Parallel and Distributed Computing with Coevolutionary Algorithms

Franciszek Seredyński
Department of Computer Science
Institute of Insurance
Wesola 52, 25-353 Kielce, Poland
sered@wsu.kielce.pl

Albert Y. Zomaya
Parallel Computing Research Laboratory
Department of Electrical and Electronic Engineering
The University of Western Australia
Nedlands, Perth, Western Australia 6907, Australia
zomaya@ee.uwa.edu.au

Abstract

The problem of parallel and distributed function optimization is considered. Two coevolutionary algorithms with different degrees of parallelism and different levels of a global coordination are used for this purpose and compared with sequential genetic algorithm (GA). The first coevolutionary algorithm called a loosely coupled genetic algorithm (LCGA) represents a competitive coevolutionary approach to problem solving and is compared with another coevolutionary algorithm called cooperative coevolutionary genetic algorithm (CCGA). The algorithms are applied for parallel and distributed optimization of a number of test functions known in the area of evolutionary computation. We show that both coevolutionary algorithms outperform a sequential GA. While both LCGA and CCGA algorithms offer high quality solutions, they may compete to outperform each other in some specific test optimization problems. The LCGA may be recommended to be used in optimization systems when high degree of parallelism is possible and non global coordination is expected while the CCGA algorithm is useful when low degree of parallelism and global coordination is acceptable.

1 Introduction

The use of evolutionary computation (EC) techniques to evolve solutions of both theoretical and real-life problems has seen a dramatic increase in popularity and success over last decade. The most popular and widely applied EC technique was a sequential GA ([5, 8]) which computational scheme is based on a single population of individuals representing a single species. Development of parallel machines stimulated parallelization of a sequential GA and resulted in two parallel EC techniques known respectively as island model and diffusion model (see, e.g. [4, 14]). These both models widely used today have been still exploring a notion of a single species, but individuals representing the species live in different subpopulations.

While these techniques are very effective in many applications, new more difficult problems were set. These problems (e.g. modeling economic phenomena such as a market) are in their nature distributed, i.e. can be seen as a number of independent interacting entities with own goals, where a global behavior can observed as the result of interactions. To meet these new requirements, researches in the area of EC were looking for new more powerful paradigmes of natural processing. In the result coevolutionary algorithms based on modeling phenomena of coexistance of several species emerged [6] as a very promising area of EC.

A very popular framework closely related to the development of coevolutionary algorithms are multi-agent systems. Solutions developed on multi-agent platforms are characterized by a high degree of a parallelism and a distributed control, which result in decreasing the complexity of the developed algorithms and increases their efficiency. However, to efficiently apply the multi-agent systems methodology, some theoretical and practical issues must be solved. Among them, in particular, are the questions: (a) what should be the nature of an agent taking a decision, (b) how agents should interact which each other to solve a problem, and (c) how to efficiently implement multi-agent systems.

In this paper we address a number of these questions. We present a competitive coevolutionary algorithm called LCGA[12]), which is based on a game-theoretical model. The algorithm is parallel and distributed and can be interpreted as a multi-agent system with locally expressed (where it is possible) goals of agents and a global behavior of the system. We use our algorithm to solve the problem of optimization of function and compare it with another known from the literature cooperative coevolutionary algorithm CCGA ([11]) which is partially parallel and needs a global synchronization.

The paper is organized as follows. In the next section we discuss issues concerning parallel and distributed algo-
rithms. In Section 3 we overview models existing in the area of EC and in Section 4 describe test functions used in experiments. Sections 5 and 6 contain presentation of two coevolutionary algorithms LCGA and CCGA studied in the paper. Section 7 contains results of experiments conducted with use of coevolutionary algorithms. The last section contains conclusions of the paper.

2 Parallel and Distributed Algorithms

The main motivation of parallelization of any sequential algorithm is the desire to reduce the total execution time of the algorithm. A straightforward approach to doing that is extracting as much parallelism as possible immediately from the sequential algorithm or after redesigning it. Extracted computation subcomponents of an algorithm called tasks can be executed concurrently in a parallel architecture, what potentially results in decreasing overall completion time of the algorithm. A sequential algorithm converted in this manner into a parallel algorithm is typically represented by a weighted and usually directed acyclic graph $G_p = \langle V_p, E_p \rangle$, where the node set $V_p$ is a set of tasks and the set $E_p$ of edges describes a precedence constraint relation between tasks. The issue related to this stage is a problem of choosing a granularity of tasks. The granularity which can be either fine-grained or coarse-grained should be chosen to match the most effectively an available parallel architecture of the SIMD or MIMD classes. The final step to execute effectively parallel programs in parallel architectures is to solve a mapping problem [3, 18] which is known to be NP-complete, i.e. to assign tasks to processors and define an order of their execution in the processors.

While parallel computing, exploiting so-called functional and data parallelism, is oriented toward solving a problem on more than one processor to solve it faster, distributed computing, dealing with distributed algorithms and exploiting so-called parallelism, is different. Distributed computing is oriented rather to collective information processing directed towards controlling some crucial parameters or resources of a distributed system, which can be met in some areas of application, such as e.g. management or control. A distributed algorithm [1] is represented by a connected directed graph $G_d = \langle V_d, E_d \rangle$, where the set $V_d$ is a set of tasks and a set $E_d$ is a set of unidirectional channels between tasks. Each node of the graph, corresponding to a task, is driven either by a local clock (asynchronous model) or by the global clock (synchronous model). Tasks are reactive entities - they perform computation as a response to the receipt of messages from other tasks. The important issues are the degree of knowledge available in a node and how a global goal of the system can be realized when nodes do not have complete information about the graph $G_d$.

3 Evolutionary Computation Models

3.1 Sequential Genetic Algorithms

The most known and widely applied evolutionary algorithm [8, 5], often termed as the Classical GA is a sequential centralized algorithm and which is presented below in OCCAM-like notation.

Algorithm: Classical GA

\textbf{chromosome:} global structure representing a solution $\bar{x} = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}$
\textbf{optimization criterion:} global function $f(\bar{x})$
\textbf{population:} global population $P()$
\textbf{SEQ}
\begin{align*}
    t &= 0 \\
    &\text{initialize } P(t) \\
    &\text{evaluate } P(t) \\
    \text{WHILE} &\text{termination \_condition NOT TRUE}
    \text{SEQ}
    \begin{align*}
        t &= t + 1 \\
        &\text{select } P(t) \\
        &\text{crossover in } P(t) \\
        &\text{mutate in } P(t) \\
        &\text{evaluate } P(t) \\
    \end{align*}
\end{align*}
\textbf{problem \_solution} = the best individual $\bar{x}$ from $P(t)$

The algorithm maintains a global population $P()$ of chromosomes coding a global solution $\bar{x}$, and evaluated by a global function $f(\bar{x})$. In each cycle $t$, called a generation, the algorithm executes sequentially three basic genetic operators: selection, crossover and mutation, and evaluates the fitness of new created individuals. An evolutionary process is continued until some stop condition (usually exceeding $t_{\text{max}}$) is satisfied. After that, the best individual from the last population $P(t)$ represents an optimal or suboptimal solution of a problem.

3.2 Parallel Genetic Algorithms

Two parallel EC techniques known respectively as island model and diffusion model (see, e.g. [4, 14]) are used today. In the island model a global population $P()$ is divided into some number of subpopulations $P()$ called islands, which are located on different, usually the MIMD class processors, and evolve in parallel. Evolutionary processing of individuals in each subpopulation is performed during some predefined number of generations called an epoch, in the same sequential way as it done in the classical GA. After each epoch neighbor subpopulations communicate by exchanging some number of their best individuals. All subpopulations contain chromosomes representing a global solution $\bar{x}$, and are evaluated according to a global function $f(\bar{x})$. A structure of a population composed of subpopulations is defined by a topology of a communication graph.
which specifies a neighborhood of each subpopulation and serves to exchange individuals between neighbor subpopulations. Both a number of subpopulations and a topology of the communication graph are user defined parameters.

In the diffusion model structural changes in a global population come to the level of individuals. It is assumed now that each individual can be treated as a subpopulation, located usually on the SIMD class processor. All individuals are considered as nodes of a user defined communication graph, usually a 2D greed. A local neighborhood of each individual is specified and genetic operators are defined on possibly overlapping neighborhoods. All individuals evolve in parallel, exchanging messages with individuals within a neighborhood to perform a local selection of parents for their recombining.

### 3.3 Coevolutionary Genetic Algorithms

The idea of coevolutionary algorithms comes from the biological observations which shows that coevolving some number of species defined as collections of phenotypically similar individuals is more realistic than simply evolving a population containing representatives of one species. So, instead of evolving a population (global or spatially distributed) of similar individuals representing a global solution, it is more appropriate to coevolve subpopulations of individuals representing specific parts of the global solution.

A number of coevolutionary algorithms have been presented recently (see, e.g., [14]). The coevolutionary GA [10]) described in the context of the constraint satisfaction problem and the neural network optimization problem is a low level parallel EA based on a predator-prey paradigm. The algorithm operates on two subpopulations: the main subpopulation $P^1(\bar{x})$ containing individuals $\bar{x}$ representing some species, and an additional subpopulation $P^2(\bar{y})$ containing individuals $\bar{y}$ (another species) coding some constraints, conditions or test points concerning a solution $\bar{x}$. Both populations evolve in parallel to optimize a global function $f(\bar{x}, \bar{y})$.

The cooperative coevolutionary GA (CCGA) [11] has been described in the context of game-theoretic approach to optimization. These two coevolutionary algorithms are the subject of study presented in next sections. Another coevolutionary algorithm called coevolutionary distributed GA [7] was presented in the context of integrated manufacturing planning and scheduling problem. It combines features of diffusion model with coevolutionary concepts.

### 4 Test Functions

In the evolutionary computation literature (see, e.g., [8, 9, 16]) there is a number of test functions which are used as benchmarks for contemporary optimization algorithms. In this study we use some number of such functions, which will be the subject of minimization. We use the following test functions:

- sphere model: a continuous, convex, unimodal function
  $$f_1(x) = \sum_{i=1}^{n} x_i^2; \quad x \in \mathbb{R}^n,$$
  with $-100 \leq x_i \leq 100$, a minimum $x^* = (0, \ldots, 0)$ and $f_1(x^*) = 0$

- exponential function: a continuous, strictly convex, unimodal function
  $$f_{12}(x) = \sum_{i=1}^{n} 10^{x_i}; \quad x \in \mathbb{R}^n,$$
  with $-2.56 \leq x_i \leq 2.56$

- Rosenbrock’s function: a continuous, unimodal function
  $$f_2(x) = \sum_{i=1}^{n} \left(100 \left(x_i^2 - x_{i+1}\right)^2 + (1 - x_i)^2\right); \quad x \in \mathbb{R}^n,$$
  with $-2.12 \leq x_i \leq 2.12$, a global minimum $f_2(x^*) = 0$ at $x^* = (1, 1, \ldots, 1)$

- Rastrigin’s function: a continuous, multimodal function
  $$f_4(x) = \sum_{i=1}^{n} (x_i^2 - 10 \cdot \cos(2 \pi x_i)); \quad x \in \mathbb{R}^n,$$
  with $A = 10$, $\omega = 2 \cdot \pi - 5.12$, $x_i \leq 5.12$, $f_4(x^*) = 0$ in $x^* = (0, 0, \ldots, 0)$

- Schwefel’s function: a continuous, unimodal function
  $$f_3(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2; \quad x \in \mathbb{R}^n,$$
  with $-100 \leq x_i \leq 100, f_3(x^*) = 0$ in $x^* = (0, 0, \ldots, 0)$

- Ackley’s function: a continuous, multimodal function
\[ f_0(x) = -20e^{-0.2\sqrt{\sum_{i=1}^{n} x_i^2}} - e^{(4\sum_{i=1}^{n} \cos 2\pi x_i)} + 20 + e \]
\[ \text{with } -32 \leq x_i \leq 32, \quad f_0(x^*) = 0 \text{ in } x^* = (0, 0, \ldots, 0) \]

- Griewank' function: a continuous, multimodal function
\[ f_7(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos \left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad (7) \]
\[ \text{with } -600 \leq x_i \leq 600, f_7(x^*) = 0 \text{ in } x^* = (0, 0, \ldots, 0). \]

5 Competitive Coevolutionary Approach: Loosely Coupled Genetic Algorithms

5.1 Loosely Coupled Genetic Algorithms

Loosely coupled genetic algorithm (LCGA) [12, 13] is a medium-level parallel and distributed coevolutionary algorithm exploring a paradigm of competitive coevolution motivated by noncooperative models of game theory. Local chromosome structures of LCGA are defined for each variable, and local subpopulations are created for them. Contrary to described earlier in the paper parallel EAs, the LCGA is assumed to work in a distributed environment described by locally defined functions. A problem to be solved is first analyzed in terms of its possible decomposition and relations between subcomponents, expressed by a problem defined communication graph \( G_{com} \) called a graph of interaction.

In the case of functions like e.g. the Rosenbrock’s function a decomposition of the problem, designing local functions \( f^i_t(x_i, x_{i+1}) \), and a graph of interaction (a local function assigned to an agent associated with the node \( i \)) depends on a variable \( x_i \) associated with this node, and on the node \( (i+1) \) with associated variable \( x_{i+1} \) is straightforward (see, [2]).

Many real-life problems e.g. describing behavior of economic systems are naturally decentralized, or their models can be designed in such a way to decentralize their global criterion. When it is not possible, a communication graph \( G_{com} \) is a fully connected graph, and a global criterion becomes a local optimization criterion associated with each agent.

LCGA can be specified in the following way:

Step 1: for each agent-player create a subpopulation of his actions:

- create for each player an initial random subpopulation of size \( \text{sub pop size} \) of player actions with values from the set \( S_k \) of his actions.

Step 2: play a single game:

- in a discrete moment of time each player randomly selects one action from the set of actions predefined in his subpopulation and presents it to his neighbors in the game.
- calculate the output of each game: each player evaluates his local payoff \( u_k \) in the game.

Step 3: repeat step 2 until \( \text{sub pop size} \) games are played.

Step 4: for each player create a new subpopulation of his actions:

- after playing \( \text{sub pop size} \) games each player knows the value of his payoff received for a given action from his subpopulation.
- the payoffs are considered as values of a local fitness function defined during a given generation of a GA; standard GA operators of selection, crossover and mutation are applied locally to the subpopulations of actions; these actions will be used by players in the games played in the next game horizon.

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

Detailed description of the LCGA algorithm in OCCAM-like notation is presented below.

Algorithm: Loosely Coupled GA (LCGA)

\begin{itemize}
  \item \textbf{chromosome}: composition of local structures representing a partial solution \( x_i \in \mathbb{R} \)
  \item \textbf{optimization criterion}: composition of local functions \( f^i() \) or global function \( f() \)
  \item \textbf{population}: subpopulations \( P^i() \) \( i = 1, \ldots, N \) - a number of variables \( x_i \)
  \item \textbf{population structure}: communication graph \( G_{com} \) defined by a problem
  \item \textbf{t} = 0
  \item \textbf{PAR} \( i = 1 \) \textbf{FOR} \( N \)
  \item \textbf{SEQ}
  \begin{itemize}
    \item initialize \( P^i(t) \)
    \item set neighborhood of \( P^i(t) \) \( n_i \) - a number of neighbors of \( P^i(t) \) \( i = 1, n_i \)
  \end{itemize}
  \item \textbf{WHILE} \textbf{termination \_condition NOT TRUE}
  \item \textbf{PAR} \( i = 1 \) \textbf{FOR} \( N \) \hspace{1cm} -- competitive coevolution
  \item \textbf{SEQ}
  \begin{itemize}
    \item \textbf{PAR} \( k = 1 \) \textbf{FOR} \( n_{ind \_in \_subpop} \)
    \item assign to individuals \( l_k^i(t) \) \( n_i \) tags,
    \item \( tag(l_k^i(t)) = \)
    \item \textbf{random} < 1 \( n_{ind \_in \_subpop} \)
  \end{itemize}
  \item \textbf{PAR}
  \begin{itemize}
    \item send \( tag(l_k^i(t)) \) to neighborhood subpopulations
    \item receive tags from neighborhood subpopulations
  \end{itemize}
\end{itemize}
After initializing subpopulations, corresponding sequences of operations are performed in parallel for each subpopulation, and repeated in each generation. For each individual in a subpopulation a number of different individuals compete to maximize their local functions. The process of local maximization is constrained by a global criterion, usually as a sum of local function values in an equilibrium point. This global criterion is controlled by the LCGA and is achieved through the value of the global function.

A final performance of the LCGA operated in a distributed environment is evaluated by some global criterion. The process of local maximization is constrained by neighbor subpopulations, sharing the same variables. As the result of this competitive coevolution one can expect the system to achieve some equilibrium, equivalent to a Nash point equilibrium in noncooperative models of game theory.

A final performance of the LCGA operated in a distributed environment is evaluated by some global criterion, usually as a sum of local function values in an equilibrium point. This global criterion is typically unknown for subpopulations (except the case when \( G_{\text{com}} \) is a fully connected graph), which evolve with their local criteria. Achieving a maximal value of the global criterion by the LCGA may need in some applications (not applied in this study) a fitness sharing between neighbor subpopulations, and this is controlled by the exchange process parameter.

Let us consider the problem of minimizing the Rosenbrock’s function \( f_2(x) \) using LCGA. The problem can be viewed as a problem of seeking a minimum in a distributed way by a multi-agent system with a game-theoretic model of interaction among agents. Each agent \( A_i \) has only information about its locally defined function

\[
    f_2(x_i, x_{i+1}) = 100 \left( (x_i^2 - x_{i+1})^2 + (1 - x_i)^2 \right),
\]

which depends only on his \( x_i \) and \( x_{i+1} \) of his neighbor \( i+1 \). This local function is minimized by \( A_i \) during a game, while the performance of the game is observed through the value of the global criterion (3).

Fig. 1 shows a computational scheme used to find a minimum of \( f_2(x) \) by a distributed optimization algorithm based on LCGAs. Individuals in a subpopulation corresponding to a given agent \( i \) \((i = 1, 2, \ldots, n - 1)\) code solutions for a variable \( x_i \). The agent \( n \) runs two subpopulations corresponding to variables \( x_n \) and \( x_{n+1} \) respectively. Individuals in subpopulations are evaluated according to their local functions. Local subpopulations containing current values of \( x_i \) are modified by locally applied GA, without any knowledge of the global function. Only an observer of the distributed optimization algorithm knows the global state of the optimization process.

6 Cooperative Coevolutionary Approach: Cooperative Coevolutionary Genetic Algorithm

Cooperative coevolutionary genetic algorithm (CCGA) has been proposed [11] in the context of a function optimization problem. Each of \( N \) variables \( x_i \) of the optimization problem is considered as a species with its own chromosome structure, and subpopulations for each variable are created. A global function \( f(\vec{x}) \) is an optimization criterion. To evaluate the fitness of a variable from a given subpopulation, it is necessary to communicate with selected individuals from all subpopulations. Therefore, the communication graph \( G_{\text{com}} \) is fully connected.

In the initial generation of CCGA individuals from a given subpopulation are matched with randomly chosen individuals from all other subpopulations. A fitness of each individual is evaluated, and the best individual \( t_{\text{max}} \) in each subpopulation is found. The process of cooperative coevolution starts from the next generation. For this purpose, in each generation the following cycle consisting of two phases is repeated in a round-robin fashion. In the first phase only one current subpopulation is active in a cycle, while the other subpopulations are frozen. All individuals from an active subpopulation are matched with the best individuals from frozen subpopulations. A new better individual is found this way for each active subpopulation. In the second phase the best found individual from each subpopulation is matched with a single, randomly selected individual from other subpopulations. A winner individual is a better individual from these two phases. When the evolutionary process is completed a composition of the best individuals from each subpopulation represents a solution of a problem.
7 LCGA and CCGA: Experimental Study

Both LCGA and CCGA algorithms were tested on the set of functions presented in Section 4. The experiments were conducted on a sequential PC machine. Results of these experiments were compared with the results of a sequential GA. In all experiments the accuracy of \( x_i \) was not worse than \( 10^{-6} \). Experiments were conducted with number of variables \( n = 5, 10, 20, 30 \), but only results for \( n = 30 \) are reported in the paper. All algorithms run 200 generations.

The purpose of the first set of experiments (not shown in the paper) was to tune parameters of the algorithms. The subject of tuning were such parameters as a size of subpopulations, a type of a selection (proportional, tournament, ranking), probabilities of crossover \( p_c \) and mutation \( p_m \). Each experiment was repeated 25 times, and 20 of the best of them was averaged and used to judge about optimal setting parameters for functions.

The following parameters were finally set for LCGA: the size of subpopulation corresponding to given species was equal \( 100 \) (sizes of subpopulations equal to 20 for sphere model, exponential and Schwefel’s functions, and 50 for Rastrigin’s and Griewank’s functions were fine), \( p_m \) ranged for different functions from 0.001 (Rosenbrock’s, Rastrigin’s function) to 0.005, \( p_c \) ranged from 0.5 (Griewank’s function), 0.6 (Rosenbrock’s function) to 0.9. Ranking selection for Rastrigin’s and proportional selection for Ackley’s function was used. For remaining functions a tournament selection with a size of tournament equal to 4 was used.

The following parameters were set for CCGA: the size of subpopulation corresponding to given species was equal \( 100 \) (sizes of subpopulations equal to 20 for sphere model, exponential and Schwefel’s function, and 50 for Ackley’s function were fine), \( p_m \) ranged for different functions from 0.001 (Rosenbrock’s, Rastrigin’s function) to 0.005, \( p_c \) ranged from 0.5 (Griewank’s function), 0.6 (Rosenbrock’s function) to 1.0. Proportional selection for Rosenbrock’s and Rastrigin’s function was used and ranking selection for Griewank’s function was used. For remaining functions tournament selection with a size of tournament equal to 4 was used.

The following parameters were set for the sequential GA: the size of a global population corresponding to given species was equal \( 100 \), \( p_m \) ranged for different functions from 0.001 (Ackley’s function) to 0.0095, \( p_c \) ranged from 0.5 (Griewank’s function) to 0.95. Ranking selection for Rastrigin’s and Griewank’s function was used and tournament selection with a size of tournament equal to 3 was used for remaining functions.

Fig. 2, 3 and 4 show results of experiments conducted with use of LCGA, CCGA and the sequential GA applied to minimize test functions presented in Section 4. Each experiment was repeated again 25 times. The best results in each generations of 20 the best experiments were averaged and accepted as results shown in following figures as a function of a number of generations.

One can easily notice that both coevolutionary algorithms LCGA and CCGA are better than the sequential GA in the problem minimization of function for all test functions used in the experiment. However, comparison of coevolutionary algorithms is not so straightforward. For the test problem corresponding to the sphere model both coevolutionary algorithms present almost the same behavior (speed of convergence and the average of minimal values) for the number of variables from the range \( n = 5 \) to \( n = 20 \) (not shown in the paper), with some better performance of LCGA.

For the problem represented by exponential function performance of CCGA is slightly better than the performance of the sequential GA for \( n = 5 \) (not shown) and improves with increase of \( n \). Performance of LCGA is distinctively better than the performance of CCGA for all \( n \). Fig. 2b shows this tendency for \( n = 30 \). One can see that LCGA outperforms CCGA achieving its minimum after about 150 generations. The problem however seems to be more difficult that one represented by sphere function.

For the Rosenbrock’s function CCGA shows slightly better performance that LCGA for all \( n \), and this situation is shown in Fig. 2c for \( n = 30 \). One can see that this test function is really difficult for both algorithms. The opposite situation takes place for the Rastrigin’s function. LCGA is distinctively better than CCGA for small values of \( n \) (\( n = 5 \)) and slightly better for greater values of \( n \). Fig. 3a illustrates this situation for \( n = 30 \).

For the Schwefel’s function and \( n = 5 \) all three algorithms achieve the same minimal value. CCGA needs for this about 60 generations, LCGA needs about 110 generations and the sequential GA needs 200 generations. For \( n = 10 \) both coevolutionary algorithms are better than the sequential GA, but CCGA is slightly better than LCGA. For greater values of \( n \) CCGA outperforms LCGA (see, Fig. 3b).

For the Ackley’s function CCGA outperforms LCGA for all values of \( n \). For values of \( n = 5, 10 \) LCGA is worse than the sequential GA, and for \( n = 30 \) LCGA and the sequential GA show similar performance (see, Fig. 3c). For the Griewank’s function situation is similar. CCGA outperforms LCGA for all values of \( n \) (see, Fig. 4a). For values of \( n = 5, 10, 20 \) performance of LCGA and sequential GA is similar.

8 Discussion and Conclusions

Results of ongoing research on the development of parallel and distributed evolutionary algorithms for function optimization have been presented in the paper. Coevolution - a new very promising paradigm in evolutionary computation has been chosen as an engine for effective parallel and distributed computation. Two coevolutionary algorithms based on different phenomena known as competition and cooperation were studied. These are respectively loosely coupled genetic algorithms (LCGA) and cooperative coevolutionary genetic algorithms (CCGA).

LCGA presents fully parallel and distributed coevolutionary algorithm in which subpopulations called also agents, using game-theoretic mechanism of competition, act to maximize their local goals described by some local functions. The competition between agents leads to establishing some equilibrium in which local goals cannot be more improved, and at the same time some global goal (typically as a sum of local functions) of the system is also achieved. The global state of the system (a value of a global goal) is not directly calculated, but is rather observed. To achieve this global goal no coordination of agents is required.
Figure 2. Exp. 1: Sphere model (a), exponential function (b), Rosenbrock’s function (c)

Figure 3. Exp. 1: Rastrigin’s function (a), Schwefel’s function (b), and Ackley’s function (c)
CCGA is partially parallel and centralized coevolutionary algorithm in which subpopulations cooperate to achieve a global goal. In a given moment of time only one subpopulation is active while the other subpopulations are frozen. The global goal of the system is at the same time a goal of each subpopulation. To evaluate a global goal a coordination center needs to communicate with each subpopulation to know a current local solution.

Results of experiments have shown that the LCGA is a very effective optimization algorithm for problems where the global goal of the system is the sum of local goals. For such unimodal problems (sphere model, exponential function) LCGA clearly shows its ability of fast (a number of generations) parallel and distributed optimization at low computational cost (no need for communication between agents to collect values of local functions and calculate a value of a global function) and high quality of solution, better than offered by CCGA. LCGA possesses such a ability also for highly multimodal functions (Rastrigin’s function) expressed as a sum of local functions. However for problems expressed in more complex way (Schwefel’s, Ackley’s and Griewank’s functions) CCGA with cooperation mechanism and a global coordination shows better performance than LCGA.

What we have learned from the study on coevolutionary algorithms applied to problem solving with use of parallel and distributed computing shows is that a success of a given computational scheme primary depends on the way of formulation of a global criterion. We believe that results presented in the paper will stimulate developing new effective parallel and distributed computational technique for problem solving and applying them in dynamically changing multi-agent environments. In the forthcoming paper we will show that the performance of LCGA can be dramatically improved by introducing new local genetic operators.

References