# Chapter 10
# A New Approach to Network Optimization Using Chaos-Genetic Algorithm

Golnar Gharooni-fard and Fahime Moein-darbari

**Abstract.** Genetic Algorithms (GAs) have been widely used to solve network optimization problems with varying degrees of success. Part of the problem with GAs lies in the premature convergence when dealing with large-scale and complex problems; Caught in local optima, the algorithm might fail to reach the global optimum even after a large number of iterations. In order to overcome the problems with traditional GAs, a method is proposed to integrate Chaos Optimization Algorithms (COAs) with GA to fully exploit their respective searching advantages. The basic idea of COA is to transform the problem variables, by way of a map, from the solution space to a chaos space and to perform a search that benefits from the randomness, orderliness and ergodicity of chaos variable. In this chapter, we will first discuss network optimization in general, and then focus on how chaos theory can be incorporated into the GA in order to enhance its optimization capacities. We will also examine the efficiency of the proposed Chaos-Genetic algorithm in the context of two different types of network optimization problems, Grid scheduling and Network-on-Chip mapping problem.

**Keywords:** network optimization, Genetic Algorithm, Chaos theory, Grid scheduling, Network-on-Chip mapping problem.

## 10.1 Introduction

Network theory basically deals with problems that have a graph structure. Graphs are mathematical structures used to model pair wise relations between objects. They consist of points, and lines connecting pairs of points. The points are called

Golnar Gharooni-fard · Fahime Moein-darbari
Computer Department of Islamic Azad University,
Mashhad Branch, Young Researchers Club, Iran
e-mail: golnar.ghf@gmail.com, fahime.md61@gmail.com

nodes or vertices and the lines are called arcs. The arcs may have a direction on them, in which case they are called directed arcs. If an arc has no direction, it is often called an edge. If all the arcs in a graph are directed, the graph is said to be directed (digraph). Graphs are among the most ubiquitous models of both natural and human-made structures. They can be used to model many types of relations and process dynamics in physical, biological and social systems. Many problems of practical interest can be represented by graphs [1]. In computer science, graphs are used to represent networks of communication, data organization, computational devices, the flow of computation, etc. Fig. 10.1 is an example of a network modeled with graphs. At any given time, a message may take a certain amount of time to traverse each line (due to congestion effects, switching delays, etc.). The expended time can vary greatly and telecommunication companies dedicate a significant amount of their resources tracking these delays. Assuming a centralized switcher knows these delays, there remains the problem of routing a call so as to minimize the delays. This is an example of a particular type of network model, called the *shortest path* which includes a network with weighted edges and two special nodes: a source and a destination. The goal is to find a path from the source to the destination with the minimum total weight.

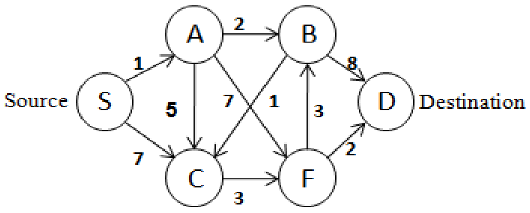

**Fig. 10.1** A Phone network modeled by a graph

Network problems that involve finding the least-cost solution to a problem where each solution is associated with a numerical cost are generally studied under combinatorial optimization which concerns the efficient allocation of limited resources to meet desired objectives when the values of some or all of the variables are restricted to be integral [2]. Still, in most such problems, there are many possible alternatives to consider and one overall goal determines which of these alternatives is best.

Different approaches have been used to solve network optimization problems [3] among which are a large family of algorithms collectively labeled metaheuristics. A metaheuristic designates a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Metaheuristics make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions [4], [5]. Metaheuristics can be used for the purpose of combinatorial optimization where an optimal solution is sought over a discrete search-space. Popular metaheuristics for combinatorial problems include Simulated Annealing (SA) [6], Genetic Algorithm (GA) [7]' Particle Swarm Optimization (PSO) [8], Ant Colony Optimization

(ACO) [9] and Tabu Search (TS) [10]. Since our focus here is on GAs, in the next section we will discuss them as one of the most popular metaheuristics used for optimization purposes. The interested reader is referred to [11] and [12] for more general surveys on the metaheuristics.

## 10.2 Genetic Algorithms

Genetic algorithms are inspired by the evolutionary theory of the origin of species which explains how weak and unfit species in nature face extinction by way of natural selection. Natural selection is the process by which traits become more or less common in a population due to consistent effects upon the survival or reproduction of their bearers, the strong species. In the long run, species carrying the correct combination in their genes become dominant in their population. Sometimes, during the slow process of evolution, random changes may occur in the genes. If these changes provide additional advantages in the challenge for survival, new species evolve from the old ones. Unsuccessful changes are eliminated by natural selection.

The concept of Genetic Algorithms (GAs) was introduced by John Holland in the early seventies as a special technique for function optimization [7]. In GA terminology, a solution vector is called an individual or a *chromosome*. Chromosomes are made of discrete units called *genes*. Each gene controls one or more features of the chromosome. In the original implementation of GA by Holland, genes are assumed to be binary numbers. In later implementations, more varied gene types have been introduced. Normally, a chromosome corresponds to a unique solution in the solution space. The GA operates with a collection of chromosomes, called a *population*. The population is normally randomly initialized. As the search goes on, populations evolve to include fitter and fitter solutions, and eventually converge, to a single solution.

The basic idea of a GA is that the genetic pool of a given population potentially contains the best solution, to a given adaptive problem, although this solution might not have been realized yet. The algorithm operates in an iterative manner and evolves a new generation from the current generation by applying genetic operators [13]. Given a clearly defined problem to be solved and strings of candidate solutions, a simple GA works as follows:

1. Initialize the population.
2. Calculate the fitness value for each individual in the population.
3. Reproduce selected individuals to form a new population.
4. Perform crossover and mutation on the population.
5. Loop to step 2 until some termination condition is met.

In some GA implementations, operations other than crossover and mutation are carried out in step 4. Crossover is considered by many to be an essential operation of all GAs. It plays an important role in distributing the individuals over the space of interest through the GA. Termination of the algorithm is usually based either on achieving a population member with some specified fitness or on running the algorithm for a given number of generations. Like many other metaheuristics, GAs do not guarantee an optimal solution is ever found. They often show a very fast initial convergence followed by progressive slower improvement. Therefore different techniques have been used to improve the results obtained from the GAs [14]. By introducing Chaos theory in the next section, we will explain how to integrate this concept with GA, in order to enhance the quality of the solutions.

## 10.3 Chaos Theory

Chaos theory is the study of the behavior of dynamical systems that are highly sensitive to initial conditions. In common usage, "chaos" means “a state of disorder”, but the adjective "chaotic" is defined more precisely in chaos theory. Although there is no universally accepted mathematical definition of chaos, a commonly used definition describes, chaos as a non-periodic, long-term behavior in a deterministic system that exhibits sensitive dependence on initial conditions [15]. None-periodic long-term behavior means that the system's trajectory in phase space does not settle down to any fixed points or periodic orbits, as time tends to infinity. Deterministic systems can have no random (or probabilitistic) parameters. It is a common misconception that chaotic systems are noisy systems driven by random processes. The irregular behavior of chaotic systems arises from intrinsic nonlinearities rather than noise. Sensitive dependence on initial conditions, the proverbial “the butterfly effect”, requires that trajectories originating from nearly identical initial conditions diverge exponentially. Despite what the name suggests, chaos is not the absence of order; it is a subtle state that is poised between order and randomness, with both aspects intermingled.

If a chaotic system’s behavior is plotted in a graph over an extended period, obscure patterns might emerge. When a bounded chaotic system does have some long term pattern, but not a simple periodic oscillation or orbit, it is said to have a strange attractor [16]. In other words, strange attractor is the natural shape of chaos. It is called strange because of its complex geometry, and it is an attractor because the system that it describes is always drawn to the behavior that it represents as if attracted to it. The mathematical model developed, called the “Lorenz system[1] has been used as a paradigm for chaotic systems that satisfy the above definition. The Lorenz system consists of three first-order coupled differential equations as follows

$$\begin{cases} \frac{dx}{dt} = \sigma(y-x) \\ \frac{dy}{dt} = x(\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases} \tag{10.1}$$

[1] The “Lorenz system” is named after the American meteorologist Edward N. Lorenz, who in 1963 discovered chaotic behavior in a computer study of weather.

where all σ, ρ, β > 0, but usually σ = 10, β = 8/3 and ρ is varied. The system exhibits chaotic behavior when ρ = 28 [15]. The Lorenz system has three dynamic variables, and consequently the state-space picture of such a system is three-dimensional. Plotting the trajectory of the Lorenz system in state space, shown in Fig. 10.2, reveals what was earlier defined as a strange attractor (the Lorenz chaotic attractor). The map shows how the state of a dynamical system (the three variables of a three-dimensional system) evolves over time in a complex, non-repeating pattern.

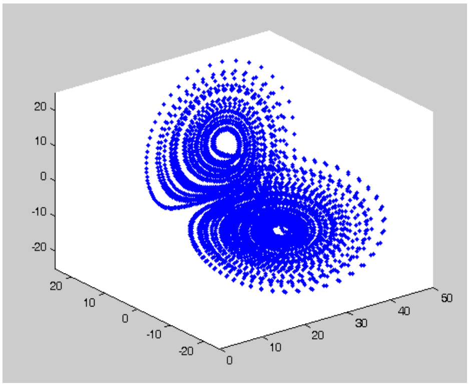

**Fig. 10.2** The Lorenz attractor

One-dimensional noninvertible maps are the simplest systems capable of generating chaotic motion. As such, they serve as a convenient starting point for the study of chaos [17]. Here, we introduce some well known one-dimensional maps.

**Logistic Map.** The logistic map proposed by Robert May is a polynomial map and is often cited as an example of how complex behavior can arise from a very simple nonlinear dynamical equation [15]. This map is defined as

$$x_{n+1} = f(\mu, x_n) = \mu x_n(1 - x_n) \ , \ 0 < \mu \leq 4 \tag{10.2}$$

where $\mu$ is a control parameter, and $x$ is a variable. Since the equation represents a deterministic dynamic system, it might seem like its long-term behavior can be predicted, but that is in fact not the case since its behavior is heavily dependent on the variations of $\mu$. The value of the control parameter, determines whether $x$ converges to a constant point, oscillates between two or more values, or behaves chaotically in an unpredictable pattern [18].

**Tent Map.** In mathematics, the tent map is an iterated function, in the shape of a tent, forming a discrete-time dynamical system. It takes a point $x_n$ on the real line and maps it to another point as

$$x_{n+1} = \begin{cases} \mu x_n \quad , & x_n < \frac{1}{2} \\ \mu(1 - x_n) \ , & \frac{1}{2} \le x_n \end{cases} \tag{10.3}$$

where μ is a positive real constant [19]. The tent map and the logistic map are topologically conjugate and thus their behavior under iteration is identical in this sense. Depending on the value of $\mu$, the tent map demonstrates a range of dynamical behavior ranging from predictable to chaotic.

**Bernoulli Shift Map.** The Bernoulli shift map belongs to a class of piecewise linear maps which consist of a number of piecewise linear segments. This map is a particularly simple, consisting of two linear segments to model the active and passive states of the source [20]. It is defined as follows

$$x_{n+1} = \begin{cases} \frac{x_n}{(1-\lambda)} \quad , & 0 < x_n < (1 - \lambda) \\ \frac{x_n-(1-\lambda)}{\lambda} \ , & (d \equiv 1 - \lambda) < x_n < 1 \end{cases} \tag{10.4}$$

**Sine Map.** The sine map is described by the following equation

$$x_{n+1} = \frac{a}{4}\sin(\pi x_n) \tag{10.5}$$

where $0 < a \le 4$ . Qualitatively this map has the same shape as the logistic map.

**ICMIC Map.** The iterative chaotic map with infinite collapses (ICMIC) has infinite fixed points in comparison with finite collapses one-dimensional maps [21], [22]. The ICMIC map is described by following equation

$$x_{n+1} = \sin\frac{a}{x_n} \tag{10.6}$$

where $a \in (0, \infty)$ is an adjustable parameter.

## 10.4 Chaos Optimization Algorithm (COA)

In random-based optimization algorithms, the methods using chaotic variables instead of random variables are called Chaotic Optimization Algorithm (COA) [23], [24]. Originally proposed by Li and Jiang, COA searches the solution space based on the regularity of chaotic variables and more easily escapes local minima compared with stochastic optimization algorithm [25]. By means of ergodicity, regularity and semi-stochastic properties of chaos, the optimal solution migrates in a chaotic way among the local minima and finally converges to the global optimal solution [26]. Experimental studies assert that the benefits of using chaotic signals instead of random signals are often evident although it is not mathematically proved yet [27]. The procedure of COA is demonstrated as followings:

1. Set k = 0 and $f(x_i^k)$ as a random solution in the problem domain and a chaotic variable $0 < r_i < 1,\ (i = 1,2,\cdots,n)$.
2. Map the chaotic sequences $r_i^k$ to $x^*$ according to the characteristics of the particular problem.
3. Compare the function value of $f(x^*)$ with $f(x_i^k)$, pick the better value and replace it with $f(x_i^k)$. Then replace $x^*$ with $x_i^k$.
4. Apply one of the aforementioned chaotic equations (denoted by *M*)

$$r_i^{k+1} = M(r_i^k) \tag{10.7}$$

   Note that the interval of chaotic sequences is between 0 and 1.
5. Set $k = k + 1$ and loop back to step 2 until the termination condition is reached.

Numerical results show that COA takes less iteration to reach to an optimum solution than most global optimization methods [25]. However, COA has the deficiency of taking much time to get to the optimum value, which affects the speed of convergence [28]. To overcome this limitation, an improved chaos optimization method that combines COA and GA is presented in the next section.

### *10.4.1 Chaos-Genetic Algorithm (CGA)*

The idea of using chaotic systems instead of random processes has recently been noticed in several fields, including optimization theory. The basic idea is to transform the variables of a problem from the solution space to chaos space and then perform a search to find a solution by virtue of the randomness, orderliness and ergodicity of the chaos variable. Although the COA has many advantages, it makes no use of the experiential information previously acquired [29]. Furthermore, in GAs there is no guaranteed convergence even to a local minimum [30]. Since the genes from a few highly fit (but not optimal) individuals may rapidly come to dominate the population, causing it to converge on local minima and once the population has converged, the ability of the GA to continue to search for better solutions is largely compromised.

In order to overcome the shortcomings of both COA and GA, one option is to integrate the two in order to bring together the searching advantages of both algorithms. The concept of Chaos-genetic algorithms (CGA), first introduced in [30], has the following characteristics: Firstly, CGA benefits from the characteristics of the chaotic variables to make the individuals of subgenerations distributed ergodically in the defined space and thus to avoid premature convergence in the subgenerations. Secondly, according to its evolutionary nature, CGA maintains the fittest individuals in each run and hence increases the probability of finding the global optimal solution. CGA can be implemented by simply adding a chaotic mapping operator to the standard GA operators, namely crossover and mutation.

As an example of a chaotic equation, the logistic map has been extensively analyzed in the past decade. The evolution of the chaotic variables could be defined through the following equation [31],

$$r_i^{k+1} = 4r_i^k\left(1 - r_i^k\right), \qquad i = 1, 2, \dots, n. \tag{10.8}$$

In principle, this is the same as the equation introduced for logistic map in Section 10.3. The value of the parameter $\mu = 4$ is chosen in order for the system to act chaotically. Here $r_i$ is the $i$-th chaotic variable and $k$ denotes the number of iterations. The value of $r_i$ , is distributed in the range of $[0, 1]$ and $n$ denotes the number of genes in each chromosome. In order to perform the chaotic mapping, the following procedure is proposed.

1. Divide the interval $[0, 1]$ to $n$ equal sub-intervals, of which the lower limit $[a_1, a_2, \dots, a_n]$ is represented by vector $a$, and the upper limit $[b_1, b_2, \dots, b_n]$ by vector $b$.
2. The real value of each $x_i$ in the first randomly produced population is linearly mapped to new values of $1 < r_i < 0$, using

$$r_i = \frac{1}{b_i - a_i}\,(x_i - a_i). \tag{10.9}$$

3. The next iteration chaotic variables $r_i^{(2)}$, will be produced through applying the logistic map equation to $r_i^{(1)}$ values, generated in the previous section.
4. The chaotic variables $r_i^{(2)}$, are then used to produce $x_i^{(2)}$, using

$$x_i^{(2)} = a_i + r_i^{(2)}(b_i - a_i), \; i = 1, 2, \dots, n. \tag{10.10}$$

We can repeat the process in order to produce the next values of $x_i^{(k)}$. Although chaos variables are usually generated by the logistic map, there's no reason not to try any of the previously defined one-dimensional maps in order to form a chaotic mapping operator. Fig. 10.3 demonstrates a flowchart of the overall process of Chaos-genetic algorithm using the logistic map as a chaotic mapping operator to produce the chaotic population $P_2$ from the randomly produced initial population $P_1$. In the next section we will examine the performance of CGA in two types of network optimization problems, namely Grid scheduling and Network-on-Chip mapping problem.

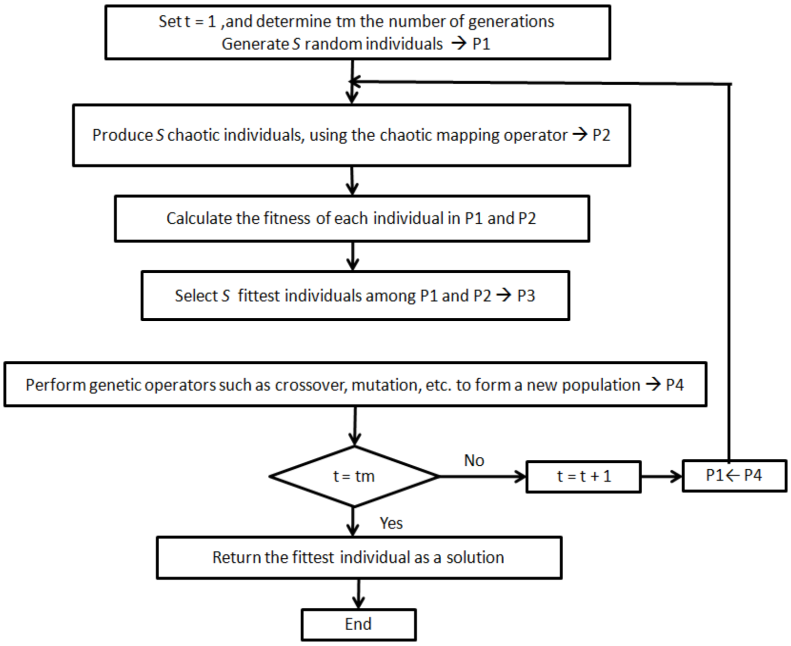
Set t = 1 ,and determine tm the number of generations
Generate S random individuals → P1
Produce S chaotic individuals, using the chaotic mapping operator → P2
Calculate the fitness of each individual in P1 and P2
Select S fittest individuals among P1 and P2 → P3
Perform genetic operators such as crossover, mutation, etc. to form a new population → P4
t = tm
No
t = t + 1
P1← P4
Yes
Return the fittest individual as a solution
End

**Fig. 10.3** Chaos-Genetic Algorithm procedure

## 10.5 Grid Scheduling: Case Study # 1

A grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [30]. It is a shared environment, implemented via the deployment of a persistent, standards-based service infrastructure that supports the creation and sharing of the resource within distributed communities. The resources might be computers, storage space, instruments, software applications, and data, all connected through the Internet and a middleware software layer that provides basic services for security, monitoring, resource management, and etc. Resources owned by various administrative organizations are shared under locally defined policies that specify what is shared, who is allowed to access what, and under what conditions [32].

From the point of view of scheduling systems, a higher level abstraction for the Grid can be applied by ignoring some infrastructure components such as authentication, authorization, resource discovery and access control. Thus, the following definition for the term *Grid* is adopted in our study: “A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements” [33].

To facilitate the discussion on grid scheduling, we need to define some frequently used terms; *tasks* are atomic units to be scheduled by the scheduler and assigned to resources. The *properties* of a task are parameters like CPU/memory requirement, deadline, priority, etc. A *job* (*metatask* or *application*) is a set of atomic tasks that will be carried out on a set of resources. *Resources* are required to carry out an operation, for example: a processor for data processing, a data storage device, or a network link for data transporting. A *site* (or *node*) is an autonomous entity composed of one or multiple resources.

Based on the definitions above, *task scheduling* can be defined as the mapping of tasks to a selected group of resources which may be distributed in multiple administrative domains. Although, a grid is a system of high diversity, which is rendered by various applications, middleware components, and resources, we can still find a logical architecture of the task scheduling subsystem in the grid that as noted by Schopf in [34], can be generalized into three stages:

1. Resource discovering and filtering,
2. Resource selecting and scheduling according to certain objectives,
3. Job submission.

Since the study of scheduling algorithms is our primary concern, we mainly focus on the second step. Scheduling of interdependent tasks in distributed heterogeneous computing environments is well known to be an NP-hard problem [35]. Several heuristic algorithms have been applied to solve the scheduling problem. These can be classified into two major groups, in view of their main objectives. First, a group of works that only attempt to minimize workflow execution time, without considering user's budget. *Min-Min*, which sets the highest priority to tasks with the shortest execution time, and *Max-Min*, which sets the high priority to the tasks with the long execution times are two major heuristic algorithms employed for scheduling workflows on grids [36]. *Sufferage*, is another heuristic algorithm which sets high scheduling priority to tasks whose completion time by the second best resource is far from that of the best resource [36]. Another workflow scheduling algorithm developed by the authors of [37], is based on a Greedy Randomized Adaptive Search Procedure (*GRASP*). Another workflow level heuristic is a Heterogeneous-Earliest-Finish-Time (*HEFT*) algorithm proposed by Wieczorek et al. [38]. Second, a group of works which address scheduling problems based on user's budget constraints. Nimrod-G [39] schedules independent tasks for parameter-sweep applications to meet user's budget. More recently, *LOSS* and *GAIN* scheduling approaches were developed, to adjust a schedule which is generated by a time-optimized heuristic and cost optimized heuristic to meet the user's budget constraints [40].

## 10.5.1 Challenges of Scheduling Algorithms in Grid Computing

Although previous research in this area is of great value, traditional scheduling models generally produce poor grid schedules in practice [32]. To remedy this let us go through the assumptions underlying traditional systems:

- All resources reside within a single administrative domain.
- To provide a single system image, the scheduler controls all of the resources.
- The resource pool is invariant.
- Contention caused by incoming applications can be managed by the scheduler according to some policies, so that its impact on the performance that the site can provide to each application can be well predicted.
- Computations and their data reside in the same site.

Unfortunately, not all of these assumptions hold in grid circumstances. There are unique characteristics in grid computing, listed by the authors of [41], which make the design of scheduling algorithms more challenging:

• *Heterogeneity and Autonomy.* In grid computing, because resources are distributed in multiple domains on the Internet, heterogeneity is a characteristic not only of computational and storage nodes but also of the underlying networks connecting them. This results in different capabilities for job processing and data access. The autonomy also gives way to aa diverse array of local resource management techniques and access control policies, such as, priority settings for different applications and resource reservation methods. Thus, a grid scheduler is required to be adaptive to different local policies. The heterogeneity and autonomy on the grid user side are represented by various parameters, including application types, resource requirements, performance models, and optimization objectives.

• *Performance Dynamism.* Making a feasible scheduling usually depends on the performance estimate that candidate resources can provide, especially when the algorithms are static. Grid schedulers work in a dynamic environment where performance of available resources is constantly changing. The change comes from site autonomy and competition for resources by various applications.

• *Resource Selection and Computation.-Data Separation* In traditional systems, executable codes of applications and input/output data are usually in the same site, or the input sources and output destinations are determined before the application is submitted. Thus the cost for data staging can either be neglected or is a constant determined before execution, and scheduling algorithms need not consider it. But in a grid which consists of a large number of heterogeneous computing sites (from supercomputers to desktops) and storage sites connected via wide area networks, the computation sites of an application are usually selected by the grid scheduler according to resource status and certain performance models. Additionally, in a grid, the communication bandwidth of the underlying network is limited and shared by a host of background loads, so the inter-domain communication cost cannot be neglected.

Many grid applications are data intensive, so the data staging cost is considerable. This situation brings about the computation-data separation problem: the advantage brought by selecting a computational resource that can provide low computational cost may be neutralized by its high access cost to the storage site. These challenges depict unique characteristics of grid computing, and put significant obstacles to design and implement efficient and effective grid scheduling systems. It is believed, however, that research achievements on traditional scheduling problems can still provide stepping-stones for a new generation of scheduling systems.

In order to introduce the Chaos-genetic algorithm to solve the workflow scheduling problem, we need to define an appropriate problem representation, fitness assignment, and genetic operators. These will be discussed in the following subsections.

### 10.5.2 Problem Description

As mentioned in the previous section, the scheduling problem becomes more challenging because of some unique characteristics of grid computing. The grid scheduling problem can be defined as follows: A workflow application can be modeled as a Directed Acyclic Graph (DAG). There is a finite set of tasks $T_i$ $( i = 1,2, \ldots, n)$ and a set of directed arcs of the form $( T_i ,T_j )$, where $T_i$ is the parent task of $T_j$ , and $T_j$ is the child of $T_i$. A child task can never be executed unless all of its parent tasks have been completed. Let $B$ be the cost constraint (budget) and $D$ the time constraint (deadline), specified by the user's workflow execution. The total number of available services is shown by $m$. There's a set of services $S_j$ $( j = 1,2, \ldots, m)$ capable of executing task $T_i$, but each task can only be assigned for execution to one of these services. Services have varied processing capabilities delivered at different prices. We denote $t_i^j$ as the processing time, and $c_i^j$ as the service price for processing $T_i$ on service $S_j$. The scheduling problem is to map every $T_i$ onto a suitable $S_j$ in order to get the best trade-off between execution time and cost in a workflow considering the user's budget and deadline.

### 10.5.3 The Chaos-Genetic Scheduling Algorithm (CGA)

For a workflow scheduling problem, a feasible solution is required to meet several conditions:

1. A task can only be started after all its predecessors have completed.
2. Every task appears once and only once in the schedule.
3. Each task must be allocated to one available time slot of a service capable of executing the task.

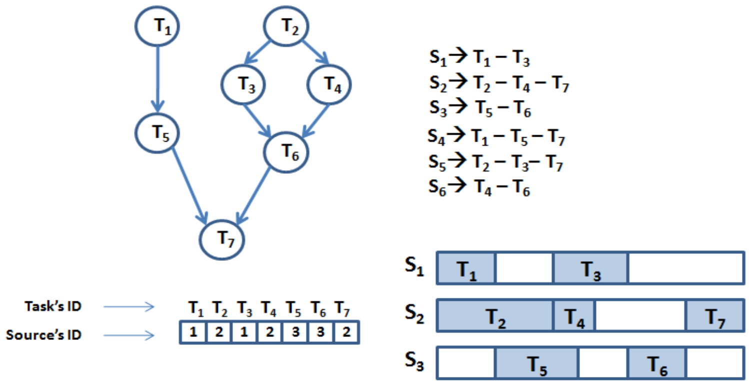

**Fig. 10.4** A sample workflow followed by a set of source-to-task assignments

Each individual in the population represents a feasible solution to the problem, and consists of a vector of task assignments. Each task assignment includes four elements (task ID, service ID, start time, end time) [42]. The first two parameters identify to which service each task is assigned. Since involving time frames during the genetic operation may lead to a very complicated situation [43], we ignore the time frames here. Therefore, the operation strings (chromosomes) encode only the service allocation for each task and the order of the tasks allocated to each service. Different execution priorities of such parallel tasks within the workflow may impact the performance of workflow execution significantly. For this reason, the solution representation strings are required to show the order of task assignments on each service in addition to service allocation of each task. As suggested by Buyya [43], we create an array to represent a schedule as illustrated in Fig.10.4. Each element of this array represents a service and the indexes refer to the task number.

As stated earlier, the problem is to schedule a workflow execution considering both time and user budget constraints. The first decision to be made is how to represent the solution, which was shown in Fig.10.4. Initializing the population is done randomly using a random generator to produce values between 1 to $n$. For each task, these random values are chosen from sources that are capable of executing that task. The length of the chromosome depends on the number of tasks in the workflow. A chaotic mapping operator is then applied to the initial population, generating a new chaotic population.

At this stage, the fitness of the individuals of the entire population is evaluated. The fitness value is often proportional to the output value of the function being optimized according to the given objectives. As the goal of scheduling is to get the best trade-off between the time and cost of the workflow execution, the fitness function divides the evaluation into two parts [43]: cost-fitness and time-fitness. For budget constrained scheduling, the cost-fitness component produces results with less cost. The cost fitness function of an individual $I$ is defined by

$$F_{cost}(I) = \frac{c(I)}{B} \tag{10.11}$$

where *c(I)* is the sum of the task execution cost and data transmission cost of *I* and *B* is the budget of the workflow. For budget constrained scheduling, the time-fitness component is designed to produce individuals that satisfy the deadline constraint. The time-fitness function of an individual *I* is defined by

$$F_{time}(I) = \frac{t(I)}{D} \tag{10.12}$$

where *t(I)* is the completion time of *I*, *D* is the deadline of the workflow. The final fitness function combines the two parts and it is expressed as:

$$F(I) = \begin{cases} F_{cost}(I) + F_{time}(I), if\ F_{cost}(I) > 1\ or\ F_{time}(I) > 1 \\ \frac{c(I)}{maxcost} \times \frac{t(I)}{maxtime} \qquad otherwise \end{cases} \tag{10.13}$$

where *maxcost* is the most expensive solution of the current population and *maxtime* denotes the largest completion time in the current population.

Elitism is incorporated into the algorithm by transferring the single fittest individual directly to the next generation. Crossover is used to create new solutions by rearranging parts of the existing solutions in the current population. The idea behind the crossover operation is that a higher quality solution may result from the combination of two of the current fittest solutions [44]. We have implemented a two-point crossover which is illustrated in Fig. 10.5. For population based algorithms, mutation occasionally occurs in order to allow a child to obtain features that are not possessed by either of its parents. This process helps the algorithm explore new and possibly better genetic material than has been previously considered. The process of mutation is shown in fig. 10.6.

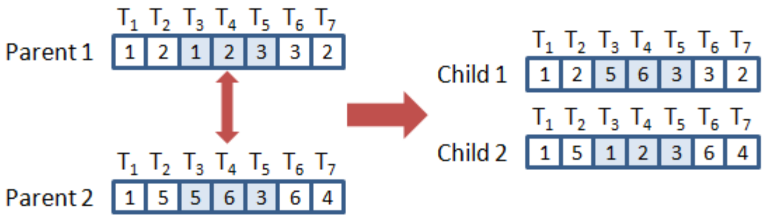

**Fig. 10.5** The Crossover operation: First, two random parents are chosen from the current population. Then two random points are selected from the schedule order of both parents. The locations of all tasks between the two parents are exchanged. Two new offsprings are generated by combining task assignments taken from two parents.

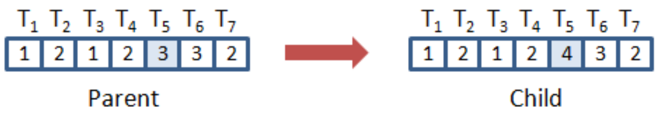

**Fig. 10.6** The Mutation operation: A task is randomly selected in a chromosome. An alternative service which is also capable of executing the task is randomly selected to replace the current task allocation

The new population is now ready for another round of chaotic mapping, crossover, and mutation, producing yet another generation. So the initial population is replaced by the newly generated individuals. More generations are produced until the stopping condition (a maximum number of generations) is met. The fittest chromosome is thus returned as a solution.

### 10.5.4 Experimental Results

Given that different workflow applications may have different impact on the performance of the scheduling algorithms, we have evaluated algorithms on different workflow structures. According to many grid workflow projects [45], workflow applications can be categorized into *balanced structures* and *unbalanced structures*. Fig. 10.7 shows balanced and unbalanced-structure applications used in our experiments. As shown in Fig. 10.7(a), the balanced-structure application consists of several parallel pipelines, which require the same types of services but process different data sets. As can be seen in Fig. 10.7(b), the structure of the unbalanced application is more complex. Unlike the balanced-structure application, many parallel tasks in the unbalanced structure require different types of services, and their workload and I/O data varies significantly.

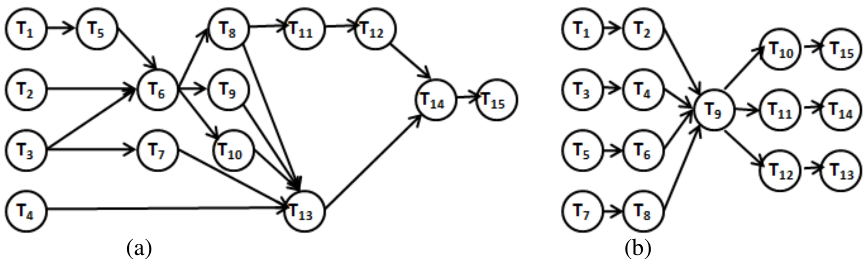

**Fig. 10.7** Workflow structures: (a) Balanced workflow (fMRI). (b) Unbalanced workflow (DNA)

A Chaos-Genetic scheduling Algorithm (CGA) is introduced to solve the workflow execution planning problem. Our goal is to simultaneously minimize two conflicting objectives; execution time and execution price while meeting users' maximum time constraint (deadline) and price constraint (budget). We have simulated 15 types of services with various price levels. The parameter settings used as a default configuration for the algorithms are listed in Table 10.1. The behaviors of algorithms are also observed at three constraint levels, namely relaxed constraint, medium constraint, and tight constraint. The relaxed constraint level assumes that users require relatively large deadline and budget, while the tight constraint level assumes that users require small deadline and budget. In other words, the relaxed/tight deadlines and budgets of an application are determined by the maximum/minimum time and cost for the workflow execution.

**Table 10.1** Parameter settings for workflow scheduling problem

| Parameter | Value/type |
|---|---|
| Population size | 10 |
| Initial population | randomly generated solution |
| Maximum Generation | 100 |
| Crossover Probability | 0.98 |
| Mutation Probability | 0.05 |
| Maximum Iteration | 10 |

As it is illustrated in Fig. 10.8, neither of GA and CGA satisfy the low budget constraint (about G$3500), however CGA shows better results in both applications. Results are gradually improved under medium budget constraints. Obviously, the descending trend in the diagram shows that as the budget increases, it'll be easier for the algorithms to meet the user budget constraints. On the other hand, considering the differences between the two approaches, it is clear that GA takes longer to complete even under relaxed constraints. Therefore, CGA shows better performance compared to GA in both applications.

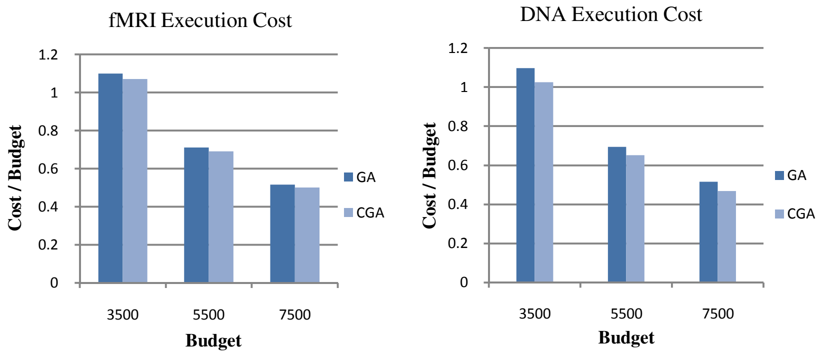

**Fig. 10.8** Comparison between the execution cost of GA and CGA on balanced (fMRI) and unbalanced (DNA) workflows, under three constraint types: tight (G$3500), medium (G$5500) and relaxed (G$7500). Each experiment was repeated 10 times and the average values are used to report the results. For fMRI, the results are obtained under the assumption of D = 220(H) and D = 240(H) for DNA. The values of the vertical axes are the result of the total cost divided by the user budget constraint.

In Fig. 10.9 a comparison between the execution times of the two algorithms on fMRI and DNA workflows is illustrated. Here we change the user deadline values from 190(H) to 290(H) for DNA and from 170(H) to 270(H) for fMRI, since the latter is a balanced workflow and takes less time to complete. It can be seen that GA takes longer to complete in most of the conditions. The differences are obviously better observed in the unbalanced workflow structure.

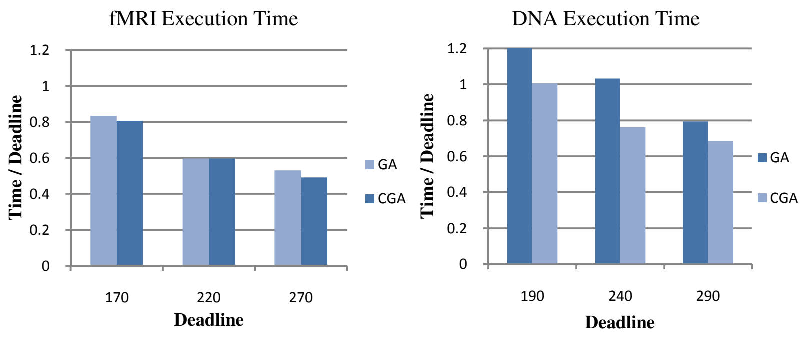

**Fig. 10.9** Comparison between the execution time of GA and CGA on balanced (fMRI) and unbalanced (DNA) workflows, under three constraints: tight (around 180H), medium (around 230H) and relaxed (around 280H) with a medium budget of G$5000. Each experiment was repeated 10 times and the average values are used to report the results.

In all of the above diagrams, there are conditions where CGA and GA show similar results (for instance in Fig. 10.9 for fMRI, under medium constraint). These are the conditions where GA solutions were not trapped in a local optimum, resulting in similar performance patterns for the two algorithms. In those conditions, CGA does not do any good in keeping the suitable solutions. In the rest of the states though, GA, is stuck somewhere in a local optimum (as it usually is), which prevents it from producing possible better results. In other words, CGA takes advantage of the characteristics of the chaotic variable to make the individuals of subgenerations distributed ergodically in the defined space and thus to avoid premature convergence [30]. It also takes advantage of the convergence characteristic of GA to overcome the randomness of the chaotic process and hence to increase the probability of finding the global optimal solution.

## 10.6 Network-on-Chip (NoC): Case Study # 2

System-on-Chip (SoC) is a chip design method where all of the components of an electronic system are integrated into a single chip. Benefits of this integration compared with traditional multi-chip design include a size and energy reduction. An important concept in chip design is the *core*. A *core* is basically a separate and reusable unit of logic. Examples of *cores* include processors, memory banks and external communication components. These cores may be licensed from a number of vendors, under the common label *Intellectual Property-cores* (*IP-cores*). A System on Chip can include many *IP-cores* that need to communicate with each other. This is traditionally done by shared buses and ad-hoc core to core links. Using such traditional communication structures, functions well without creating communication bottlenecks when a system has few *cores* [46]. But as the number of *IP-cores* increases, the number of potential connections between them increases exponentially to a point where assigning the same bus to many *IP-cores* is not a practical option due to latency issues.

Over time, traditional SoC communication methods gradually became inefficient and complex, and do not scale well for large SoCs (say, more than 20 IP-cores) [46]. The increasing complexity of such systems leads to some difficulties in creating a proper communications infrastructure for the chip. When time-division buses and custom point to point communications are no longer sufficient, more elaborate networks are the obvious choice. By going beyond current buses and custom communication designs for the higher levels of interconnection on the chip, it might indeed be possible to reach higher performance with lower design and verification costs. A scalable communication architecture that avoids these problems is required and this is where creating a global network on the chip becomes a viable option.

### 10.6.1 Network on Chip

Network on Chip (NoC) is an emerging communication method for a System on Chip [47]. NoC attempts to solve the communication problems mentioned in the previous section by creating an inter-chip network consisting of network adapters, routers/switches and links between them. Each IP-core is connected to a network adapter which converts the transaction data from the IP-core to the flow digits (flits) transmitted across the network. Fig. 10.10 illustrates the basic concepts in a NoC.

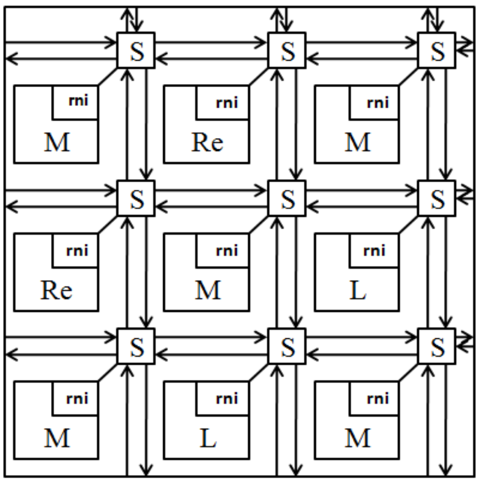

**Fig. 10.10** An example of the Network-on-Chip architecture with ‘S’ for Switches, ‘M’ for Memory, ‘Re’ for Reconfigurable logic, ‘rni’ for resource-network interface and ‘L’ for dedicated hardware**.**

The individual IP-cores do not need to be aware of how the data is transmitted on the network. This decoupling of the processing from communication is an important benefit of NoC. This means that IP-cores with different transaction standards can easily communicate with each other, simplifying the design process. Other benefits include shorter, simplified wiring and lower energy usage. There are also some potential drawbacks including increased delay/latency especially if the network is congested, and the extra space used on the chip for the routers and network adapters. However the overhead is estimated to be fairly small [48] and space is usually not the bottleneck in chip design especially with the continuing shrinking microchip technologies.

The future for NoC looks promising, but many problems need to be addressed in order for it to find more widespread application. One of the problems is how to connect systems of IP-cores that vary in size and/or communication requirements (a heterogeneous SoC) defined as the network layout or topology selection. Three main factors that have to be taken into account when evaluating a NoC design are latency, energy usage and size (area overhead). Another important matter is application mapping which deals with finding the best node arrangement with the aim of improving the quality of service parameters. The mapping process can be described as follows: first select a set of IP-cores to distribute the data processing on and secondly construct a topology that connects the IP-cores and minimizes communication costs. The selected set of IP-cores and the data transmission between them constitutes what is defined as the Core Graph.

### *10.6.2 Problem Description*

The investigation of different network topologies pointed to a two-dimensional mesh as the most suitable topology for most on-chip networks. This is also the common topology proposed by most researchers [48], [49], [50]. The main reasons for selecting the two-dimensional mesh instead of other topologies such as hypercubes, butterflies, or trees are that a two-dimensional mesh has an acceptable wire cost, reasonably high bandwidth, and a nice mapping onto a chip. Routing either refers to the problem of connecting a topology or choosing data transmission routes through a constructed topology. The transmission routes can either be static or dynamic. Dynamic routing is definitely more flexible [46] but requires more complex routers and larger buffers in the network. The static routing scheme chosen in the implementation of the algorithm is a shortest path routing algorithm.

The input of our problem is a directed task graph $G(V, E)$, in which every $v_i \in V$ denotes a processing element or a memory unit (generally an IP core), and a directed edge $e_k = (v_i, v_j)$ denotes a communication trace from the source node $v_i$ to the destination node $v_j$. The weights of the edges $w(e_k)$ usually refer to the communication cost between two corresponding nodes. A mesh based topology of NoC is defined by $(U, L)$, where each vertex $u_i \in U$ denotes a node in the topology and each $l_i \in L$ denotes a physical link between two vertices. The weight of a link $w\ (l_k)$ represents the bandwidth available across the link $l_k$. Fig. 10.11 exhibits the mapping process of a sample task graph onto a tile-based mesh structure.

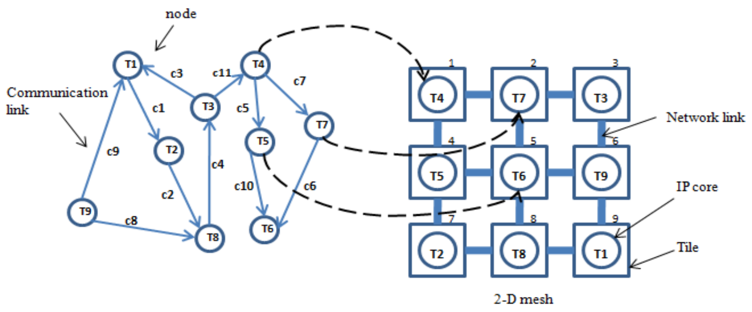

**Fig. 10.11** The mapping process

In order to optimize the results of the mapping process, various authors have tried to enhance the results considering different performance elements. Lei and Kumar, proposed a two-step GA for mapping task graphs to the NoC architecture in [51], with the objective of minimizing the average communication delay of the network. Following the same objective, Murali and De Micheli, proposed NMAP, as a fast algorithm that maps the cores onto mesh NoC architecture under bandwidth constraints, in [52]. BMAP a binomial mapping and optimization algorithm that reduce the hardware cost of on-chip network infrastructure [53].

In Chaos-Genetic Mapping (CGMAP) approach [54], determining solution representation is the first priority. The values of the genes in this problem can only be integer values between 1 and $n$ (the value of $n$ is proportional to the number of tiles in the mesh). The length of each chromosome depends on the number of nodes in the communication task graph. Population size is another important parameter. In an actual application, it would be common to have somewhere between a few dozen and a few hundred individuals. For the purposes of this problem, we assume that the first population consists of 100 individuals. The initialization of the first population is done randomly by means of a random number generator which assigns values between 1 and $n$, to each of the $n$ positions in every one of 100 individuals. Then the chaotic mapping operator is applied to each individual in the initial population and creates the chaotic population. At this stage, the fitness of all 200 individuals is evaluated. The fitness value is often proportional to the output value of the function being optimized. Since data always take the shortest distance in the network and often more than one such path exists for data going from node $v_i = (x_i, y_i)$ to $v_j = (x_j, y_j)$ , we estimated this hop distance as

$$hd(e_k) = (|x_i - x_j| + (|y_i - y_j|), \tag{10.14}$$

and defined the fitness function as follows

$$F = \sum\nolimits_{\forall e_k} w(e_k) hd(e_k) \,. \tag{10.15}$$

Elitism is incorporated into the algorithm by transferring the single fittest individual directly to the next generation. Crossover and mutation are also performed on randomly selected individuals. The initial population is replaced by these newly generated individuals. Obviously, more generations are produced until the stopping condition (a maximum number of generations) is met. The fittest chromosome is thus returned as a solution.

## *10.6.3 Simulation Results*

The results of the execution of CGMAP on two benchmark applications are demonstrated in this section; a Video Object Plane Decoder (VOPD) with 16 IP-cores and 20 links and an MPEG-4 decoder with 12 nodes and 13 links. Fig. 10.12 is an instance of a VOPD task graph mapped onto a two-dimensional mesh using CGMAP. Afterwards the results are compared with those of previous mapping algorithms such as NMAP [52], BMAP [53], PBB [55], etc. using the same routing and scheduling characteristics.

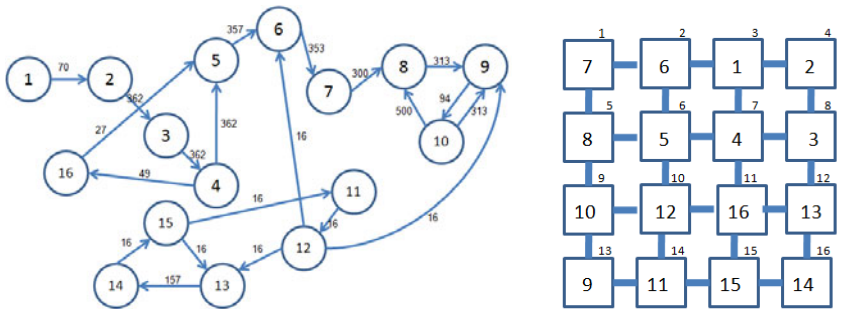

**Fig. 10.12** VOPD task graph and the place of each associated IP core in a 2-dimensional mesh

Fig. 10.13 demonstrates the results of CGMAP compared with five other mapping algorithms in both applications, considering the communication costs. As it is clear in the figure, CGMAP performs well in both applications. Table 10.2, shows a comparison between the hop counts of the three most efficient mapping algorithms for two benchmark applications. The hop count is a measure of distance across an IP-based network which keeps track of the number of intermediate devices (like routers) an IP packet has to pass through in order to reach its destination. Generally speaking, the more hops data must traverse to reach their destination, the greater the transmission delay incurred. Assuming the average hop count in NMAP is 1, the table proves that using CGMAP decreases the hop number to an average of 0.97 in the first application (MPEG-4) and to 0.99, in the case of the second application (VOPD).

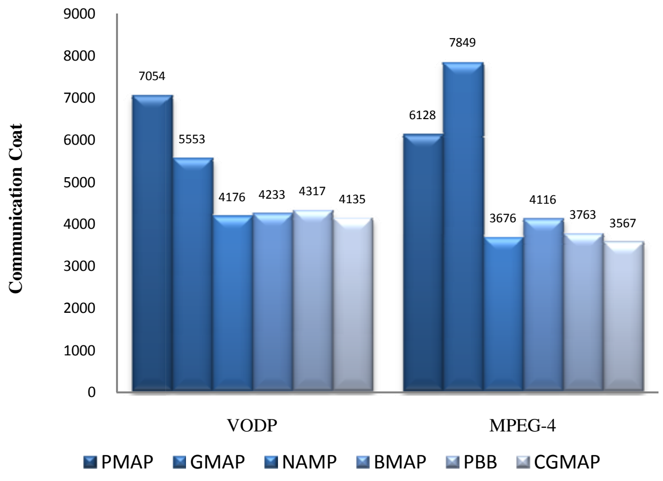

**Fig. 10.13** Comparison between the communication costs of six mapping algorithm in NoC

**Table 10.2** Comparision between the average hop count of mapping algorithms in NoC

| | NMAP | BMAP | CGMAP | GMAP | PMAP | PBB |
|---|---|---|---|---|---|---|
| **VOPD** | 1 | 1.01 | 0.99 | 1.33 | 1.69 | 1.03 |
| **MPEG - 4** | 1 | 1.71 | 0.95 | 2.13 | 1.67 | 1.02 |

## *10.6.4 Performance Analysis of One-Dimensional Chaotic Maps*

In this section, the efficiency of the discussed chaotic maps is compared when used as an operator in GA [56]. Fig. 10.14 shows a convergence rate comparison

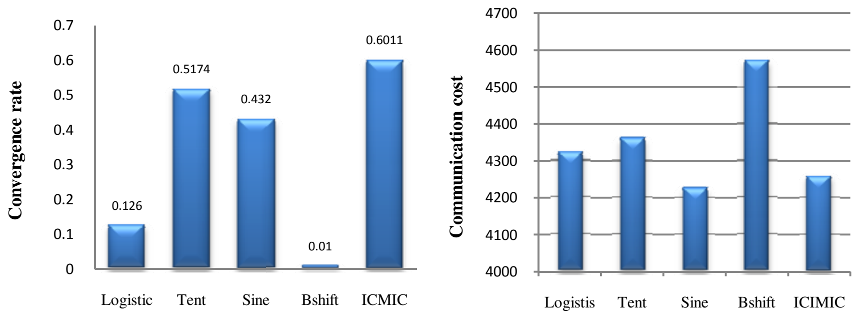

**Fig. 10.14** Convergence rate and communication cost comparison between chaotic maps. The algorithm is run10 times with each chaotic operator and the average results are depicted. The benchmark application used in this section is VOPD.

between five one-dimensional chaotic maps. It is noticeable that, ICMIC and Tent map have the greatest convergence rates and the lowest convergence rate belongs to the Bernoulli Shift and the Logistic map. This means that in order to get the best results for this specific application, CGMAP should be implemented with the ICMIC map as a chaotic operator. This way the algorithm reaches an optimum solution within the shortest time period.

Communication costs of executing CGMAP are also compared with each of the discussed chaotic maps and the average results are demonstrated in Fig. 10.14. The Sine map achieves the lowest communication cost among all and the Bernoulli Shift costs a lot to complete the application. The main aim of this experiment was to prove that the choice of the most effective chaotic map is a function of the benchmark problem on the one hand and the main objective of the problem on the other.

## 10.7 Concluding Remarks

The chaos optimization algorithm adopts chaos variable to search, and the search goes on according to the regularity characteristics of the chaotic variables. Chaos variable's traversal property ensures that a true optimum solution can be found if allowed to run for sufficient time. Even if the optimization calculation time is limited we can get approximate solution with extremely good precision.

Grid Scheduling and Network-on-Chip mapping problems both belong to the group of NP-complete problems, which are traditionally solved using metaheuristic algorithms such as GA. In this chapter a Chaos-Genetic Algorithm (CGA) was used in order to take advantage of the properties of the chaotic variables to make the search of optimal values in GA more effective and faster. This is done by designing a chaotic mapping operator, using one-dimensional chaotic maps and applying it to the GA along with the common genetic operators, namely crossover and mutation. Experimental results were highly dependent upon the chaotic map that was used. Therefore, by prioritizing the favorites that one seeks, a chaotic equation may be selected that is the most congruent with ones will.

## References

1. Bondy, J.A., Murty, U.S.R.: Graph Theory. Springer, Heidelberg (2008)
2. Korte, B., Vygen, J.: Combinatorial Optimization: Theory and Algorithms, Algorithms and Combinatorics, 4th edn. Springer, Heidelberg (2008)
3. Weise, M.: Global Optimization – Theory and Application, 2nd edn. Thomas Weise (2009)
4. Yang, X.-S.: Nature-Inspired Metaheuristic Algorithms. Luniver Press (2008)
5. Yang, X.-S.: Engineering Optimization: An Introduction with Metaheuristic Applications. Wiley, Chichester (2010)
6. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science 220(4598), 671–680 (1983)

7. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
8. Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm Intelligence. Morgan Kaufmann, New York (2001)
9. Dorigo, M.: Optimization, Learning and Natural Algorithms (Phd Thesis), Politecnico di Milano, Italy (1992)
10. Glover, F., Laguna, M.: Tabu Search. USA Norwell. Kluwer Academic Publishers, Dordrecht (1997)
11. Blum, C., Roli, A.: Metaheuristics in Combinatorial Optimization: Overview and conceptual comparison. ACM Computing Surveys 35(3), 268–308 (2003)
12. Ribeiro, C., Hansen, P.: Essays and Surveys in Metaheuristics. Kluwer Academic Publishers, Norwell (2002)
13. Melanie, M.: An Introduction to Genetic Algorithm. A Bradford book MIT press, London (1998)
14. Haupt, R.L., Haupt, S.E.: Practical Genetic Algorithms, 2nd edn. Wiley-Interscience Publication, Hoboken (1998)
15. Stavroulakis, P.: Chaos Application in Telecommunications. CRC Press Taylor and Francis group, New York (2006)
16. Strogatz, S.: Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering. In: Perseus Books (1994)
17. Tavazoei, M.S., Haeri, M.: Comparison of Different One-Dimensional Maps as Chaotic Search in Chaos Optimization Algorithms. Applied Mathematics and Computation 187(2), 1076–1085 (2007)
18. He, Y.Y., Zhou, J.Z., Xiang, X.Q.: Comparison of different chaotic maps in particle swarm optimization algorithm for long term cascaded hydroelectric system scheduling. Chaos Solitons Fractals 42(5), 3169–3176 (2009)
19. Ott, E.: Chaos in Dynamical Systems. Cambridge University Press, U.K (2002)
20. Erramili, A., Singh, R.P., Pruthi, P.: Modeling Packet Traffic with Chaotic Maps. Royal Institute of Technology, Sweden (1994), ISRN KTH/IT/R-94/18-SE
21. He, D., He, C., Jiang, L.G., Zhu, H.W., Hu, G.R.: A chaotic map with infinite collapses. In: Proc IEEE tencon., Kuala Lumpur, Malaysia, vol. 3(9), pp. 95–99 (2000)
22. He, D., He, C., Jiang, L.G., Zhu, H.W., Hu, G.: Chaotic characteristics of a one-dimensional iterative map with infinite collapses. IEEE Trans. 48(7), 900–906 (2001)
23. Lu, Z., Shieh, L.S., Chen, G.R.: On robust control of uncertain chaotic systems: a sliding-mode synthesis via chaotic optimization. Chaos Solitons & Fractals 18(4), 819–836 (2003)
24. Yang, J.J., Zhou, J.Z., Wu, W., Liu, F.: A chaos algorithm based on progressive optimality and tabu search algorithm. In: IEEE Proc. 4th International Conf. Machine Learning and Cybernetics, vol. 5, pp. 2977–2981 (2005)
25. Li, B., Jiang, W.S.: Chaos Optimization Method and Its Application. Control Theory and Application 14, 613–615 (1997)
26. Gao, L., Liu, X.: A Resilient Particle Swarm Optimization Algorithm based on chaos and applying it to optimize the fermentation process. International Journal of Information and Systems Sciences 5(3-4), 380–391 (2009)
27. Bucolo, M., Caponetto, R., Fortune, L., Frasca, M., Rizzo, A.: Does chaos work better than noise? IEEE Circuits and Systems Magazine, 4–19 (2002)
28. Hongkai, W., Zhiming, C., Pingbo, W., Yinfeng, F.: Study of Intelligent Optimization Methods Applied in Fractional Fourier Transform. International Journal of Computer Theory and Engineering 2(4), 1793–8201 (2010)

29. Cheng, C.: Optimizing hydropower reservoir operation using hybrid genetic algorithm and chaos. Water Resources Management 22(7), 895–909 (2008)
30. Yan, X.F., Chen, D.Z., Hu, X.S.: Chaos-genetic algorithms for optimizing the operating conditions based on RBF-PLS model. Elsevier Computers and Chemical Engineering, 1390–1404 (2003)
31. Moein-Darbari, F., Khademzaheh, A., Gharoonifard, G.: CGMAP: A new Approach to Network-on-Chip Mapping Problem. IEICE Electronic Express 6(1), 27–34 (2009)
32. Foster, I., Kesselman, C.: Computational Grids. In: The Grid: Blueprint for New Computing Infrastructure, pp. 15–52. Morgan Kaufmann, San Francisco (1998)
33. Dong, F., Akl, S.G.: Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. In: School of Computing, Queen's University Kingston, Ontario, pp. 1–55 (2006)
34. Schopf, J.M.: Ten Actions When SuperScheduling, document of Scheduling Working Group. In: Global Grid Forum (2001)
35. Yang, Y., Casanova, H.: NP-complete Scheduling Problems. Journal of Computer and System Sciences 10, 434–439 (1975)
36. Mandal, A., Kennedy, K., Koelbel, C., Martin, G., Mellor-Crummey, J., Liu, B., Johnsson, L.: Scheduling Strategies for Mapping Application Workflows onto the Grid. In: IEEE International Symposium on High Performance Distributed Computing (HPDC 2005), Research Triangle Park, NC, pp. 125–134 (2005)
37. Eilam, T., Appleby, K., Breh, J., Breiter, G., Daur, H., Fakhouri, S.A., Hunt, G.D.H., Lu, T., Miller, S.D., Mummert, L.B., Pershing, J.A., Wagner, H.: Using a utility computing framework to develop utility systems. IBM System Journal 43(1), 97–120 (2004)
38. Laszewski, G.V.: Java CoG Kit Workflow Concepts for Scientific Experiments. Argonne National Laboratory, Argonne, IL, USA Technique Report (2005)
39. Buyya, R., Giddy, J., Abramson, D.: An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications. In: 2nd Workshop on Active Middleware Services (AMS 2000). Kluwer Academic Press, Dordrecht (2000)
40. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.: Scheduling workflows with budget constraints. In: Gorlatch, S., Danelutto, M. (eds.) Integrated Research in GRID Computing, ser., pp. 189–202. Springer, Heidelberg (2007)
41. Zhu, Y.: A Survey on Grid Scheduling System. Department of Computer Science, Hong Kong University of Science and Technology (2003)
42. Gharooni-fard, G., Moein-darbari, F., Deldari, H., Morvaridi, A.: Scheduling of Scientific Workflows Using a Chaos-Genetic Algorithm. In: Procedia Computer Science vol. 1(1), pp. 1439–1448 (2010)
43. Yu, J., Buyya, R.: Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms. Scientific Programming, 217–230 (2006)
44. Yu, J., Kirley, M., Buyya, R.: Multi-objective Planning for Workflow Execution on Grids. In: 8*th* IEEE ACM International Conference on Grid Computing, Singapore, pp.10—17 (2007)
45. Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., Kennedy, K.: Task scheduling strategies for workflow-based applications in grids. In: 5*th* IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005), vol. 2, pp. 759–767 (2005)
46. Bjerregaard, T., Mahadevan, S.: A Survey of Research and Practices of Network-on-Chip. ACM Computing Surveys, New York (2006)

47. De Micheli, G., Benini, L.: Network on Chip: A New Paradigm for System-on-Chip Design., pp. 7–78 (2002)
48. Dally, W.J., Towles, B.: Route Packets, not Wires: on Chip Interconnection Networks. In: Proceedings of the Design Automation Conference (DAC), pp. 684–689 (2001)
49. Saastamoinen, I., Sigüenza-Tortosa, D., Nurmi, J.: Interconnect IP Node for Future System-on-Chip Designs. In: IEEE International workshop on Electronic design, Test, and Applications, New Zealand, pp. 116–120 (2002)
50. Sgroi, M., Sheets, M., Mihal, A., Keutzer, K., Malik, R.J., Sangiovanni-Vincentelli, A.: Addressing the System-on-Chip Interconnect Woes through Communication-based Design. In: The Design Automation Conference (DAC), pp. 667–672 (2001)
51. Lei, T., Kumar, S.: A Two Step Genetic Algorithm for Mapping Task Graphs to Network on Chip Architecture. In: Proceedings of the 3rd International Conference DSD 2003, Turkey, pp. 180–187 (2003)
52. Murali, S., Micheli, G.D.: Bandwidth-Constrained Mapping of Cores on to NoC Architectures. In: $4^{th}$ International Conference on DATE 2004, pp. 896–901 (2004)
53. Shen, W.T., Chao, C.H., Lien, Y.K., Wu, A.Y.: A new Binomial Mapping and Optimization Algorithm for Reduced-Complexity Mesh-Based On-Chip Network. In: $1^{st}$ IEEE International Symposium on Networks-on-Chip (NOCS 2007), New Jersey, pp. 317–322 (2007)
54. Moein-darbari, F., Khademzadeh, A., Gharooni-fard, G.: Evaluating the Performance of Chaos Genetic Algorithm for Solving the Network-on-Chip Mapping Problem. In: IEEE International Conference on Computational Science and Engineering, Vancouver, Canada, vol. 2, pp. 366–373 (2009)
55. Hu, J., Marculescu, R.: Energy-Aware Mapping for Tile-based NoC Architectures under Performance Constraints. ASP-DAC, 233–239 (2003)
56. Gharooni-fard, G., Khademzade, A., Moein-darbari, F.: Evaluating the Performance of Chaotic Maps in Network-on-Chip Mapping Problem. IEICE Electronic Express 6(12), 811–817 (2009)