/*

[17] MuPAD: *http://www.mupad.de/*

[18] OpenMath: *http://www.openmath.org/*

[19] OpenXM: *http://www.math.kobe-u.ac.jp/OpenXM/*

[20] Reduce: *http://www.reduce-algebra.com/*

[21] Scientific Workplace, Scientific Word, Scientific Notebook: *http://www.mackichan.com/*

[22] Singular: *http://www.singular.uni-kl.de/*

[23] TEXmacs:
http://www.math.upsud.fr/ anh/TeXmacs/TeXmacs.html

[24] Yacas: *http://yacas.sourceforge.net/*

S. Cojocaru, L. Malahova, A. Colesnicov,
Received October 3, 2005

Institute of Mathematics and Computer Science,
5 Academiei str.
Chişinău, MD−2028, Moldova.
E–mail: *sveta@math.md*, *mal@math.md*, *kae@math.md*