INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Coevolutionary Game-Theoretic Multi-Agent Systems: the Application to Mapping and Scheduling Problems

Franciszek Seredynski*

TR–96-045
October 1996

Abstract

Multi-agent systems based on iterated, noncooperative N-person games with limited interaction are considered. Each player in the game has a payoff function and a set of actions. While each player acts to maximise his payoff, we are interested in the global behavior of the team of players, measured by the average payoff received by the team. To evolve a global behavior in the system, we propose two coevolutionary schemes with evaluation only local fitness functions. The first scheme we call loosely coupled genetic algorithms, and the second one loosely coupled classifier systems. We present simulation results which indicate that the global behavior in both systems evolves, and is achieved only by a local cooperation between players acting without global information about the system. The models of coevolutionary multi-agent systems are applied to develop parallel and distributed algorithms of dynamic mapping and scheduling tasks in parallel computers.

*Institute of Computer Science, Polish Academy of Sciences, Ordona 21, 01-237 Warsaw, Poland, email: sered@ipipan.waw.pl

1 Introduction

One of the widely accepted assumptions concerning the general model of real life systems in business, politics, engineering or the military is considering them as a collection of individuals acting selfishly and interacting to fulfil their own goals. The immediate consequence of this assumption is conflict between individuals and the need for some cooperation to resolve the conflict. For this reason, game theory (Ordeshook 1986, Wang & Parlar 1989) modelling conflict and cooperation has been widely accepted as a useful tool to study the behavior of complex systems.

A game-theoretic model which has received much attention of late is the prisoner's dilemma (Rapoport 1966, Axelrod 1987, Lomborg 1992, Fogel 1993). This two-player, non cooperative game has been explored to yield insight into the conditions of cooperating and defecting between players. Evolutionary computation methods using genetic algorithms (Axelrod 1987) or finite state machines (Fogel 1991) have been successfully used to discover TIT FOR TAT strategy or better strategies of behavior. One of the distinctive features of the prisoner's dilemma is a focus on a player's individual goal. Cooperation in this model is desired and observed, but it is not the ultimate aim of a player.

In the area of Distributed Artificial Intelligence (DAI), game-theoretic models are also the subject of current research (Genesereth at al. 1988, Levy & Rosenschein 1992, Lomborg 1992). Agents of a multi-agent system in such models are considered as players working towards their own selfish goals and taking part in an N-person game. In many DAI applications, (e.g. in engineering) a global behavior of the multi-agent system is expected, rather than simply a fulfilment of players' individual goals. Competing players should act in such systems as a decision group choosing their actions to realise a global goal. One of the problems which must be addressed here is the problem of incorporating the global goals of the multi-agent system into the local interests of all agents. The obvious conflict between individual and global goals is not solved, according to our knowledge, on the grounds of game theory literature concerning social, political or economic phenomena. Such conflict has a deep substantiation in the psychological nature of human beings as well as society. It also acts to limit applications of game-theoretic models in engineering. Those models are replaced consciously due to the aforementioned difficulties by either other formal bases e.g. the social choice theory (Wong 1993), or some new mechanisms e.g. coordinated balancing (Kuwabara & Ishida 1992) are developed.

The paper is application-driven. We are interested in models of N-person games and their potential application in selected problems in the area of parallel and distributed computing (Fox at al. 1988). Mapping (Fox & Furmanski 1988, Moon & Sklansky 1990) or scheduling (Błażewicz at al. 1994, El-Rewini at al. 1994) parallel programs and communication in massively parallel and distributed computers are one of the central questions to be solved to efficiently use their computational power. One of the possible solutions of e.g. the mapping problem is a coordinated migration of program modules in a parallel and distributed system (Seredynski 1994a). A collection of program modules can be considered as some society whose members-players coordinate in some distributed manner directions of their migration to minimise some global performance criterion.

The paper is organized as follows. The following section presents the model of noncooperative N-person games with limited interaction. We discuss the model from the position of

the collective behavior (Tsetlin 1973, Varshavsky 1972, Barto & Anandan 1985) of players taking part in a game, and the conditions of local cooperation between players, providing a maximal payoff for the team of players in a static game. In Section 3, we propose two parallel and distributed genetic algorithm-based schemes to implement the model of dynamic N-person games. These are loosely coupled genetic algorithms and loosely coupled classifier systems. Section 4 presents an application of developed multi-agent models to problems of dynamic mapping and scheduling tasks in parallel computers. The last section contains our conclusions.

2 N-Person Games with Limited Interaction

2.1 Prisoner's Dilemma Background

The typical two person prisoner's dilemma is described by a matrix shown in Table 1 (Axelrod 1987, Fogel 1993).

Table 1: The payoff function used in the prisoner's dilemma

		player B: **C**	player B: **D**
player A	**C**	$\gamma_1 = 3,\ \gamma_1 = 3$	$\gamma_2 = 0,\ \gamma_3 = 5$
player A	**D**	$\gamma_3 = 5,\ \gamma_2 = 0$	$\gamma_4 = 1,\ \gamma_4 = 1$

It is assumed that each player has two alternative actions: **C** - cooperate, and **D** - defect. The game is conducted as a sequence of trials. During each trial, both players must choose independently one of their actions. After each trial, the players receive a payoff whose value is defined by one of four situations: (**C**,**C**) - player A and player B cooperate and both receive the payoff $\gamma_1 = 3$; (**D**,**D**) - both players defect and receive the payoff $\gamma_4 = 1$; (**C**,**D**) - player A cooperates while player B defects and the players obtain respectively the payoff $\gamma_2 = 0$ and $\gamma_3 = 5$; (**D**,**C**) - player A defects and player B cooperates, and they obtain respectively the payoff $\gamma_3 = 5$ and $\gamma_2 = 0$. The values of the payoff defining the game are the subject of some constrains (Rapopport 1966, Fogel 1993).

In our model of N-person games described below we use the basic notation from the prisoner's dilemma and the basic structure of the payoff function. However, our approach to study the game differs from the prisoner's dilemma in stressing the necessity to realize the global goal of the system.

2.2 Homogeneous Games with Limited Interaction

We consider a finite game represented by a set $\boldsymbol{N}$ of N players, $\boldsymbol{N} = \{0, 1, \ldots, N-1\}$; set S_k of actions for each player $k \in \boldsymbol{N}$; and a payoff function $u_k(s_k, s_{k1}, s_{k2}, \ldots, s_{kn_k})$ which depends only on the actions of a limited number of players: on its own action s_k and the actions of its n_k neighbors in the game. Such a model, termed a game with

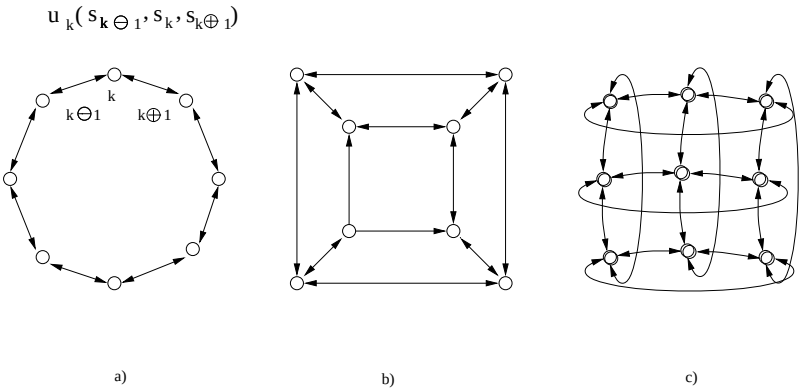

Figure 1: Interaction graph of a game on a) a ring, b) a cube, c) a torus 3×3

limited interaction, had been primarily considered from positions of learning automata games (Tsetlin 1973, Varshavsky 1972, Seredynski 1990).

The game with limited interaction can be represented by an oriented graph $G =< V, E >$ called an interaction graph. V is the set of nodes corresponding to the set of players while set E represents the pattern of interaction between players: arcs incoming to the k-th node define players whose actions influence the payoff of player k, and arcs outgoing from the k-th node define players whose payoff depends on actions of player k.

In the paper we consider a class of games with limited interaction characterized by a regular interaction graph ($n_k = r$, where r is the degree of the interaction graph) and called homogeneous games. In such games, the payoff function is the same for all players. Figure 1 shows the interaction graphs of three homogeneous games.

The simplest homogeneous game with limited interaction is a game on a ring (Figure 1a). For the game on a ring, which will be the subject of our study in the paper, we can simplify notation of the payoff function to read:

$$u_k(s_k, s_{k1}, s_{k2}, \ldots, s_{kn_k}) = u_k(s_{k\ominus 1}, s_k, s_{k\oplus 1}), \tag{1}$$

where $\ominus$ and $\oplus$ denotes subtraction and addition modulo N. The payoff function of any player in the game on the ring depends on his actions and on the actions of his two neighbors $k \ominus 1$ and $k \oplus 1$. Assuming that the set S_k of actions for each player is limited to the set {**D**,**C**}, the payoff function has 8 entries and Table 2 shows a payoff function u_k^1 used in our study.

It is assumed that each player acts in the game independently and selects his action to maximise his payoff. If players play the game defined by the payoff function u_k^1 and player k and his neighbors $k \ominus 1$ and $k \oplus 1$ all select action **D** in a trial, his payoff will be defined by a sequence of actions (**D**,**D**,**D**) and is equal to 10 (entry 0 of Table 2). If player k selects action **C** while both his neighbors select action **D**, player k will receive the payoff equal to 0 (entry 2 of Table 2). The remaining entries of Table 2 are self-explaining.

The most widely used solution concept for noncooperative games is a Nash equilibrium point (Nash 1950, Wang & Parlar 1989). A Nash point is an N-tuple of actions, one for each player, such that anyone who deviates from it unilaterally cannot possibly improve his

Table 2: Payoff function of a game on a ring

	$s_{k\ominus 1}$	s_k	$s_{k\oplus 1}$	$u_k^1(s_{k\ominus 1}, s_k, s_{k\oplus 1})$
0	**D**	**D**	**D**	10
1	**D**	**D**	**C**	0
2	**D**	**C**	**D**	0
3	**D**	**C**	**C**	0
4	**C**	**D**	**D**	0
5	**C**	**D**	**C**	50
6	**C**	**C**	**D**	0
7	**C**	**C**	**C**	30

expected payoff. If s_k denotes an action of the k-th player, then a Nash equilibrium point is an N-tuple $(s_1^*, s_2^*, \ldots, s_k^*, \ldots, s_N^*)$ such that

$$u_k(s_1^*, s_2^*, \ldots, s_k^*, \ldots, s_N^*) \geq u_k(s_1^*, s_2^*, \ldots, s_k, \ldots, s_N^*) \tag{2}$$

for $s_k \neq s_k^*$ and $k = 1, 2, \ldots, N$. A Nash equilibrium point will define payoffs of all the players in the game. However, we are not interested in the payoff of a given player, but in some global measure of the payoff received by the team of players. This measure can be e.g. the average payoff $\overline{u}(s)$ received by the team as a result of their combined actions' $s = (s_1, s_2, \ldots, s_N)$, i.e.

$$\overline{u}(s) = \left(\sum_{k=1}^{N} u_k(s) \right) / N, \tag{3}$$

and it will be our global criterion to evaluate the behavior of the players in the game. The question which arises immediately concerns the value of the function (3) in a Nash point. Unfortunately, this value can be very low.

Analysing all possible actions' combinations in the game and evaluating their prices, i.e. a value $\overline{u}(s)$, we can find actions' combinations characterized by a maximal price and we can call them maximal price points. Maximal price points are actions' combinations which maximise the global criterion (3), but they can be reached by players only if they are Nash points. A maximal price point usually is not a Nash point and the question which must be solved is how to convert a maximal price point (points) into a Nash point (points).

2.3 Exchange processes in games

It is useful from the point of view of a decentralisation of behavior of players in games with limited interactions to introduce a notion of an exchange process (Varshavsky at al. 1977)

The exchange process in a game is a procedure of redistribution of payoffs between players according to the following rules:

a) the exchange process is given by an oriented graph;

b) for each k-th vertex of the graph, $p_k + 1$ is the number of arcs going out of it and $q_k + 1$ is the number of arcs incoming to it;

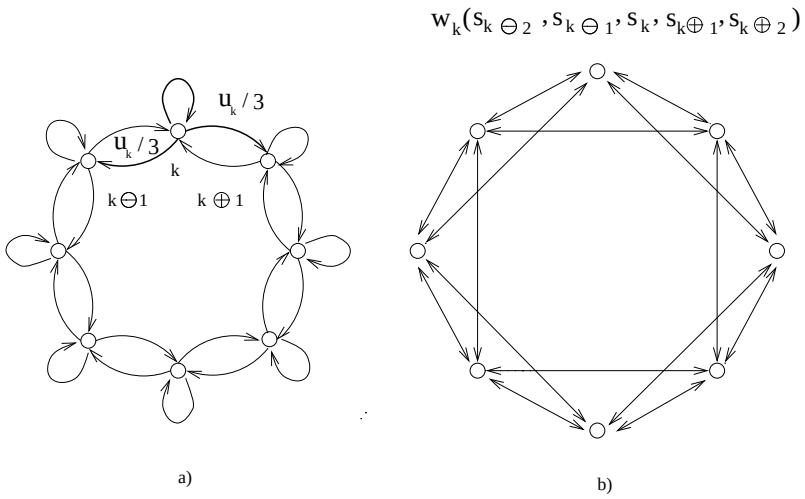

Figure 2: Game on a ring: a) graph of a conjugate exchange process, b) interaction graph of the game transformed by the conjugate exchange process

c) u_k is the payoff of player k (who corresponds to the k-th vertex of the graph) in some actions' combination of the game;

d) each player sends on each outgoing arc a part of his payoff equal to $u_k/(p_k+1)$;

e) each player receives on an incoming arc from each player l a part of his payoff equal to $u_l/(p_l+1)$.

Players taking part in a game with an exchange process share their payoffs. The exchange process results in a transformation of a payoff function of a game into a new payoff.

A conjugate exchange process in a game with limited interaction is an exchange process whose graph coincides with the interaction graph of the game. The conjugate exchange process in the homogeneous game with limited interaction corresponds to the organization of local coalitions with neighbors in the game. It should be noticed that each player in the game takes part simultaneously in $n_k = r$ coalitions, where r is the degree of the interaction graph. Figure 2a shows the graph of a conjugate exchange process for a game on a ring from Figure 1a.

The following theorem is the result of introducing the notion of the conjugate exchange process:

Theorem 1 *Introducing a conjugate exchange process into a homogeneous game with limited interaction transforms a maximal price point into a Nash point.*

It means that an organization of players' coalitions only with the nearest neighbors in the game ensures the achievement by the players' team of the maximal payoff existing in the game.

For the game on a ring from Figure 1a, the conjugated exchange process transforms the payoff u_k of the player k into a new payoff w_k in the following way:

$$u_k \longrightarrow w_k = (u_{k\ominus 1} + u_k + u_{k\oplus 1})/3. \tag{4}$$

It also results in the transformation of the interaction graph of the game as shown in Figure 2b.

Using the idea of an exchange process introducing sharing payoffs in a game, we will consider the following schemes of a game:

- *no cooperation*
- *local cooperation*-sharing a payoff received by a player k with his neighbors in a game, i.e. his payoff is transformed as:

$$u_k \longrightarrow w_k = \frac{\sum_{l \in n_k} u_l}{\max_{l \in \mathbf{N}} n_k + 1}, \tag{5}$$

  where u_l is a payoff of a neighbor l of a player k, n_k is the number of neighbors of the player k, and $\max_{l \in \mathbf{N}} n_k$ denotes a maximal number of neighbors in a game
- *global cooperation*-sharing a payoff received by a player k by all players participating in a game, i.e. his payoff u_k is transformed into a new payoff w_k in the following way:

$$u_k \longrightarrow w_k = \overline{u}(s). \tag{6}$$

3 Coevolutionary Systems

Attempting to apply genetic algorithms (GA) methodology to iterated games with limited interaction we face two difficulties: (a) how to create a GA model of a player who acts concurrently to maximise his payoff, and at the same time (b) how in a dynamic process of the game to search for a solution of the game according to the global criterion (3), i.e. to search for a maximal price point. If we base ourselves on the positions of traditional GAs, we should emphasize the second issue, but in this case we lose the concurrent nature of the game. If we emphasize the concurrent behavior of players, it seems that sequential GAs (Holland 1975, Goldberg 1989) as well as parallel and distributed GAs (Pettey at al 1987, Manderick & Spiessens 1989, Muhlenbein at al 1991, Dorigo & Maniezzo 1993) of both *island* and *diffusion* models are not suitable for our purpose. The common feature of all aforementioned models of GAs is the evaluation of the *global fitness* of an individual, despite his belonging to a global population or subpopulation. For this reason, we call this class of GAs *tightly coupled* GAs. For our purpose, we need an evolutionary system with coevolving subpopulations representing behavior of players, evaluating only their local fitness functions and, at the same time, we expect from such a model a global behavior, in the sense of searching for a global optimum. The need of an extension of the traditional GAs into the direction of coevolutionary systems has been lately recognized. While a *cooperative coevolutionary approach* to function optimization has been proposed (Potter & De Yong 1994), *loosely coupled* GAs (LCGAs) have been proposed (Seredynski 1994b) to support the above described game-theoretic model of computation, suitable for distributed decision-making.

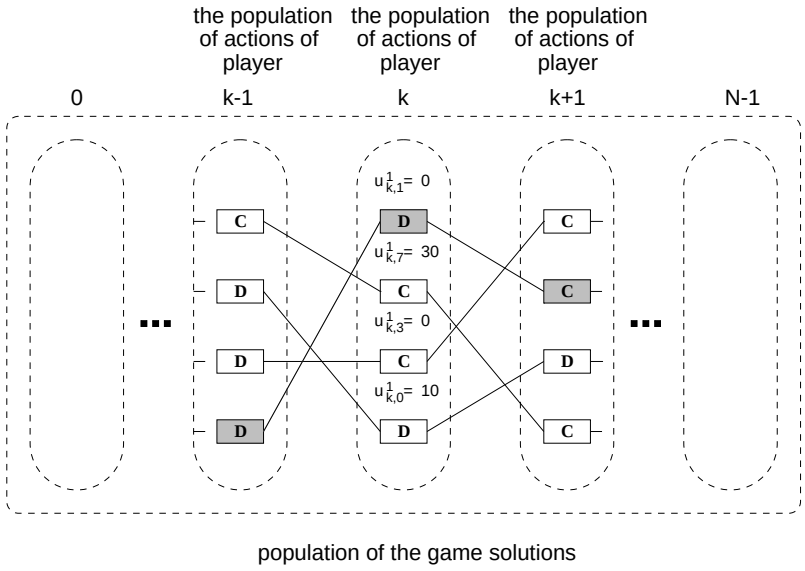

Figure 3: Loosely coupled GA-based system

3.1 Loosely Coupled GA-based Systems

The idea of LCGAs implementing the game on a ring from Figure 1a is shown in Figure 3. The algorithm of the LCGAs can be specified in the following way:

#1: for each player create an initial population of his actions

- create randomly for each player an initial population of size n of player actions taking values from the set S_k (see, Section 2.2) of his actions; Figure 3 shows the initial subpopulations of the size $n = 4$ of actions for players $k \ominus 1$, k and $k \oplus 1$ respectively; the value of n defines a game horison for a player; actions predefined in a subpopulation of a given player will be used in subsequent n games (see, Figure 4)

#2: play a single game

- in a discrete moment of time each player selects randomly one action from the set of actions predefined in his subpopulation and not used until now, and presents it to his neighbours in the game; shadowed actions in Figure 3 show possible situation after the first game: players $k \ominus 1$, k and $k \oplus 1$ have selected respectively actions **D**, **D** and **C**; the chain of selected actions can be interpreted as a possible solution of the game.
- calculate the output of the game: each player evaluates his local payoff u_k in the game; if the payoff function in the game is e.g. the function u_k^1 from Table 2 then player k obtains for action **D** (Figure 3, shadowed box) the payoff $u_{k,1}(\mathbf{D}, \mathbf{D}, \mathbf{C}) = 0$ ($u_{k,1}^1$: read, the entry 1 of the payoff function u_k^1)

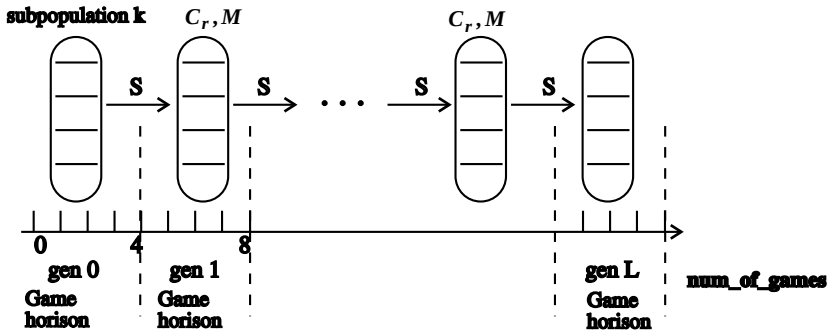

Figure 4: Evolving a subpopulation k in LCGAs from Fig. 3

- if the game with the conjugate exchange process is played, each player informs his neighbours in the game about his current payoff, and calculates his modified payoff w_k.

#3: repeat step #2 until n games are played

#4: using GA operators create for each player a new population of his actions (see, Figure 4)

- after playing n games each player knows the value of his payoff received for a given action from his population (see, Figure 3); chains of actions present actually found solutions of the game
- the values of the payoff are considered as values of a local fitness function defined during a given generation (initially, gen 0) of GA; standard GAs operators of selection (S), crossover (C) (this operator is not used here because of binary values of actions) and mutation (M) are applied locally to subpopulations of actions; these actions will be used by players in games played in the next game horizon

#5: return to step #2 until the termination condition is satisfied

- if a given population evolved during $l+1$ generations then the number of played games num_of_games can be defined as $num\_of\_games = n * (l+1)$.

Below we study the behavior of players in the game implemented with LCGAs. We study it by observation of the global criterion (3) and we will particularly be interested in the following situations:

- there exists a maximal price point in the game, which is not a Nash point
- taking into account the above condition, but with players creating local coalitions defined by the conjugate exchange process

Let us first analyse payoff functions u_k^1 from Table 2 for the game on the ring (Figure 1a). The game with the payoff function u_k^1 has the maximal price point $s_{mp} =$ (**C**,**C**,**C**,**C**,**C**,**C**,**C**,**C**). Each player taking part in the game defined by the vector of actions

s_{mp} obtains a payoff equal to $u^1_{k,7}(\mathbf{C},\mathbf{C},\mathbf{C}) = 30$. The average payoff $\overline{u}^1_k$ received by the team of players achieves the maximal value equal to 30. However, the actions' combination s_{mp} is not a Nash point because it is reasonable for each player to change his action $\mathbf{C} \rightarrow \mathbf{D}$ and obtain a higher payoff equal to $u^1_{k,5}(\mathbf{C},\mathbf{D},\mathbf{C}) = 50$. The game has several Nash points. For one of them, Nash point $s_N = (\mathbf{D},\mathbf{D},\mathbf{D},\mathbf{D},\mathbf{D},\mathbf{D},\mathbf{D},\mathbf{D})$ the inequality (2) is strong for all values of k. One can see that there is no reason for any player to change his action $\mathbf{D} \rightarrow \mathbf{C}$ and obtain instead of the payoff $u^1_{k,0}(\mathbf{D},\mathbf{D},\mathbf{D}) = 10$, a lower payoff equal to $u^1_{k,2}(\mathbf{D},\mathbf{C},\mathbf{D}) = 0$. We can therefore expect that in such a game players will not achieve the maximal price point but will rather play this Nash game.

Below we present the results of experiments with the game on the ring with the number of players $N = 16$. In such a game, the number of possible game solutions (actions' combinations) is equal to 2^{16}. We will use the payoff functions u^1_k. Increasing the number of players in the game will not result substantially in changing the main points of our discussion concerning Nash points and maximal price points. In all experiments, which were conducted on a Sun 10 computer, we observed the game during 250 generations.

Figures 5 and 6 show the average payoff

$$\overline{U}(s) = (\sum_{i=1}^{l+1} \sum_{j=1}^{n} \sum_{k=0}^{N-1} u_k(s))/(n * (l+1) * N) \tag{7}$$

of a player, obtained in the game. Each point of the curves represents the mean of 30 runs of the game.

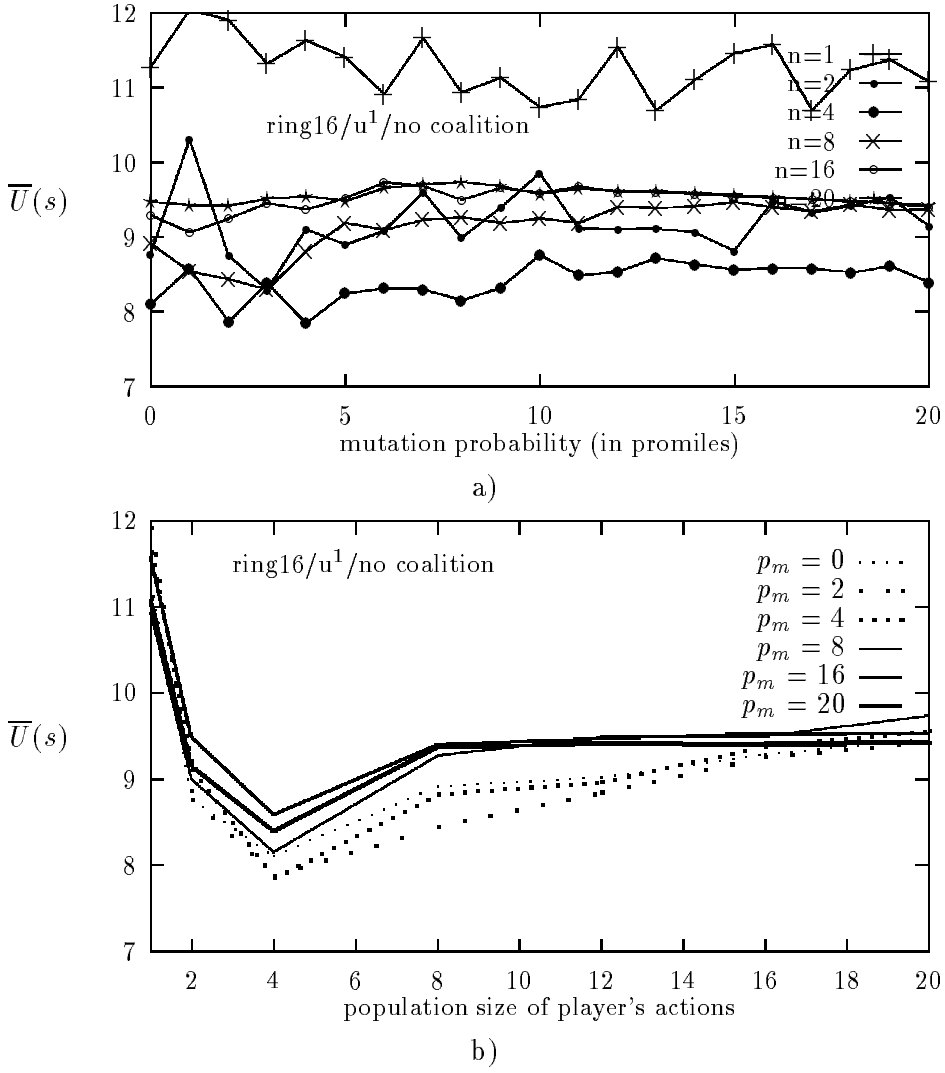

Figure 5: The average payoff received by the team of players during 250 generations as a function of (a) mutation probability, (b) a size of a population of actions of a player.

Figure 5 presents the results of an experimental study of the game with the payoff function u_k^1, without a local cooperation defined by the conjugated exchange process. Figure 5a shows the average payoff received by the team of players during 250 generations as a function of a mutation probability. This curve as well as the following curves represent the mean of 30 runs of the game. One can see that for the observed range $[0, 0.02]$ of mutation probabilities, the average payoff received by the team of players depends little on p_m under constant n, but it depends on the size n of a population of player actions. As shown in Figure 5b, with increasing n, the average payoff received by the team in the game becomes closer to the value 10 defined by the Nash point, and is far from the value defined by a maximal price point.

Results of an experimental study of the game with the payoff function u_k^1, and cooperation according to the conjugated exchange process are shown in Figure 6. One can see that similarly as in the previous experiment, the average payoff received by the team of players depends little on p_m (Figure 6a) under a constant n. It depends largely on n, as demonstrated in Figure 6b. The qualitative change of the players' behavior in the game with the conjugated exchange process is of note, if compared with their behavior in the same game without the conjugated exchange process.

The players discover and play the maximal price point when n is large enough. They now show the ability of the global behavior which is realized is in a fully distributed manner, without any knowledge about either the global optimization criterion or a number of players participating in the game.

3.2 Loosely Coupled Classifier Systems

In this section we study the behavior of a game-theoretic multi-agent system from Section 2 when each agent is a genetics-based machine learning system called a *classifier system* (Goldberg 1989).

Classifier systems constitute the most popular approach to genetics-based machine learning. Recently, they have been succesfully applied (Matwin at al. 1991, Dorigo & Schnepf 1993) to solve some real-life problems. A general framework for studying the behavior of a game-theoretic *multi-agent* system when each agent-player is a classifier system has been proposed (Seredynski at al. 1995) recently. A learning classifier system (CS) maintains a population of decision rules, called *classifiers*, evaluated by using them to generate actions and observing received rewards defined by a payoff function, and modified by periodically applying GAs.

A classifier c is a condition-action pair

$$c =< \textbf{condition} >:< \textbf{action} >,$$

with the interpretation of the following decision rule: if a current observed state matches the **condition**, then execute the **action**. The action part of a classifier is an element of the set S_k of actions a player k in a game with limited interaction. The conditional part of a classifier of CS representing a player k contains his action and actions of his neighbours in the game, and additionally a *don't-care* symbol #.

A real-valued *strength* of a classifier is estimated in terms of rewards obtained according to a payoff function of a player k, using by the player the given classifier to generate an

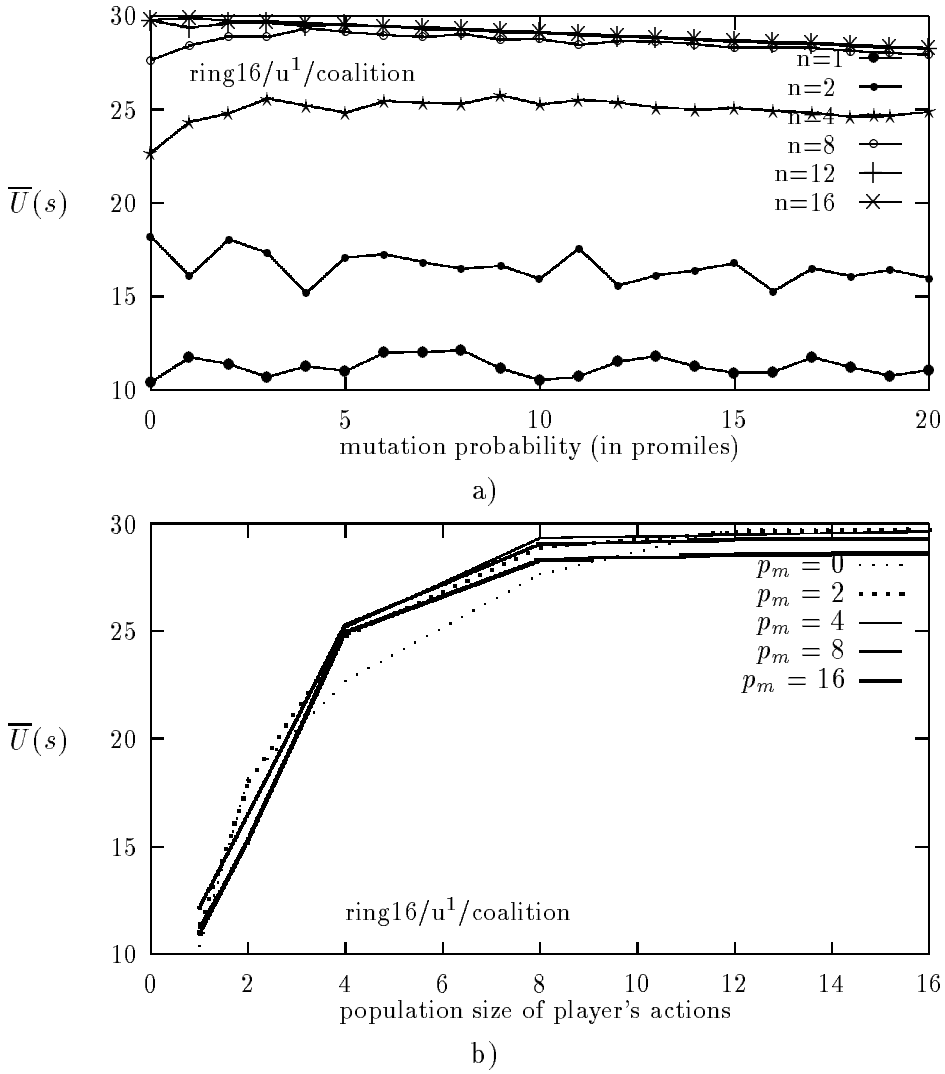

Figure 6: The average payoff received by the team of players during 250 generations as a function of (a) mutation probability, (b) a size of a population of actions of a player.

action. Action selection is implemented by a competition mechanism, where the winner is a classifier with the highest strength.

To modify classifier strengths the simplified *credit assignment* algorithm (Goldberg 1989) was used. The algorithm consists in subtracting a tax of the winnining classifier from its strength, and then dividing equally the reward received after executing an action, among all classifiers matching the observed state.

To create new classifiers a standard GA is applied, with three basic genetic operators: selection, crossover and mutation. Crossover is applied only to the conditional parts of classifiers. Mutation consists in altering with a small probability randomly selected condition elements or actions. GA is invoked periodically and each time it replaces some classifiers with new ones.

We will study below iterated games of CSs. An iterated game consists of a number T of single games $s(t) = (s_0(t), s_1(t), ..., s_{N-1}(t))$ played in subsequent moments of time $t = 0, 1, ..., T-1$, with a value T unknown for players. In a single game played at the moment t each player autonomously selects an action to match an observed state $x_k(t)$ of a game environment. Observed by a player k state $x_k(t)$ of the environment is formed by his and his neighbors' actions played in a previous moment of time. Rewards defined by a payoff function are transfered to players directly or after their redistribution, if an exchange process is used.

A number of experiments with use of CSs as players in the iterated game on a ring has been conducted. A detailed discussion of CSs and GAs setting parameters used in experiments can be found in (Seredynski at al. 1995). Figure 7 shows some results of experiments with 8 players participating in an iterated game defined by a payoff function from Table 2 and consisting of 10000 games. The figure shows the average payoff $\overline{U}_t(s, \Delta t)$ of players, received during each $\Delta t = 50$ games, i.e.

$$\overline{U}_t(s, \Delta t) = \left(\sum_{t'=\text{entier}(t/\Delta t)}^{\text{entier}(t/\Delta t)+\Delta t-1} \overline{u}(s, t') \right) / (\Delta t), \tag{8}$$

in a game without cooperation (each player receives a payoff according to his payoff function)(see, Figure 7a), in a game with a global cooperation (Figure 7b), and in a game with a local cooperation (Figure 7c) respectively.

One can see that in the iterated game without a cooperation the multi-agent system converges to a steady-state with a corresponding the average payoff of a team of players equal to 10, defined by the Nash point. In the game with a global coperation a self-organization process can be observed. The system converges to a steady-state corresponding to playing the maximal price game, providing the maximum value of the average payoff received by the players and equal to 30. In the game with a local cooperation a similar adaptive process can be observed. One can see that in the system, which is fully distributed, a global behavior evolves, and is achieved only by a local interaction between players acting without a global knowledge about a game, i.e. about the number of players in a game and a global optimization criterion.

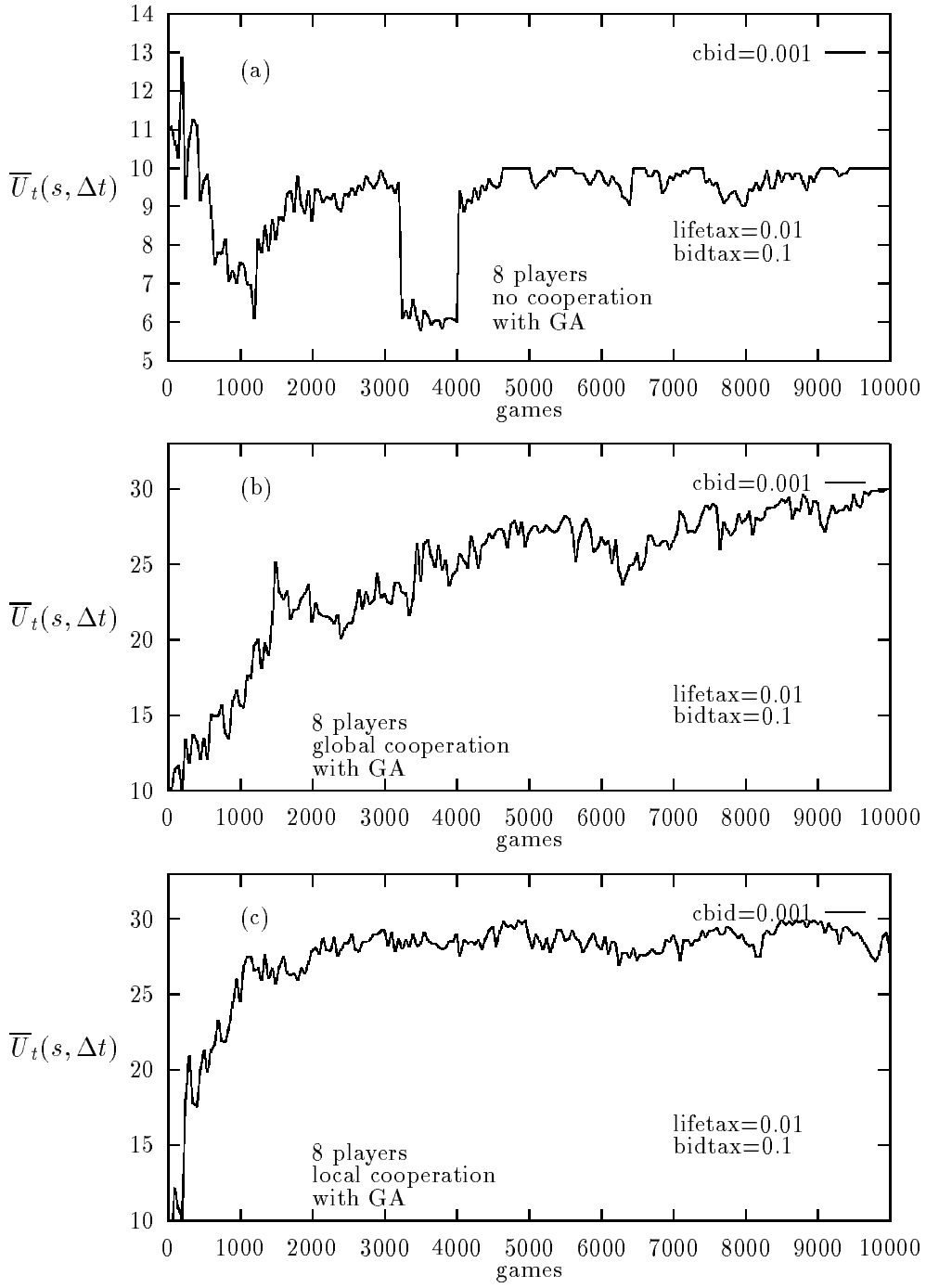

Figure 7: Games of classifier systems without cooperation (a), with a global cooperation (b), and with a local cooperation (c)

4 Applications

Parallel and distributed computers built today consist of thousands of processors which need to coordinate their work. It is hard to imagine a centralised control of such systems, therefore the efficient distributed algorithms for such a control are needed. In this section we consider two problems from the area of parallel and distributed processing, namely the mapping and scheduling problems. These problems have the key influence on the performance of parallel and distributed systems. They both are known to be NP-complete and searching for effective heuristics, for the problems is a subject of a current research in the area. To solve the problems we propose to use the methodology of evolutionary coevolving multi-agent systems.

4.1 Dynamic Mapping Problem

Current results concerning mapping communicating processes of a parallel program into parallel architectures show that applying static mapping algorithms raises the efficiency of using parallel computers, when characteristics of programs do not change during their execution. However, if characteristics of the programs change in time, static policies can not be efficiently applied. For this reason, a growing interest in developing algorithms and environments enabling applying dynamic mapping policies can be observed.

We propose an approach to dynamic mapping based on a multi-agent interpretation of a parallel program migrating in a parallel system environment to search an optimal allocation of program modules in a topology of a parallel system. To find multi-agent system strategies of migration in the system graph we use LCGAs presented in the previous section.

Models of parallel programs and parallel computers presented below are oriented on MIMD machine environment and are formulated as follows. A parallel program is represented by a weighted undirected graph $G_p =< V_p, E_p >$ with a set V_p of N_p nodes and a set E_p of edges. The nodes of the program graph represent processes and the edges represent a fixed communication pattern between processes. Node weights b_k characterize computation costs of the processes and edge weights a_{kl} characterize a communication cost between a given pair of processes located in neighbouring system nodes. It is assumed that the weights of the program graph represent time-averaged properties of the processes and the communications between them for the program processed in a given parallel computer. A parallel computer is represented by an undirected graph $G_s =< V_s, E_s >$ called a system graph, with a set V_s representing nodes of the system and a set E_s representing an interconnection pattern of the system .

Let θ be a mapping function from the vertex set of the program graph to the vertex set of the system graph and Θ the set of all mapping functions, i.e. $\Theta = \{\theta : V_p \rightarrow V_s\}$. We suppose that a local cost function $C(k, \theta)$ is defined for kth program node mapped to the system graph. This function is defined as follows (Seredynski 1994a):

$$C(k, \theta) = C_1(k, \theta) + C_2(k, \theta), \tag{9}$$

where $C_1(k, \theta)$ is a cost of communications of the kth program node with its neighbour program nodes, and $C_2(k, \theta)$ is the computational load of a system node to which the kth

program node was mapped. Local communication and computational cost functions are defined as follows:

$$C_1(k,\theta) = 0.5 \sum_{l=1}^{r_k} a_{kl} * d_{min}(\theta(k),\theta(l)), \tag{10}$$

and

$$C_2(k,\theta) = \sum_{n=1}^{n_i} b_n, \tag{11}$$

where r_k is the number of neighbour program nodes of the kth node; $d_{min}(\theta(k),\theta(l))$ is a minimal number of hops between system nodes $\theta(k)$ and $\theta(l)$ where are located neighbour program nodes k and l respectively, and n_i is a number of program nodes (not including a program node k) located in the system node $\theta(k)$.

The problem of static mapping can be formulated now as the problem of seeking a mapping function $\theta \in \Theta$ that minimises the total cost function $C(\theta)$ defined as a sum of the local cost functions (9), i.e.

$$\min_{\theta\in\Theta}(C(\theta) = \sum_{k\in V_p} C(k,\theta)). \tag{12}$$

The dynamic mapping formulation of the problem can now be easily obtained due to locally defined cost functions and a multi-agent interpretation of the mapping problem.

We assume that a collection of agents is assigned to nodes of the program graph in a such way that one agent is assigned to one program node. Each agent has some number of actions which influence a local cost function of a program node attached to the agent. If nodes of the program graph together with agents attached to them are placed in some way, e.g. randomly into the system graph, the agents' actions can be interpreted in terms of possible moves of the agents in the system graph.

The dynamic mapping algorithm can be specified in the following way:

- agents assigned to the processes and located in some system nodes are considered as players taking part in a game with limited interaction; each agent-player has $r+1$ actions interpreted as follows: do not migrate or migrate to one of r nearest neighbour system nodes, where r is the degree of the system graph. Taking an action by a player corresponds to the player simulating the action, not a physical move
- each player has a local cost function (9) describing the communications and computational costs; the function depends on the action of a given player and the actions a limited number of neighbours (according to the program graph)
- the objective of each player is to minimise its local cost function; a player maintains a local communication with neighbour players informing them about its current action
- behavior of the players in the game is observed through the global criterion (12) of the mapping problem; the objective of the game is to find the optimal (in the sense of the global criterion) directions of the migration of the program graph in the system graph while each player can move at the distance of one hop only

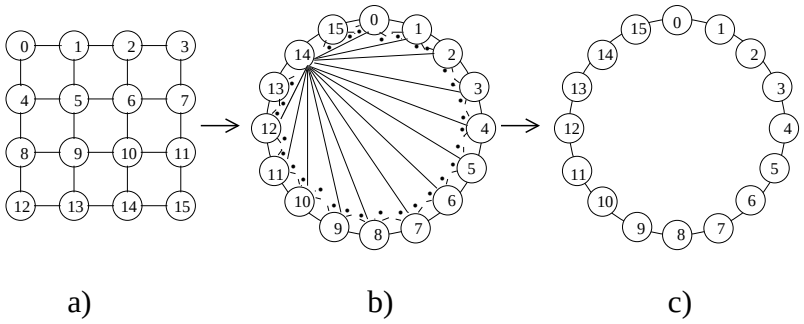

Figure 8: Behavior of dynamically changing program represented by a sequence of three program graphs respectively a), b) and c)

- after a predefined number of games a migration of the program nodes (together with the agent-players) is performed; the game starts again.

The evolutionary coevolving multi-agent system with LCGAs implementation (see, Section 3.1) was used to solve the dynamic mapping problem. In the experiment described below it was supposed that a parallel program dynamically changed in time during its execution (see, Figure 8) and a target parallel machine was a torus 4×4 (similar to Figure 1c). Figure 8 shows behavior of the parallel program represented by a sequence of three graphs describing changing in time the communication pattern and a computation load of the program. Figure 9 shows performance of the dynamic mapping algorithm implemented with use of LCGAs. The program represented initially by a grid 4×4 from Figure 8a (with parameters $a_{kl} = 10$, $b_k = 2$) is randomly mapped into the target system, what results in the initial physical allocation of the program with the cost $C(\theta) = 450$ (see, Figure 9). It is assumed that the execution of the program begins, but at the same time the distributed dynamic mapping algorithm starts its work monitoring the system, and searching optimal migration strategies for program modules. One can see, that during the first 50 generations of LCGAs, the system is able to find directions of program modules migration in a fully distributed way, providing decrease in the the cost allocation. After 50 generations, a physical migration of the program modules is performed, which results in a new, better allocation, and the dynamic mapping algorithm continues to work. After three migrations of the program graph a near optimal mapping is found.

It is assumed that after 130 generations the parallel program changes its communication pattern of activity and computation load, which is modelled by a complete graph shown in Figure 8b (only edges for a selected node are shown, parameters $a_{kl} = 2$, $b_k = 10$). It is clear that the previously found allocation of the program is not optimal now, and the value of the cost function $C(\theta)$ is now near to 800 (see, Figure 9). The distributed dynamic mapping algorithm starts immediately to search for a new optimal allocation, without the need of any central synchronization (which would be necessary in the case of tightly coupled GAs).

After 260 generations the parallel program changes its communication activity again, as modelled by the graph from Figure 8c (with parameters $a_{kl} = 2$, $b_k = 10$). The behavior of the distributed dynamic mapping algorithm is similar to that described above. The

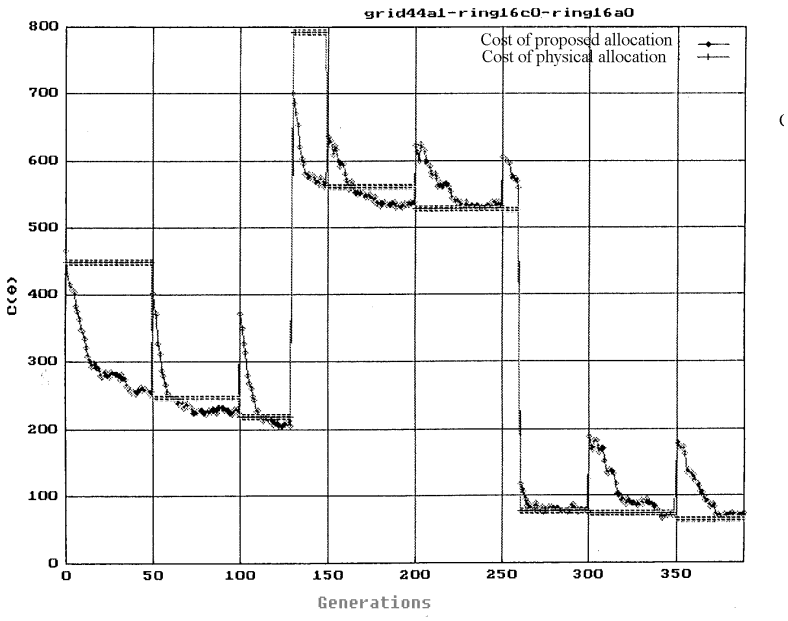

Figure 9: Performance of distributed dynamic mapping algorithm

algorithm is able to catch the changes of the program graph parameters without any central synchronization and continues in a fully distributed way searching for an optimal mapping.

4.2 Scheduling Problem

To design a parallel and distributed scheduler we will interpret a parallel program represented by a precedence task graph as a multi-agent system with agents taking part in a game with limited interaction. We assume that a collection of agents is assigned to tasks of the program graph in a such way that one agent is assigned to one task.

The scheduling algorithm can be specified in the following way:

- agents assigned to tasks of a parallel program and located in system graph nodes are considered as players taking part in a game with limited interaction
- the objective of each player is to minimise the response time T_r
- the objective of the iterated game is to find the optimal (in the sense of the global criterion T_r) directions of the migration of the program graph in the system graph while each player can move at the distance of one hop only
- after a predefined number of games a migration of the tasks is performed, which results in a new better value of T_r; the game starts again.

Each agent is a CS learning machine and it is responsible for local mapping decisions concerning a given task. Each CS has a number of classifiers describing rules and corresponding to them decisions which are used in a process of an iterated game.

Designing classifiers of CSs is the most responsible and difficult part of the proposed approach to a scheduling problem. Classifiers will describe accepted attributes and basic heuristics and will influence a performance of the proposed approach. As mentioned in Section 3.2, each classifier has two parts: a conditional and action part. It is assumed that a conditional part of a classifier will describe situations which can be recognized by an agent P_k located in some system node. Objects recognized by an agent are some specific tasks of a program graph. Objects are considered as being "close" to an agent, if a distance to them in a system graph is less or equal then d hops; otherwise, they are considered as being "far" from him.

Objects recognised by an agent are

- an immediate task predecessor which data arrives last
- immediate tasks predecessors and immediate tasks succesors with the highest processing time
- tasks from the same level of the program graph as a task associated with a given agent
- not immediate tasks predecessors and succesors.

All these objects can be classified as being all "far", as some of them being "close" or all of them being "close". Some other attributes of a situation in a system graph can be recognised, such as a delay of a given task to be processed on a processor or to a communication channel, a total delay of tasks located in a given node to a processor and to communication channels. These delays can be classified as "accetable" or "not acceptable" by a comparison with similar charactristics of respectively tasks located in the same system node, or with a situation in neighboring system nodes.

An action part of a classifier will describe potential actions (moving on a distance of one hop in the system graph) which can be taken by an agent in a given situation. The actual list contains 16 actions, and below are given some them:

- stay in a processor where you are located (do not move)
- move to a randomly chosen neighboring processor
- move in the direction where is located your randomly chosen predecessor/succesor
- move in the directon where is located a predecessor with the highest processing time
- move in the direction where is located a predecessor which data arrives last

Conditional and action parts of a classifier contain, coded with use of binary substrings, and - a don't care symbol # (only conditional part) information concerning the condition and the action, represented by the classifier. For example a classifier

$$< 1\ 0\ \#\ 01\ \#\#\ \#\#\ >:< 0011 >$$

can be interpreted in the following way:
IF (an immediate task predecessor which data arrives last is "far"
AND immediate task predecessor with the highest processing time is "close"
AND immediate task successor with the highest processing time is not important where (# symbol)

AND some of tasks from the same level are "close"
AND not immediate tasks predecessor are not important where
AND not immediate tasks successors are not important where)
THEN move in the direction where is located a predecessor with the highest processing time.

An intial population of classifiers for each CS is randomly created. In the process of a game the usefulness of classifiers is evaluated. New classifiers are generated with use of GAs and they replace weak classifiers. As the result of the game, the global behavior of the system is expected, resulting in a mapping program tasks to processors, corresponding to optimal or suboptimal scheduling.

4.3 Implementation of a Simple Version of the Scheduler

A game theoretic- base scheduler presented in the previous section is a complex system, therefore to evaluate a potential usefulness of the proposed aproach a simple version of the system has been implemented and tested. In a given implementation of the algorithm each agent is modelled by a CS-like system with the following properties:

- a set of actions is limited to the following four actions:
  a1: move to a randomly chosen processor
  a2: move to a processor most frequently used by your immediate predecessor and successors
  a3: move to a processor least frequently used by your immediate predecessor and successors
  a4: move to a processor selected by one of your predecessors
- a classifier is limited to only an action part
- each CS-like system has a constant number of four classifiers, each of them corresponding to one of actions
- each classifier has the same initial value of a strength str_i^k ($i = 1, 2, 3, 4; k = 1, 2, ..., N_p$), which is modified during a game
- a winning classifier is a one with the highest strength, i.e. a classifier i with

$$\max_i \ str_i^k \tag{13}$$

- a strength of a classifier selected to take action is increased in the case of a success, i.e.

$$str_i^k = str_i^k + 1, \quad \text{if} \quad T_r^{new} < T_r^{old} \tag{14}$$

  and decreased in the case of a failure, i.e.

$$str_i^k = str_i^k - 1, \quad \text{if} \quad T_r^{new} \geq T_r^{old}, \tag{15}$$

  where T_r^{old} and T_r^{new} are response times respectively, before an execution of an action, and after an execution of the action.

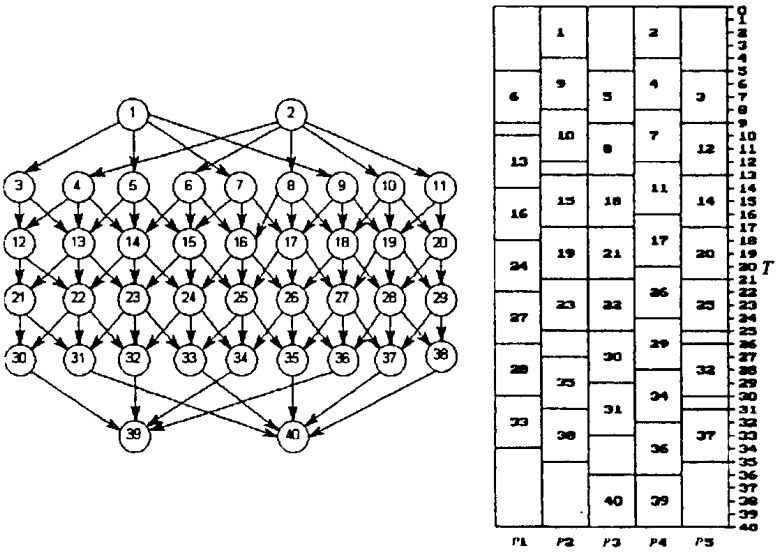

Figure 10: Precedence task graph (left), and a schedule for the task graph (right).

A single game is conducted as a sequence of moves (actions) of agents, i.e. at a given moment of time only one agent takes an action, and a single game is completed in N_p moments of time. A game consists of G single games g, i.e. $g = 1, 2, ..., G$. A game can be played with two options: a player can withdraw its actions in the case of failure, or a player can not do it.

A number of experiments with different precedence graphs has been conducted to test the proposed approach to scheduling. Figure 10(left) an example of the precedence task graph taken from (Schwehm & Walter 1994), with processing and communication times equal to 4 and 1 respectively, and scheduled on a fully connected 5-processors system. Some results of experiments with this precedence graph are described below.

Figure 11(a) shows a run of the scheduler when a withdrawal of actions by players playing a game is not allowed. One can see that in such a game only a random search is performed. Figure 11(b) shows a typical run of the scheduler when a withdrawal of actions in a game is allowed. In such a game, after some number of single games, a solution is found. Figure 10(right) shows the solution represented in a form of a Gantt chart showing the allocation of each task on processors, and the times when a given task starts and finishes his execution. The schedule has found, which defines the response time $T_r = 40$, is better than that presented in (Schwehm & Walter 1994), whose response time was equal to 43.

Figure 11(c) gives some insight into a process of searching a solution during a game, showing changing the average value $\overline{str}_i(g)$ of each classifiers in a process of a game, i.e.

$$\overline{str}_i(g) = \left(\sum_{k=1}^{N_p} str_i^k(g) \right) / N_p. \tag{16}$$

The figure shows that some order of importance of classifiers in a game is discovered and maintained. While the game ranges the classifiers in the order of their decreasing

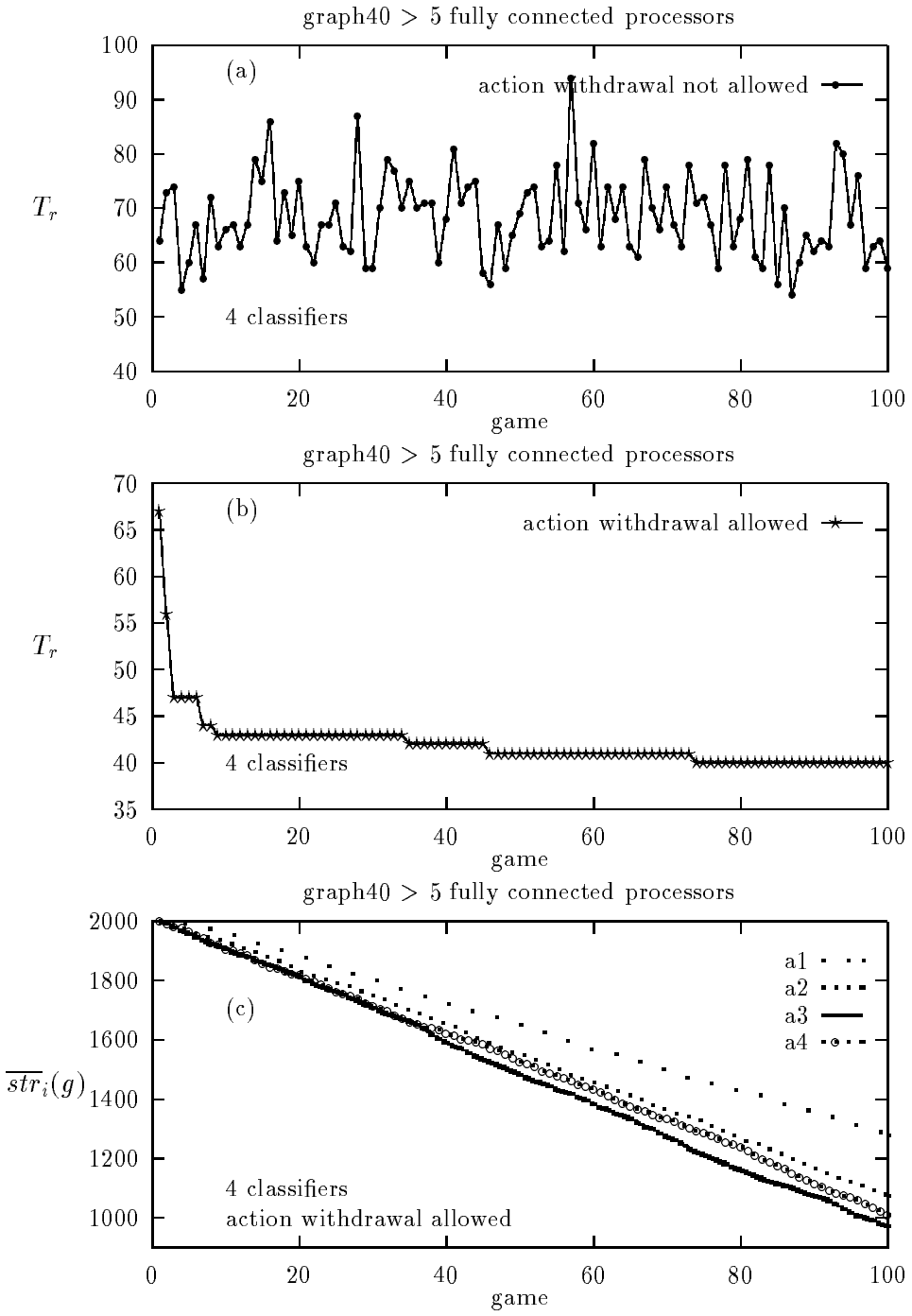

Figure 11: Game-based scheduler: action withdrawal not allowed (a), action withdrawal allowed (b), strengths of actions (c)

Table 3: Response time for a different number of fully connected processors

n	2	3	4	5	6	7	8	9	10
T_r	80	57	46	40	36	33	31	29	29

importance from *a1* to *a3*, a discovery of a solution is a result of a combined use of the classifiers.

Described in this section game-based scheduler has been compared with a standard genetic algorithms-based scheduler (Seredynski and Frejlak 1994). While both schedulers found the same responce time for a given n-fully connected processor system (see, Table 2), the time needed to find a solution by the game-based scheduler was expressed in seconds (Pentium 133 computer), and for the GA-based scheduler it was expressed in minutes. To find an optimal solution for e.g. 5-processors system the average time of 10 runs of each scheduler was equal to 1.3 sec. and 1.2 min. respectively.

5 Conclusions

We have considered in the paper the model of noncooperative games with limited interaction. We addressed the problem of a global behavior of the team of players, measured by the value of the average payoff received by the team in the iterative game. We showed the rules of a local interaction between players providing a global behavior of the system. To implement a concurrent nature of players' behavior we proposed two parallel and distributed GA-based schemes with an evaluation of local fitness functions of players. Conducted experiments have shown that the system is capable of evolving a global behavior, if rules of local cooperation between players are preserved. Behavior of the system in its main points is similar to that predicted by game theory with its concept of a Nash equilibrium point.

We have applied the developed evolutionary coevolving multi-agent system to solve two problems from the area of parallel and distributed processing: the dynamic mapping problem and scheduling problem. We believe that presented results give rise to think that the developed model can serve as a useful metaphor for distributed decision-making and distributed control in real life systems.

References

Axelrod R. (1987) The Evolution of Strategies in the Iterated Prisoners' Dilemma. In: L. Davis (ed.): *Genetic Algorithms and Simulated Annealing.* London: Pitman.

Barto A.G., Anandan P. (1985). *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-15, No 3, May/June, 360-375

Błażewicz J., Ecker K. H., Schmidt G., Węglarz J. (1994) Scheduling in Computer and Manufacturing Systems, Springer

Chlebus B. S., Diks K., Pelc A. (1994), Fast Gossiping with Short Unreliable Messages, *Information Processing Letters*

Dorigo M., Maniezzo V. (1993) Parallel Genetic Algorithms: Introduction and Overview of Current Research. J. Stender (ed.): *Parallel Genetic Algorithms* IOS Press

El-Rewini H., Lewis T. G., Ali H. H. (1994), Task Scheduling in Parallel and Distributed Systems, PTR Prentice Hall

Fogel D.B. (1991). The Evolution of Intelligent Decision-Making in Gaming. *Cybernetics and Systems*, 22, 223-236.

Fogel D. B. (1993) Evolving Behaviors in the Iterated Prisoner's Dilemma. *Evolutionary Computation*. vol. 1. N 1.

Fox G. C., Furmansky W. (1988), Load Balancing Loosely Synchronous Problem with a Neural Network, Proc. of 3rd Conf. on Hypercube Concurrent Computers and Application

Fox G.C., Johnson M., Lyzenga G., Salmon J. and Walker D. (1988). *Solving Problems on Concurrent Processors*, Prentice Hall

Genesereth M. R., Ginsberg M.L., Rosenschein J. S. (1988) Cooperation without Communication. A.H. Bond and Les Geser (eds.). *Readings in Distributed Artificial Intelligence*. Los Angeles

Goldberg D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley. Reading. MA

Holland J.H. (1975) Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press

Kuwabara K., Ishida T. (1992) Symbiotic Approach to Distributed Resource Allocation: Toward Coordinated Balancing. *The 4-th European Worskhop on Modeling of Autonomous Agents in a Multi-Agent Word*. S. Martino el Cimino. Italy

Levy R., Rosenschein J. S. (1992) A Game Theoretic Approach to Distributed Artificial Intelligence and the Pursuit Problem. E. Werner, Y. Demazeau (eds.): *Decentralized A.I.-3*. Elsevier

Lomborg B. (1992) Game Theory vs. Multiple Agents: the Iterated Prisoner's Dilemma. *The 4-th European Workshop on Modeling of Autonomous Agents in a Multi-Agent Word (MAAMAW'92)*. S. Martino el Cimino Italy

Manderick B., Spiessens P. (1989) Fine-Grained Parallel Genetic Algorithms. *Proc. of the Third Int. Conf. on Genetic Algorithms*

Moon Y., Sklansky J. (1990). A Class of Mapping Algorithms for Hypercube Computers. *The Fifth Distributed Memory Computing Conference*. Charleston. South Caroline. IEEE Computer Society Press

Moore E.F. (1957). Gedanjen - Experiments on Sequential Machines: Automata Studies. *Annals of Mathematical Studies*, 34, 129-153. Princeton, NJ: Princeton University Press

Muhlenbein H., Schomisch M., Born J. (1991) The Parallel Genetic Algorithms as Function Optimizer. *Proc. of the Fourth International Conference on Genetic Algorithms*

Nash J. F. (1950) Equilibrium Points in n-Person Games. *Proc. of the National Academy of Sciences USA* 36, pp. 48-49

Ordeshook P. C. (1986) *Game Theory and Political Theory: an Introduction:* Cambridge University Press

Pettey C.B., Lutze M.R., Grefenstette J. J. (1987) A Parallel Genetic Algorithm. *Proc. of the Second Int. Conference on Genetic Algorithms*

Plateau B., Trystam D. (1992), Optimal Total Exchange for 3-D Torus of Processors, *Information Processing Letters*

Potter M.A., De Jong K.A. (1994) A Cooperative Coevolutionary Approach to Function Optimization, *Parallel Problem Solving from Nature – PPSN III*, Y. Davidor, H. -P. Schwefel and R. Männer (eds.), LNCS 866, Springer

Rapoport A. (1966) *Optimal Policies for the Prisoner's Dilemma*. Tech. Report N 50. The Psychometric Laboratory: Univ. of North Carolina

Seredynski F.(1990) *Homogeneous Networks of Learning Automata*. Tech. Report N 684. Institute of Computer Science PAS. Warsaw

Seredynski F. (1994a) Dynamic Mapping and Load Balancing with Parallel Genetic Algorithms. *IEEE World Congress on Computational Intelligence*. Orlando. Florida.

Seredynski F. (1994b) Loosely Coupled Distributed Genetic Algorithms, *Parallel Problem Solving from Nature – PPSN III*, Y. Davidor, H. -P. Schwefel and R. Männer (eds.), LNCS 866, Springer

Seredynski F., Cichosz P. and Klebus G. P. (1995) Learning Classifier Systems in Multi-Agent Environments, First IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA), Shefield, UK

Schwehm M., Walter T. (1994) Mapping and Scheduling by Genetic Algorithms, CONPAR 94 - VAPPVI, B. Buchberger and J. Volkert (eds.), LNCS 854, Springer

Tsetlin M. L. (1973) *Automaton Theory and Modelling of Biological Systems*. Academic Press. N.Y.

Varshavsky V. I. (1972) Some Effects in the Collective Behaviour of Automata, *Machine Intelligence* 7. Edinbourgh University Press

Varshavsky V. I., Zabolotnyj A. M., Seredynski F. (1977) Homogeneous Games with a Conjugate Exchange Process. *Proc. of the Academy of Science USSR Techniczeskaja Kibernetika*. N 6

Wang Q., Parlar M. (1989). Static Game Theory Models and Their Applications in Management Science, *European Journal of Operational Research* 42. North-Holland. 1-21

Wong S.T.C. (1993). *Preference-based Decision Making for Cooperative Knowledge-based Systems*. ICOT Technical Report: TR-0827