NAIST-IS-DD0461016

Doctoral Dissertation

Improvement of Exploration Efficiency of Island Model GA and Application to Routing Problem in Mobile Ad Hoc Network

Eiichi Takashima

February 1, 2007

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Eiichi Takashima

Thesis Committee:

Professor Minoru Ito	(Supervisor)
Professor Kenji Sugimoto	(Co-supervisor)
Associate Professor Keiichi Yasumoto	(Co-supervisor)

Improvement of Exploration Efficiency of Island Model GA and Application to Routing Problem in Mobile Ad Hoc Network[*]

Eiichi Takashima

Abstract

In the real world, there are various combinatorial optimization problems (called COPs, hereafter) which belong to NP-hard class. Therefore, if the problem size becomes large, it is difficult to find an optimal solution in practical time. Then, as one of the approximation methods, genetic algorithm (called GA, hereafter) has been studied. GA is an approximation method for COPs, imitating evolution mechanism in nature. However, it is known that GA still requires large computation cost. Therefore, parallel GA has been studied. Parallel GA is a technique to compute a semi-optimal solution at high speed by parallel computation on multiple computers.

This study focuses on island model GA (called IGA, hereafter) which is a kind of parallel GA, and treats two main topics. The first topic treats design and implementation of the self-adaptive mechanism for IGA to improve its exploration efficiency. The second topic treats a method to apply IGA to a multicast routing problem in mobile ad-hoc network (called MANET, hereafter).

In the first topic, we propose a new method which can automatically adjust optimal parameters for IGA such as mutation rate, population size (i.e., the number of candidate solutions) and so on. In order to reduce labor required to adjust parameters, adaptive GAs (called AGA, hereafter) which automatically adjust parameters have been proposed. Most of the existing AGAs adjust only a few parameters at one time. Although several AGAs can adjust many parameters simultaneously, these algorithms need large computation cost. In this thesis, we propose a new adaptive method. The proposed

[*]Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0461016, February 1, 2007.

method has following features: adjustment for multiple parameters, reasonable computation cost and easy parallelization. The proposed method adopts a mechanism to simultaneously adjust parameter values while IGA is searching a solution, by dividing the whole search process to multiple phases and alternately executing meta-GA for deriving parameter values and IGA for searching a solution with the derived values, in each phase. When multiple computers with different performances synchronize, faster computer may have to wait until slower computer finishes its task. To avoid this problem, we have developed a mechanism to allow different computers (corresponding to islands of IGA) to proceed asynchronously so that search efficiency is improved by reducing idle time. Through experiments, we confirmed that performance of the proposed method outperforms simple GA with standard parameter values, and is close to simple GA with manually adjusted optimal parameter values. Also we confirmed that the proposed method can reduce idle time on computers which have different computation powers.

In the second topic, we propose a method which efficiently constructs a semi-optimal multicast tree in consideration of QoS using IGA in MANET. Generally, it is known that the problem to compute the optimal multicast tree satisfying multiple QoS constraints belongs to NP-hard class. Therefore, approximation methods to find QoS multicast tree using GA have been proposed. However, there is a problem that the computation and communication resources on a mobile terminal are too restrictive on MANET. Since existing methods are based on centralized control, it is difficult to apply them to MANET environments. In this thesis, we propose a new multicast routing method for MANET. The proposed method constructs the semi-optimal multicast tree satisfying QoS constraints for any given objective using IGA. In order to increase scalability, the proposed method first divides the whole MANET to multiple clusters, and computes a tree for each cluster and a tree connecting all clusters. Each tree is computed by IGA in some of the mobile terminals. Through experiments using network simulator, we confirmed that the proposed method outperforms existing on-demand multicast routing protocol in terms of some useful objectives on power consumption and tree stability.

Keywords:
island model GA, self-adaptation, mobile ad hoc network, multicast routing, QoS, distributed computation

Contents

List of Figures

List of Tables

1. Introduction

In the real world, there are various combinatorial optimization problems (called COPs, hereafter) such as parcel pickup and delivery problem, processor allocation problem for multiple concurrent tasks with deadlines and computation problem for optimal data delivery path in network, etc. It is known that most of these COPs belong to NP-hard class. In problems of NP-hard class, computation time for finding optimal solution increases exponentially with the problem size. Therefore, if the problem size becomes large, it is difficult to find an optimal solution in practical time. Then, as one of the approximation methods, genetic algorithm (called GA, hereafter) has been studied. GA is an approximation method for COPs, imitating evolution mechanism in nature. In GA, one point of the search space for the target problem is represented by a character string called a chromosome. An individual has a chromosome and a fitness value (which indicates how good solution it is) with a given evaluation function. First, GA randomly generates a set of individuals (called population, hereafter). The population is evolved by repeating evaluation, selection, crossover and mutation. Consequently, a semi-optimal solution is obtained. However, it is known that GA still requires large computation cost until obtaining a good solution. Therefore, parallel GA has been studied. Parallel GA is a technique to compute a semi-optimal solution at high speed by parallel computation on multiple computers. Parallel GA is classified into an island model, a cellular model, etc [28]. In island model GA (called IGA, hereafter), it prepares multiple populations, and they are assigned to different computers. In each computer, GA applies GA operations to population, and searches for a semi-optimal solution within the population. Since some individuals are exchanged between computers periodically, those computers share information of superior individual as well as individuals with diverse chromosomes, and can search for a better solution efficiently.

In this thesis, we focus on IGA and deal with two main topics. The first topic treats design and implementation of the self-adaptive mechanism for IGA to improve its exploration efficiency, which simultaneously searches for a solution and adapts parameters required for GA such as population size (i.e., the number of individuals), mutation rate and so on. The second topic treats a method to apply IGA to a problem in the real world. The problem is a multicast routing problem in mobile ad-hoc network (called MANET, hereafter).

First, in Chapter 2, we describe a new method which can automatically adjust optimal parameters for IGA [32, 33, 34].

For efficient execution of GA, many parameter values (e.g., crossover rate, mutation rate and so on) have to be set appropriately. However, optimal parameter values depend on a target problem and genetic operators. Also, the optimal value for each parameter depends on the other parameter values. Accordingly, adjusting these values tends to be tedious and time-consuming. In order to address this problem, several studies about setting of suitable parameter values have been proposed [9, 14]. However, the existing studies are targeting some specific problems, and in order to apply these studies to a new problem with different characteristics, analysis for the problem is required. To cope with the above problems, many adaptive GAs (called AGAs, hereafter) which automatically adjust parameter values have been proposed. Most of the existing AGAs adjust only a few parameters at one time [3, 16, 39].

Meanwhile, some of AGAs such as meta GA [40] and agent-oriented self-adaptive GA (A-SAGA, hereafter) which is the direct ancestor of the proposed method [24] can adjust many parameter values simultaneously. Meta-GA is traditional method. Meta-GA is a GA which searches the optimal parameter values for other GA (i.e., GA for solving the target problem). A-SAGA is a method which applies meta-GA to IGA. Since A-SAGA has the mechanism of island model GA and can evaluate as many parameters as the number of islands in parallel, A-SAGA can adapt parameters more efficiently than Meta-GA. Since these algorithms solve a problem repeatedly, they are effective in the case of solving many mutually similar instances in the same problem continuously. However, since meta-GA and A-SAGA solve a problem repeatedly in order to adjust parameter values, they still need large computation cost and are unsuitable in the case of solving only one instance of a problem.

In Chapter 2, we propose self adaptive island model GA (called SAIGA, hereafter). SAIGA has following features: simultaneous adjustment for multiple parameters, reasonable computation cost and easy parallelization. SAIGA adopts a mechanism to simultaneously adjust parameter values while IGA is searching a solution, by dividing the whole search process to multiple phases and alternately executing meta-GA for deriving parameter values and IGA for searching a solution with the derived values, in each phase. With this improvement, SAIGA solves a problem without repeating searches from scratch as in meta-GA. SAIGA can efficiently find a solution with automatic parameter adjustment, even when solving only one instance of a problem (For example, it is helpful when a circuit layout is designed by trial and error.).

When multiple computers with different performances synchronize, faster computer may have to wait until slower computer finishes its task. To avoid this problem, we propose asynchronous SAIGA (called A-SAIGA, hereafter) which has a mechanism to allow different computers (corresponding to islands of IGA) to proceed asynchronously so that search efficiency is improved by reducing idle time. In order to evaluate effectiveness of the proposed method, we measured search efficiency for various problems. The results showed that performance of the proposed method outperforms simple GA with standard parameter values, and is close to simple GA with manually adjusted optimal parameter values. Also, we confirmed that the proposed method can reduce idle time on computers which have different computation powers.

Second, in Chapter 3, we describe an application of IGA to solving a multicast routing problem in MANET [35, 36]. Recent progress of wireless communication technology and small but powerful computing devices has enabled important and useful applications of MANET, such as information dissemination to pedestrians and vehicles in urban areas or disaster areas even without infrastructure. Among many types of information for dissemination, video and other streaming data would be most important. For this purpose, several multicast protocols for MANET have been proposed, to dynamically construct multicast tree. In order to realize multimedia communication on MANET, a lot of QoS routing protocols have been proposed so far. As unicast routing methods which treat a single QoS constraint, [7, 22, 23, 30, 41] have been proposed. Several QoS multicast routing methods on MANET have also been proposed [6, 31]. However, these existing methods are not designed to compute multicast tree satisfying two or more QoS constraints at the same time or optimized for a given objective such as minimization/maximization of power consumption, stability, the number of receivers, and so on. Generally, it is known that the problem to compute the optimal multicast tree satisfying two or more QoS constraints is NP-hard. Since, in MANET, topology dynamically changes as node moves, it is necessary to reconstruct a multicast tree in accordance with topology change. Moreover, it is necessary to consider that computation and communication capabilities of mobile terminals in MANET are limited. So, it is hard to solve the problem to construct/maintain the optimal multicast tree in MANET.

There is a study on routing techniques to construct a multicast tree which simultaneously satisfies two or more QoS constraints and is semi-optimal for a given objective [21]. However, this existing method supposes a single computation node which knows topology information of the whole MANET and computes multicast tree in a centralized manner. So, they do not scale against the number of nodes due to communication cost for gathering the topology information and computation cost. Therefore, if the number of nodes of MANET increases, it becomes difficult to compute a tree on mobile node in scarce computation resources.

In Chapter 3, we propose a new QoS multicast routing method for MANET called HQMGA (hierarchical QoS multicast routing using GA in MANET). HQMGA allows MANET nodes to dynamically construct a semi-optimal multicast tree satisfying several QoS constraints at the same time for any given objective. HQMGA is scalable against the number of nodes and can treat arbitrary objective for optimization such as minimization of total power consumption in MANET, maximization of tree stability, and so on. HQMGA reduces the cost for constructing multicast trees consisting of many mobile nodes so that the computation and communication can be executed within capability of each mobile node. To do so, it divides MANET into multiple clusters, and makes them construct sub-trees covering their clusters in parallel, and computes a tree which connects the sub-trees of all clusters. Consequently, one semi-optimal multicast tree is constructed with them. Here, HQMGA abstracts the topology information spanning all clusters so that a node can compute the tree connecting all clusters within small computation power and communication amount. For computation of semi-optimal trees, we developed IGA-based algorithm which can run on mobile terminals. We also designed and developed protocols for gathering topology information and for distributing the computed tree. Through experiments, we show that our algorithm can compute the multicast tree by mobile computer in reasonable time, and the computation time of the sub-tree for each cluster and the tree connecting all clusters was within 3.3 seconds by laptop PCs when MANET includes one thousand of nodes. We confirmed that the control traffic required for gathering topology information is small enough. We also implemented HQMGA on a network simulator GTNetS [27] and compared the superiority of the computed trees with one of existing QoS multicast routing protocols, AQM [6]. Consequently, we confirmed that HQMGA outperforms AQM in terms of some useful objectives on power consumption and tree stability.

2. Improvement of Exploration Efficiency of Island Model GA

2.1 Introduction

Genetic Algorithm (called GA, hereafter) is a combinatorial optimization method imitating natural evolution of creatures. For efficient execution of GA, many parameter values such as mutation rate and crossover rate have to be set appropriately. However, the optimal parameter values depend on the target problem and genetic operators. Also, the optimal value for each parameter depends on the other parameter values. Accordingly, adjusting these values tends to be tedious and time-consuming. In order to address this problem, adaptive GAs (called AGA, hereafter) which automatically adjust parameters have been proposed. However, most of the existing AGAs adjust only a few parameters at one time. Although several AGAs can adjust many parameters simultaneously, these algorithms need large computation cost.

In this Chapter, we propose self adaptive island model GA (called SAIGA, hereafter) which adjusts multiple parameter values while searching for the solution of the given problem so that the total computation cost is reduced.

We also propose asynchronous SAIGA (called A-SAIGA, hereafter) which eliminates waiting time for synchronization among islands so that search efficiency is improved by reducing idle processor time.

2.2 Related works

De Jong has found appropriate parameter values for five test functions by experiments [9]. The values are 50 to 100 for population size, 0.6 for crossover rate, and 0.001 for mutation rate per bit. This set of values is called the rational parameters, and is widely used for GAs.

In order to find the effective range of parameter values, Goldberg et al. have proposed the control map [14]. The control map indicates the range of parameter values in which search is conducted effectively. In [14], Goldberg et al. proposed and analyzed the control map using the one max problem. But, these analyses are based on some specific problems, and thus a new control map must be made when applied to other problem with different characteristics.

Several self AGAs which automatically adjust the parameter values while searching for the solution have been proposed. Meta-GA is a GA which searches for the optimal parameter values using the results of ordinary GA execution as fitness values [40]. The advantage of meta-GA is that it can adapt any combination of parameter values. But, it requires huge computation power, since repeated searches are needed for the same problem.

Hinterding et al. have proposed an AGA which runs three GAs with the different numbers of individuals in parallel [17]. The process of search is divided into epochs. At each epoch, fitness values of elite individuals are compared, and the number of individuals is changed according to the result. For example, if the GA with the largest number of individuals yielded the best result, all GAs will use a larger number of individuals in the next epoch.

Bäck has proposed an AGA [4] whose individual has its own mutation rate encoded in its gene. Individuals with good mutation rates are expected to survive. However, since individuals with high mutation rates die in high probability, only individuals with low mutation rates tend to survive in the last phase of search. Actually, literature [13] points out that this algorithm shows only low performance in the last phase of search.

Espinoza, et al. have proposed another AGA [11] whose individuals can independently search solutions using local search. In this algorithm, search efficiency per unit number of evaluations has been improved by adapting a ratio between computation costs of crossover/mutation and local search.

An AGA proposed by Krink, et al.[19] determines crossover rate and mutation rate of each individual by its location in 2-dimensional lattice space. Individuals are expected to move toward a location with better parameters. The algorithm keeps diversity of these parameter values by limiting the number of individuals in each lattice.

Kee, et al. has improved meta-GA so that it performs a presearch before performing the main search [18]. After the presearch, the population are classified into several categories. Each subpopulation is investigated using several tens of parameter sets. In the main search, the algorithm classifies the population, and uses a parameter set which was effective in the investigation after presearch.

Tongchim et al. have proposed an AGA which adapts mutation rate and crossover rate [39]. This GA is based on IGA. As the search progresses, increases in average fitness of each island are compared to those of neighbor islands. If an increase of a neighbor island is larger, parameter values are changed according to that of the neighbor island.

Alba et al. measured search efficiencies of synchronous IGA and asynchronous IGA [1]. The results showed that asynchronous IGA is always more efficient than synchronous IGA. Berntsson et al. investigated convergence of population on asynchronous IGA [5]. They revealed that convergence can be sped up by making immigration more frequent. In these studies, parameter values are not adapted.

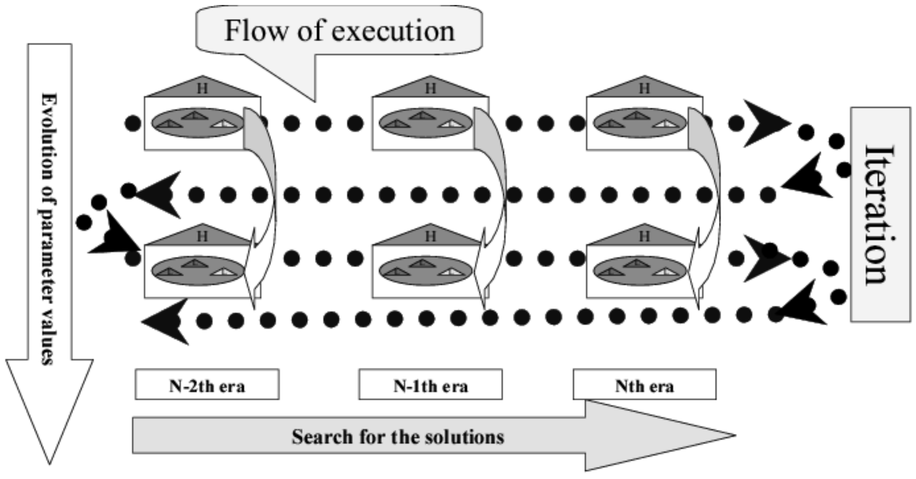

Figure 1. Evolution of set of parameter values in A-SAGA

2.3 Proposed Algorithm

In this section, we first outline the proposed algorithm. Then we explain the notations used in this Chapter. After that, we describe the details of SAIGA and A-SAIGA.

2.3.1 Outline

As described earlier, meta-GA contains two types of GAs. We call the GA which solves a given problem *the low level GA*, and the GA which searches the optimal set of parameter values for low level GA *the high level GA*.

Once a set of parameter values is obtained with meta-GA, similar problems can be efficiently solved using the parameter values. But, if there is only one problem to solve at one time, meta-GA requires a larger computation power to solve the problem.

We previously proposed agent-oriented self-adaptive GA (A-SAGA, hereafter) [24]. The structure of A-SAGA is similar to meta-GA, but its low level GAs are replaced with islands of IGA. A-SAGA assigns different sets of parameters to each island in order to evaluate them. With this mechanism, A-SAGA can evaluate as many parameters as the number of islands in parallel, and thus it is more efficient than meta-GA. But, since A-SAGA requires repeated searches for a given problem, it requires larger computation power if there is only one problem to solve.

A-SAGA divides the whole search into several eras. Each era is a predefined num-

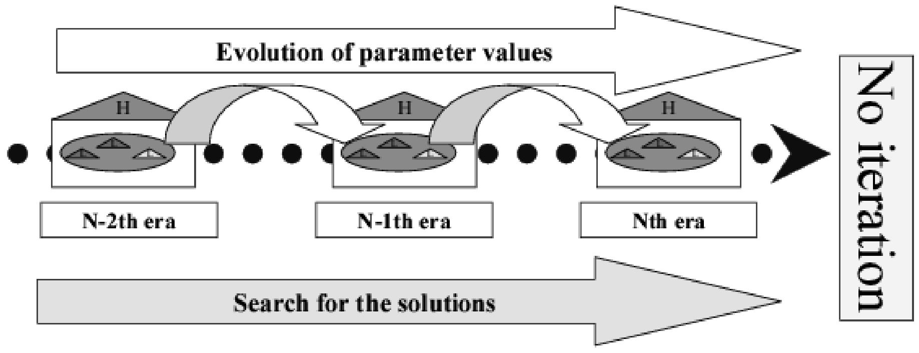

Figure 2. Evolution of set of parameter values in SAIGA

ber of consecutive generations. The high level GA of A-SAGA has different populations for each era, and it searches for an efficient set of parameter values for each era, as shown on Fig. 1. This means that even if a good set of parameter values is found in an era, it is not handed to other eras. On the other hand, SAIGA starts the search of each era using populations of the previous era, as shown in Fig. 2. With this improvement, SAIGA solves a problem without repeating searches from the start.

SAIGA can be executed on a single computer, but it can also be executed on multiple computers by assigning each island to a computer. But, SAIGA requires synchronization between all islands when it generates new sets of parameter values. Thus, if some computers have lower processing power than others, the faster computers have to wait for slow computers during synchronization (see Fig. 3). Asynchronous SAIGA eliminates this synchronization wait so that it reduces idle time of computers and thus search efficiency is improved (see Fig. 4).

2.3.2 Notation

In this Chapter, we use the following notations.

The number of evaluation per era, the number of islands, the maximum population size for high level GA, and the number of generations for the high level GA are denoted as **evaluation_count_per_era, island_max, high_pop_max** and **high_generation_max**, respectively. The number of islands is assumed to be the same as the number of available processors.

These constant values and the parameter values for the high level GA have to be

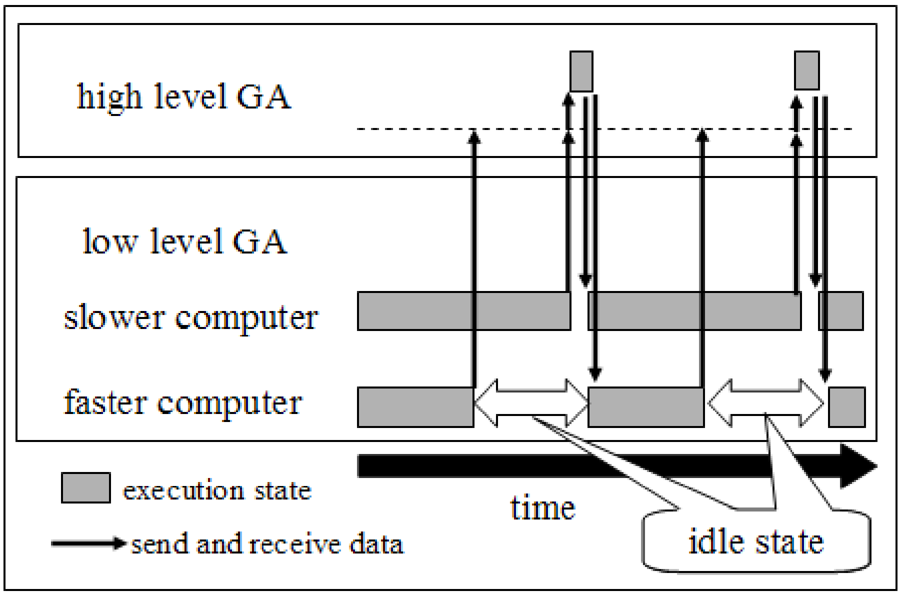

Figure 3. Behavior of SAIGA on computers with deferent performance

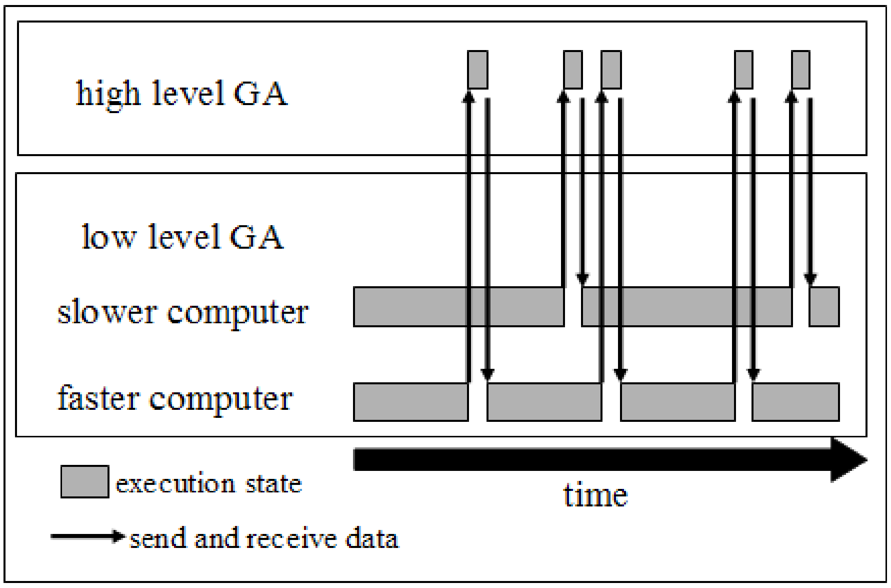

Figure 4. Behavior of A-SAIGA on computers with deferent performance

specified by the user. But once appropriate values for the low level GA are obtained, these values can be used when applied to other problems, as in meta-GA. In the evaluation experiments, we obtained one set of values using a certain problem, and used the same values for other problems.

An individual consists of a chromosome and a fitness value. The chromosome and fitness value for a high level GA individual p are denoted as $p.\vec{x}$ and $p.fit$, respectively. When adapting l parameter values, $p.\vec{x}$ is an l-dimensional vector. The population and population size for a high level GA at t-th generation is denoted as P_t and $|P_t|$, respectively. The set of parameter values for a high level GA is denoted as $\vec{w}$. The states of immigration and communication to island i are denoted as $E[i]$ and $R[i]$, respectively.

A low level individual q consists of a chromosome $q.\vec{x}$ and a fitness value $q.fit$. The population on island i at g-th generation is denoted as Q_g^i. Note that g is different on each island. The set of parameter values assigned to island i is denoted as $\vec{v}^i$. Immigration to and from i are denoted as $q^i_{\mathbf{immig}}$ and $q^i_{\mathbf{emig}}$, respectively. f^i is a variable expressing the elite fitness value for island i.

f is a function which projects a chromosome $p.\vec{x}$ for a high level GA to a set $\vec{v}^i$ of parameter values. Its inverse function is denoted f^{-1}.

2.3.3 SAIGA

SAIGA is a combination of a high level GA, which is a Simple GA, and a low level GA, which is an IGA. The parameter values which can be adapted by SAIGA are limited to values which can be evaluated within each island (such as population size, mutation rate, choice of crossover method and choice of selection type), and parameters which cannot be evaluated within an island (such as the number of islands or immigration rate) cannot be adapted.

The pseudo code for the high level GA and low level GA of SAIGA are shown on Algo. 1 and Algo. 2, respectively.

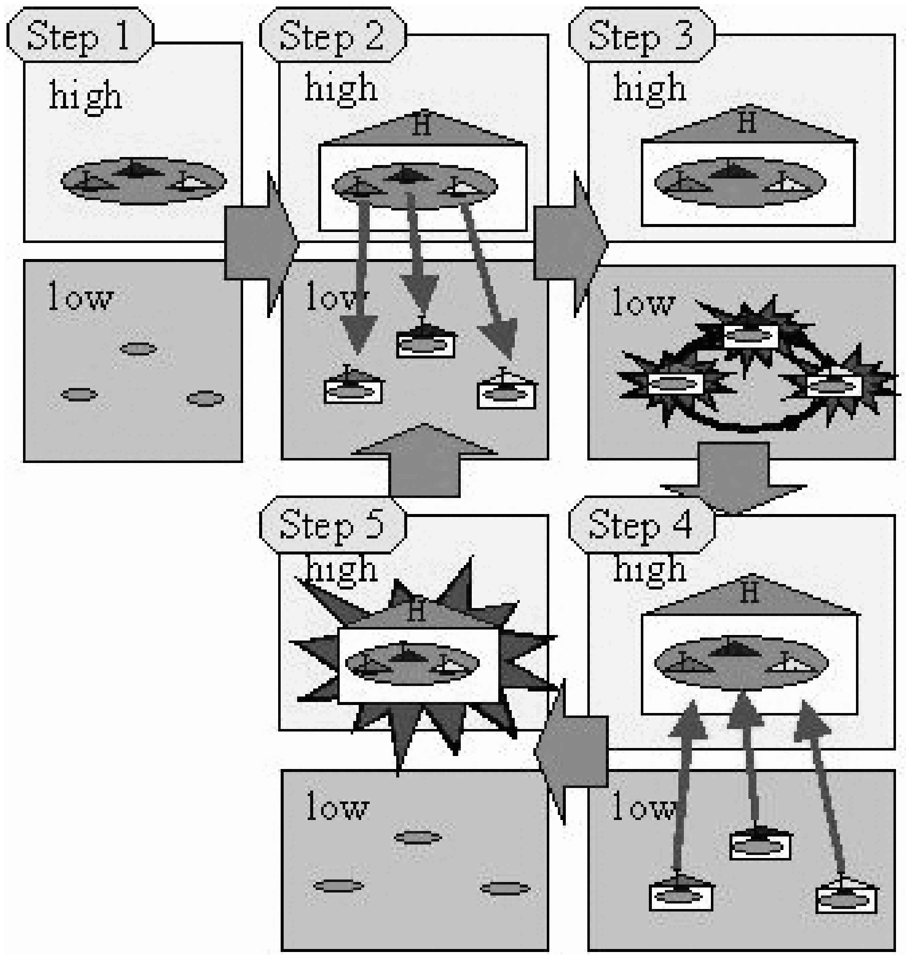

Figure 5. Overview of SAIGA

Algorithm *HighLevelGA*()

```
1   $t$ :=0; //$t$: Generation counter for high level GA
2   $j$:=0; //$j$: The number of islands which finished search for the era
3   For $i$ := 1 to island_max Do
4       Island($i$) start the process
        Substitute $q^i_{\mathbf{immig}}$ for a dummy value
5       Generate a set $\vec{v}^i$ of parameter values using random values
6       Island($i$) Send ($\vec{v}^i, q^i_{\mathbf{immig}}$) to the process
7   Next
8   While $t <$ high_generation_max Do
9       $t := t + 1$; $P_t := \emptyset$;
10      For $i := 1$ to island_max Do
11          $R[i] := 0$; //$R[i]$ is set 1 iff both a fitness value and a set of parameter values are
            received from island $i$, and set 0 otherwise.
12      Next
13      While true Do
14          If ($\vec{v}^i$, $fit$, $q^i_{\mathbf{emig}}$, $i$) are received fromIsland($i$) and $R[i] = 0$ Then
            // $q^i_{\mathbf{emig}}$: an immigrated individual from island $i$
15              $j := j + 1$;$R[i] := 1$;
16              $p.\vec{x} := f^{-1}(\vec{v}^i)$; // a set of parameter values is converted to a chromosome
17              $p.fit := fit$; $P_t := P_t \cup \{p\}$;
18              If $i$ = island_max Then
19                  $q^0_{\mathbf{immig}} := q^i_{\mathbf{emig}}$;
                    // $q^i_{\mathbf{immig}}$: retains immigrating individual to island $i$
20              Else $q^{i+1}_{\mathbf{immig}} := q^i_{\mathbf{emig}}$;
21              EndIf
22          EndIf
23          If $j$:=island_max Then Break ; EndIf
24      EndWhile
25      $P'_t := HighPopGeneticOperation(P_t, \vec{w})$;
        //Mutation, crossover and selection operators are applied based on the set of parameter
        $\vec{w}$, and substitute $P'_t$ for the result.
26      For $i$:=1 to island_max Do
27          Choose an element of $P'_t$ randomly, and substitute $p$ for it. $P'_t$ := $P'_t$ - $\{p\}$;
```

```
28          $\vec{v}^i := f(p.\vec{x})$; //Convert to a set of parameter values
29          Send ($\vec{v}^i, q^i_{\mathbf{immig}}$) to process *Island*($i$).
30      **Next**
31  **EndWhile**
32  Terminate all processes
**End** /**HighLevelGA**/
```

Algorithm 1. Pseudo code of high level GA in SAIGA

```
**Algorithm** *Island*($i$)
1   $f^i$ := 0; // The highest fitness value in island $i$
2   $g$:= 0; // $g$:The number of generations for island $i$
3   Initialize all individuals $q \in Q^i_g$ in island $i$
4   **While** true **Do**
5       **While** true **Do**
6           **If** ($\vec{v}^i, q^i_{\mathbf{immig}}$) are received from *HighLevelGA*() **Then Break** ; **EndIf**
            // $q^i_{\mathbf{immig}}$ retains immigrating individual to island $i$
7       **EndWhile**
8       $(Q^i_{g+1}, fit, f^i, q^i_{\mathbf{emig}})$ :=
            *ExecuteIsland*( $Q^i_g$, $\vec{v}^i$, $f^i$, $q^i_{\mathbf{immig}}$, $i$);
        // Evolve $Q^i_g$ using **evaluation_count_per_era** of evaluations
        // $fit$: fitness value of set $\vec{v}^i$ of parameter values
        // $q^i_{\mathbf{emig}}$:Immigration individual to other island
9       Send $(\vec{v}^i, fit, q^i_{\mathbf{emig}}, i)$ to *HighLevelGA*() process
10      $g$:= $g + 1$;
11  **EndWhile**
**End** /**Island**/
```

Algorithm 2. Pseudo code of low level GA

First, SAIGA generates random sets of parameter values for each island (lines 5–6 on Algo. 1). Each island performs GA operations using a predetermined number of

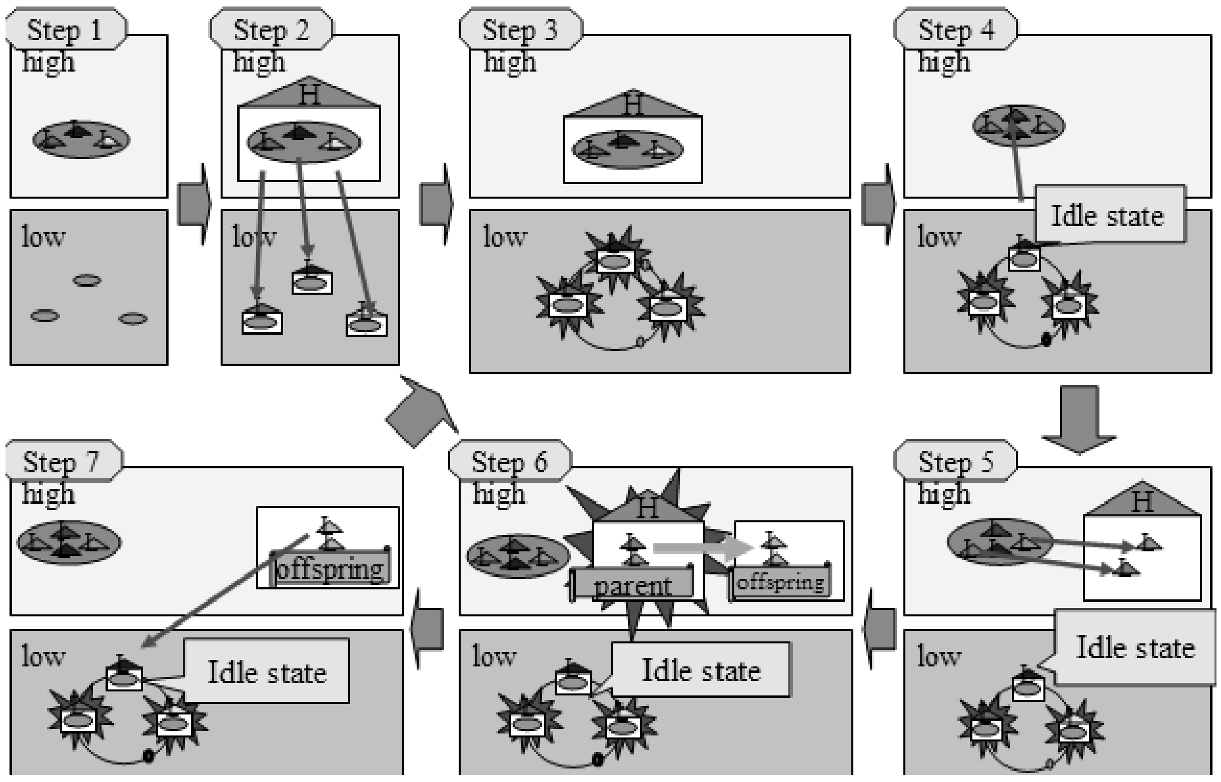

Figure 6. Overview of A-SAIGA

evaluations (lines 6–8 on Algo. 2). Then, the search efficiencies are observed for each island, and are returned as the fitness value of the set of parameter values for the island. The high level GA evolves the set of parameter values using the obtained fitness values (lines 14–25 on Algo. 1). Newly generated sets of parameter values are assigned to each island (lines 29 on Algo. 1). Each island continues search without initializing its population. The search is performed using a predetermined number of evaluations, again. Each of such cycle is called an era.

2.3.4 A-SAIGA

The pseudo code for the high level GA of A-SAIGA is shown on Algo. 3.

Introducing the generation model of the Steady-state GA

The high level GA for SAIGA uses the mechanism of SGA. We introduced the mechanism of the Steady-state GA in order to make SAIGA asynchronous (lines 11–31 in Algo. 3). The Steady-state GA replaces individuals partially at every generation. In the proposed method, every time the high level GA receives data from one island, it generates one new set of parameter values.

```
Algorithm HighLevelGA()
1   $P_0 := \emptyset$;$t := 0$; //$t$ retains the number of generations for high level GA
2   For $i := 1$ to island_max Do
3       Start process Island($i$)
4       Substitute $q^i_{\textbf{immig}}$ for a dummy value
5       Generate a set $\vec{v}^i$ of parameter values using random values
6       $E[i] := 0$; //$E[i]$ retains state of immigration to island $i$
7   Next
8   While $t <$ high_generation_max Do
9       $t := t + 1$;
10      While true Do
11          If ($\vec{v}^i$, $fit$, $q^i_{\textbf{emig}}$, $i$) is given from Island ($i$) process Then Break ; EndIf
12          // $q^i_{\textbf{emig}}$: an immigrated individual from island $i$
13      EndWhile
14      If $|P_{t-1}| >$ high_pop_max Then
15          $P_{t-1} := P_{t-1} - \{oldest\_of(P_h)\}$;
            // Remove the oldest individual
16      EndIf
17      $p.\vec{x} := f^{-1}(\vec{v}^i)$; // // a set of parameter values is converted to a chromosome
18      $p.fit := fit$; $P_t := P_{t-1} \cup \{p\}$;
19      If $i$ = island_max Then
20          $q^0_{\textbf{immig}} := q^i_{\textbf{emig}}$; $E[0] := 1$;
            // $q^i_{\textbf{immig}}$: retains immigrating individual to island $i$
21      Else
22          $q^{i+1}_{\textbf{immig}} := q^i_{\textbf{emig}}$; $E[i + 1] := 1$;
23      EndIf
24      $p^{\textbf{parent1}} := HighIndivSelection(P_t, \vec{w})$;
        // A parent individual is obtained by selection
25      $p^{\textbf{parent2}} := HighIndivSelection(P_t, \vec{w})$;
26      $p^{\textbf{offspring}} := HighIndivGeneticOperation(p^{\textbf{parent1}}, p^{\textbf{parent2}}, \vec{w})$;
```

```
        // A child individual is obtained by crossover and mutation
27      $\vec{v}^i := f(p^{\mathbf{offspring}}.\vec{x})$; // Convert to a set of parameter values
28      If $E[i] = 1$ **Then** $E[i] := 0$;
29      **Else** Substitute $q^i_{\mathbf{immig}}$ for a random value
30      **EndIf**
31      Send $(\vec{v}^i, q^i_{\mathbf{immig}})$ to process *Island*$(i)$
32  **EndWhile**
33  Terminate all processes
**End** /**HighLevelGA**/
```

Algorithm 3. Pseudo code of high level GA in A-SAIGA

Introducing FIFO deletion

Introducing the generation model of the Steady-state GA leads to mixed individuals, some recently evaluated and some evaluated many generations before, in the population. This is unfavorable, since individuals evaluated long before may survive for many generations even though the evaluation criteria for the high level GA change as the search proceeds. Accordingly, we also introduced FIFO deletion proposed by De Jong [10] into the proposed method (line 15 in Algo. 3). This is a method which deletes the oldest individual if the population size exceeds a predefined limit.

Evolution of individuals in asynchronous SAIGA

Whether a search can be performed efficiently using a set of parameter values depends on the state of the individuals. If we can make the populations of the islands similar to each other, similar search performance should be obtained using one same set of parameter values. In SAIGA, the populations of the islands are made similar by the immigration mechanism. Also, the number of evaluations for each island is made identical by synchronization. But, A-SAIGA does not synchronize between islands, and this may somewhat affect to the search performance. But, it is known that if immigration is frequent, the populations of the islands becomes similar in asynchronous IGA with different number of evaluations for each island [5]. Thus, the influence of eliminating synchronization can be minimized by making immigration frequent.

2.4 Experiment

2.4.1 Experimental Conditions

In order to evaluate the proposed method, we conducted experiments under the following conditions.

For the high level GA, we used a mutation operator using the Gauss distribution, crossover between each parameter, and roulette selection. The parameter settings for the high level GA are as follows: crossover rate, mutation rate, scaling rate and the number of islands are 0.8, 0.6, 1:5 and 10, respectively. In the experiments, we used Simple GA as the low level GA of the proposed method for simplicity of analysis. The adapted parameters for the low level GA are the crossover rate, mutation rate, the number of individuals in an island, and tournament size. De Jong's standard parameter values include the crossover rate, mutation rate, and population size. We added the tournament size, which adjusts the selection pressure, to these parameters.

The individuals in the high level GA are coded as follows. Since the number of parameters in the experiment is four, the number of elements in chromosome $a.\vec{x}$ for the high level GA individual a is four. Each element $a.x_i$ in $a.\vec{x}$ is expressed in 16-bit fixed point decimal number between 0 and 1 $(0 < a.x_i < 1, a.x_i \in \Re)$.

Each $a.x_i$ is transformed into the parameter value space by using following function:

$$\begin{aligned}
n_i &= \lfloor 2 \cdot \exp(8 \cdot a.x_1 \cdot \log(2)) \rfloor. \\
s_i &= \begin{cases} \lfloor a.x_2 \cdot 8 + 2 \rfloor & \text{if } n_i > \lfloor a.x_2 \cdot 8 + 2 \rfloor, \\ 2 & \text{if } n_i \leq \lfloor a.x_2 \cdot 8 + 2 \rfloor. \end{cases} \\
c_i &= a.x_3. \\
m_i &= 0.00005 \cdot \exp(a.x_4 \cdot \log(1/0.0001)).
\end{aligned}$$

where n_i, s_i, c_i, and m_i denote population size, tournament size, crossover rate, and mutation rate respectively, after transformation.

We used the following problems to evaluate the proposed method: the minimization problem of the Rastrigin function, lin105 [26] of Traveling Salesman Problem (TSP), the deception problem, and la38 [20] of Job-Shop Scheduling Problem.

The results are average values of 300 times of executions. The number of evaluations for each problem is 6.4×10^6. We compared the proposed method with following

GAs: a SGA with De Jong's standard parameter values, a SGA with manually optimized parameter values, an IGA with De Jong's standard parameter values, and an IGA with manually optimized parameter values. Since De Jong's rational parameter values do not include values for selection, we used tournament selection with a size of 2.

We also measured processing time for one generation of execution for SAIGA and A-SAIGA.

Under the experimental conditions, we executed GAs on 10 computers with 400 MHz to 2.6 GHz processor speed.

2.4.2 Considerations

In this subsection, we describe our considerations about the experiments.

The results of our experiments are shown in Table 1. Hereafter, SGA with De Jong's standard parameter values, SGA with manually tuned parameter values, IGA with De Jong's standard parameter values, and IGA with manually tuned parameter values are denoted as SGA (DeJong), SGA (tuned), IGA (DeJong), and IGA (tuned), respectively.

Comparison between the proposed method and A-SAGA

The proposed method gives better results than A-SAGA when the proposed method and A-SAGA are applied to either problem.

In this experiment, A-SAGA executes 100 sets of population initialization and other genetic operations using 6.4×10^4 evaluations. On the other hand, the proposed method executes population initialization only one time and continues the explorations using 6.4×10^4 evaluations. Because of this difference, the proposed method seems to have given better results in these problems.

Comparison between SGA and IGA

A-SAIGA gives better results than SGA (DeJong) and IGA (DeJong) when A-SAIGA and SGA are applied to TSP and JSP. However, SGA (DeJong) and IGA (DeJong) have given better results than the proposed method when applied to the problem of Rastrigin function. These results show that De Jong's rational parameter values are suitable for benchmark problems such as the problem of the Rastrigin function. The proposed method is not effective if the suitable parameter values for a problem are

	Rastrigin		deception	
	optimal 0		optimal 0	
	best	mean	best	mean
SGA(Dejong)	0	0	1	3.5
SGA(tuned)	0	0	2	3.3
IGA(Dejong)	0	0	0	2.26
IGA(tuned)	0	0	0	0.25
A-SAGA	10	20.52	0	0.66
SAIGA	0	0	0	0.47
A-SAIGA	0	0.02	0	0.53
	TSP(lin105)		JSP(la38)	
	optimal 14379		optimal 1196	
	best	mean	best	mean
SGA(Dejong)	14434	15705	1216	1289
SGA(tuned)	14475	15175	1203	1227
IGA(Dejong)	14483	15489	1226	1261
IGA(tuned)	14401	14932	1205	1218
A-SAGA	25197	28366	1228	1249
SAIGA	14464	15391	1205	1242
A-SAIGA	14486	15336	1207	1234

Table 1. Evaluation results

known in advance. However, A-SAIGA gives better results for more realistic problems like TSP and JSP which use more complex encoding and operators.

In the Deception problem, IGA (tuned) gives far better results than SGA (tuned). This seems to be because good chromosome fragments found on islands are assembled to make a good result in IGA (tuned). But, the result of IGA (DeJong) for the Deception problem is about 2, which is not close to the optimal value. This seems to be because De Jong's standard parameter values are not suitable for solving the Deception problem. A-SAIGA gives results close to 0 in this problem, which are better than the results with IGA (DeJong). This shows that the A-SAIGA acquires parameter values better than IGA (DeJong). In TSP, JSP and the Rastrigin function problem, the results of our method were close to those of SGA (tuned) and IGA (tuned).

The results confirmed that A-SAIGA gives better results than the use of De Jong's rational parameter values. Also, the performance of A-SAIGA is close to that of SGA with manually tuned parameter values for various problems,

Comparison between SAIGA and A-SAIGA

There were no large differences in performance between SAIGA and A-SAIGA, except TSP and JSP. For TSP and JSP, A-SAIGA showed better performance than SAIGA. This seems to be for the following reasons. SAIGA gives evaluation counts to islands equally, though A-SAIGA does not. Thus, in A-SAIGA, the islands executed on higher performance computers are given evaluation counts much larger than the other islands. This leads to prominent local search on such islands. Since local search is effective for TSP and JSP, A-SAIGA has given better results in these problems.

2.4.3 Comparison of processing time

We measured processing time of one generation in SAIGA and A-SAIGA. The results are shown on Table 2.

	SAIGA	A-SAIGA
Rastrigin	6.6	2.3
deception	1.4	0.9
TSP	16.4	4.1
JSP	111.6	37.2

Table 2. Comparison of processing time in one era (milliseconds)

These results show that the processing time of A-SAIGA is always shorter than those of SAIGA, and this also means that A-SAIGA has a reduced waiting time for synchronization between islands.

The results show that the processing time of A-SAIGA is one-third that of SAIGA for the Deception problem, while the processing time of A-SAIGA is two third that of SAIGA on other problems. This seems to be because the evaluation time in the Deception problem is much shorter than in the other problems.

In order to examine this hypothesis, we added dummy processing time to the evaluation process of the Deception problem, and measured the overall processing time while varying the amount of dummy processing time. The results are shown in Table 3.

Function calls	0	1	10	100	1000	10000	100000
Processing time	0.47	0.37	0.35	0.38	0.38	0.26	0.24

Table 3. Number of function calls and ratio of processing time

The result show that the more we increase the dummy processing load, the greater the difference in processing time between A-SAIGA and SAIGA becomes. Therefore difference in processing time of the evaluation function causes the difference in the processing time ratio for the Deception problem and other problems.

This result shows that the processing time of A-SAIGA is always shorter than that of SAIGA. But table 1 shows that the search efficiency of SAIGA gives better than A-SAIGA on a part of the result of Rastrigin function. We believe this is because SAIGA gives evaluation counts to islands equally, but A-SAIGA does not. We are planning to examine this in the future works,

2.4.4 Number of adapted parameter value

We increased the number of adapted parameter values for the low level GA to six, and performed experiments using the Rastrigin function in SAIGA. The increased parameter values are related to the selection method (choice either Roulette method or Tournament method) and the parameter which relates to selection pressure when Roulette method is chosen. Hereafter, this is denoted as **TR6**. We compared **TR6** to SAIGA with the Tournament method (**T4**, hereafter). We also compared **TR6** to SAIGA with the Roulette method (**R6**, hereafter). **T4** and **R4** are adapted in four parameter values, respectively. The results are the averages of 30 executions.

The search efficiency results are as follows: **T4**, **R4**, and **TR6** are 0.033, 0.565, and 0.133, respectively. In this case, **T4** performs better than **R4**. This result shows that the Tournament method is suitable for the Rastrigin function. **TR6** gives better performance than **R6**, and the performance of **TR6** is close to that of **T4**. This is a result of the adjustment of the parameter value for selection when six parameter values are adapted. But, if the number of parameter values is increased, parameter values search efficiency is expected to decline, since the search space of the parameter values extends. It seems that this is the reason why **TR6** did not perform as well as **T4**.

2.4.5 Obtained Parameter Values Using The Proposed Method

To evaluate the parameter values obtained using SAIGA and A-SAIGA, we compared the parameter values after the algorithms converged, De Jong's rational parameter values, the manually tuned parameter values, and Grefenstette's parameter values [15]. The results are averages of 300 executions. The results of this experiment are shown in Table 4. where, the mutation rate, crossover rate, tournament size, and population size are denoted as mu, cr, sc, and po.

	mu	cr	sc	po	mu	cr	sc	po
DeJong	0.001	0.6	2	50				
Grefenstette	–	0.5	-	50				
	Rastrigin				TSP			
tuned	0.003	0.58	4	4	0.01	0.55	4	16
SAIGA	0.014	0.59	3	12	0.016	0.53	5	49
A-SAIGA	0.009	0.62	4	40	0.014	0.50	4	46
	deception				JSP			
tuned	0.2	0.5	2	256	0.02	0.55	4	256
SAIGA	0.064	0.50	5	58	0.029	0.49	5	45
A-SAIGA	0.062	0.53	5	62	0.051	0.47	5	49

Table 4. Parameter obtained using the proposed method

This Table 4 shows that the parameter values obtained using SAIGA and A-SAIGA are close to the manually tuned parameter values. Also, the population size obtained is 50 and the crossover rate obtained is close to the crossover rate of Grefenstette's parameter values. Therefore, we believe SAIGA and A-SAIGA successfully adapt parameter values to these problems.

2.4.6 Number of islands

To evaluate the scalability of the proposed method, we determined the search efficiencies of the proposed methods with the number of islands varied. By adjusting the computation time in each experiment, we allocated computation power to computers in accordance with the number of islands for the execution of the methods. The results are shown in Table 5. SAIGA and A-SAIGA achieve high search efficiency when the number of islands is increased.

Number of islands		10	20	50	100
Calculation time (min.)		1	2	5	10
SAIGA	Rastrigin	1.067	0.433	0.167	0.133
A-SAIGA	Rastrigin	0.400	0.133	0.067	0.067
SAIGA	deception	1.406	0.173	0.000	0.000
A-SAIGA	deception	0.000	0.000	0.000	0.000
SAIGA	TSP	15618	15283	14754	14627
A-SAIGA	TSP	15254	15092	14662	14532
SAIGA	JSP	1294	1287	1267	1256
A-SAIGA	JSP	1248	1242	1238	1237

Table 5. Number of islands and search efficiency

2.5 Conclusion

In this chapter, we proposed self adaptive island model GA (SAIGA) which can adjust multiple parameter value. By reducing the number of iterations in our previous work called A-SAGA (proposed agent-oriented self-adaptive GA), Computation cost with SAIGA can be lowered than that with A-SAGA. We also proposed asynchronous SAIGA (A-SAIGA) which eliminates waiting time for synchronization among islands in SAIGA. In order to evaluate effectiveness of the proposed method, we applied the proposed method to various problems and measured search efficiencies. Our experiments showed that the proposed method can find appropriate parameter values for various problems, and the performance of the proposed method was close to that of simple GA with manually optimized parameter values. Also, A-SAIGA reduced waiting time for synchronize among islands to a great extent.

3. Application of Island Model GA to Routing Problem in Mobile Ad Hoc Network

3.1 Introduction

Video and other streaming data are important applications of mobile ad hoc network (MANET). In construction of a multicast tree, it is desirable to minimize relaying costs and maximize the number of nodes which receive the data, satisfying QoS constraints such as bandwidth and delay. There is a study on routing techniques to construct a multicast tree which simultaneously satisfies two or more QoS constraints and is semi-optimal for a given objective [21]. However, this method supposes a single computation node which knows topology information of the whole MANET and computes multicast tree in a centralized manner.

In this chapter, we propose a new QoS multicast routing method for MANET called HQMGA (hierarchical QoS multicast routing using GA in MANET). HQMGA allows MANET nodes to dynamically construct a semi-optimal multicast tree satisfying several QoS constraints at the same time for any given objectives (e.g. power consumption and tree stability, etc). We designed and developed protocols to run on mobile terminals in MANET. We suppose that mobile terminals, such as PDA and cell phone, are used in MANET. The computation capability of these mobile terminals is scarce compared with a fixed terminal. Thus, HQMGA reduces the cost for constructing multicast trees consisting of many mobile nodes so that the computation and communication can be executed within capability of each mobile node. To do so, it divides MANET into multiple clusters, and makes them construct sub-trees covering their clusters in parallel and computes a tree which connects the sub-trees of all clusters. Consequently, one semi-optimal multicast tree is constructed with them. Moreover HQMGA abstracts the topology information spanning all clusters so that a node can compute the tree connecting all clusters within small computation power and communication amount.

3.2 Related works

In order to realize multimedia communication on MANET, the many QoS routing protocols have been proposed so far. As unicast routing methods which treat a single QoS constraint, [7, 22, 23, 30, 41] have been proposed. [7] is a method of QoS unicast routing for discovering the path to satisfy the specified bandwidth (or delay). This method floods route request messages including a fixed number of logical tickets, and investigates whether paths satisfying given constraints exist. This method can suppress flooding range is limited since logical tickets in a message are divided when an intermediate node forwards the message to its neighbors. [23, 41] are methods which discover paths satisfying specified bandwidth. The former method adopts pro-active routing, and the latter does reactive routing based on AODV [25] [1]. These methods divide the available bandwidth by time slots using TDMA. The method in [23] assigns unused time slots to a requested data stream until constraints are satisfied. On the other hand, in [41], information on available bandwidth to the next hop is added to messages at every relay node so that the destination nodes can check available bandwidth in paths by the information in received messages, and reply to the messages whose available bandwidth information satisfy given constraints. CEDAR [30] is a unicast routing method, where a set of nodes is divided into several areas, and paths satisfying QoS constraints are computed area by area like OSPF [8] [2]. In CEDAR, a network core satisfying required bandwidth is dynamically constructed and maintained. A source node and destination node can efficiently compute a path with required bandwidth through the core. [22] is a QoS multipath routing method using two or more path simultaneously. This method first investigates available bandwidth on various paths by limited flooding using logical tickets like [7]. The required bandwidth is reserved by combining two or more paths by summing up their available bandwidths.

Several QoS multicast routing methods on MANET have been proposed [6, 31]. MCEDAR [31] is an enhanced version of CEDAR to treat multicast routing. In MCEDAR [31], a mesh network consisting of nodes belonging to multicast groups is constructed

[1] AODV is reactive routing protocol. In AODV, route request message is flooded toward the destination node. The destination node replies to the message by sending the route reply message back to the sender node. Relay nodes create routing table by this message exchange.

[2] OSPF is a link-state routing protocol. An OSPF network is divided into areas. OSPF executes routing at two levels : inside area and backbone area. This makes routers reduce the amount of information which they need to maintain.

and used as a network core. By distributing data via the mesh, multicast tree is computed efficiently. AQM [6] is a method of multicast routing which satisfies bandwidth constraint. AQM constructs a multicast tree with specified bandwidth on demand by three phases (Demand – Response – Reservation). Here, each node exchanges the reservation information on bandwidth with adjacent nodes at regular time intervals. By inserting this information in route request messages, each node can find paths which have required bandwidth.

As mentioned above, various QoS routing protocols for MANET have been proposed. However, these existing methods are not designed to compute multicast tree satisfying two or more QoS constraints at the same time or optimized for a given objective such as minimization/maximization of power consumption, stability, the number of receivers, and so on.

3.3 Proposed Method

In this section, after describing overview and assumptions of the proposed method, we define the target problem for deriving multicast tree and give details to solve the problem.

3.3.1 Overview and Assumptions

The proposed method uses a genetic algorithm (GA) to compute semi-optimal multicast tree. The merits using GA are as follows: (1) GA can quickly recompute a new multicast tree for slightly changed topology of MANET by using solution candidates used for the last computation; and (2) GA can solve combinatorial optimization problem for an arbitrary objective function in a short time. However, GA requires large computation power in general.

Moreover, in order to correspond to change of topology, a multicast tree is reconstructed periodically.

However, in large-scale MANET, it is difficult for even GA to compute the semi-optimal multicast tree in a centralized manner due to costs of computation and communication. So, in the proposed method, nodes in MANET are divided into multiple clusters like OSPF and sub-trees are computed for those clusters (we call each subtree *local tree*, hereafter), in order to reduce costs of gathering topology information and computation of trees. A tree connecting sub-trees of clusters (hereafter, we call it *global tree*) is also computed. The tree consisting of local trees of all clusters and the global tree is used as the multicast tree. Since the isolated node etc. exists in MANET, the node which cannot be connected exists. So, by removing the node which does not satisfied criteria from multicast tree, the proposed method constructs the multicast tree consisting of nodes which are satisfied criteria. We define problems for constructing local trees and global tree in Section 3.3.2.

The proposed method targets MANET with the following assumptions.

Mobile nodes are pedestrians, and thus they move at speed of around 4 km/hour. Nodes transmit and receive small video and/or voice data. Each node can send and receive packets through wireless network interface, where we suppose to use IEEE 802.11 as MAC protocol. Each node can perform a career sense, but does not exchange RTS/CTS packets. Communication is done by broadcast, that is, when each node broadcasts a packet to its radio range, nodes in its radio range receive the packet. Here, each upper-stream node transmits the packets to its downstream nodes at once (not separately). The protocol as MAC layer protocol is the IEEE802.11. Each node can know its own moving speed and a rough distance to other nodes from the strength of their radio wave signals.

3.3.2 Problem Definition

We use the following notation.

$G = (V, E)$ denotes a weighted undirected graph which expresses topology of MANET at a certain point. Here, V and E denote the set of nodes and the set of links, respectively. Unique ID (integer value) is given to each node. Only when two nodes $v_i, v_j \in V$ exist in their common radio range, we suppose that a link $e_{ij} \in E$ exists between them. $s \in V$ denotes a source node (a node transmitting multimedia data). $U \subseteq V$ denotes a set of *user nodes* which require reception of multimedia data. Each user node $u \in U$ sends a request message $req_u(c, Br, Dr)$ where c, Br and Dr denote a content ID, required bandwidth and allowable delay time, respectively. We suppose that the values of Br and Dr are integer and determined depending on the content c so that one multicast tree can be constructed for each content. $T = (V', E')$ denotes a multicast tree which connects a source node s and user nodes of U, although all nodes in U may not be contained in the tree. We treat the problem of finding the multicast tree which satisfies the constraints about bandwidth and delay time, and maximizes a given evaluation function $f(T)$. However, as explained in the next sub-section, we use different evaluation functions for global tree and local tree. Below, we use an evaluation function taking into account three criteria: the number of user nodes which can receive data stream, tree stability, and latency. However, note that we can define any evaluation function suitable for a target environment.

Problem of Finding Global Tree

The purpose of this problem is to find the global tree T' which maximizes the evaluation function $f\_Global(T')$ defined below (see Fig. 7). We assume that the whole network is divided to clusters, and two representative nodes called *cluster head* and *sub cluster head* are selected for each cluster, and a node called *top cluster head* is selected from all cluster heads, in advance. These nodes are responsible for computation of global/local trees. Techniques for cluster division and node selection are explained later.

$$\begin{aligned} f\_Global(T') = \alpha \cdot \mathrm{Satisfied\_User\_Global}(T') \\ - \beta \cdot \mathrm{Delay\_Global}(T') - \gamma \cdot \mathrm{Instability\_Global}(T') \end{aligned}$$

Here, $\mathrm{Satisfied\_User\_Global}(T')$ is the function representing the number of user nodes included in the tree satisfying the constraints, $\mathrm{Delay\_Global}(T')$ is the penalty function on delay constraint[3], and $\mathrm{Instability\_Global}(T')$ is the function on tree's instability. α, β and γ are positive weighting coefficients.

The delay constraint must be satisfied for the sum of a global tree's delay and local tree's delay. So, we use a coefficient $\delta (0 < \delta < 1)$ so that the global tree satisfies the delay constraint of δDr. In the evaluation function, the bandwidth constraint is not used (as described in Section 3.3.4).

$\mathrm{Satisfied\_User\_Global}(T')$ is defined as

$$\mathrm{Satisfied\_User\_Global}(T') = \sum_{C_i \in \mathscr{C}} \mathrm{deliver_c}(C_i, T')$$

where

$$\mathrm{deliver_c}(C_i, T') =$$

$$\begin{cases} 1 & \text{if } \mathrm{path_c}(C_i) \neq \emptyset \wedge \ \mathrm{Delay_c}(\mathrm{path_c}(C_i)) \leq \delta Dr, \\ 0 & \text{otherwise.} \end{cases}$$

$\mathscr{C}$ is a set of the cluster on the network, $path_c(C_i)$ is a path from cluster head of the cluster including source node s to cluster head of cluster C_i via some intermediate nodes. If such a path does not exist, then $path_c(C_i) = \emptyset$. $Delay_c(path)$ is an estimated delay on $path$ (we assume that delay can be estimated by the path length and so on).

The function $\mathrm{Delay\_Global}(T')$ is a criterion for delay and expressed as ratio of the number of hops from the top cluster head to the corresponding cluster head, over the number of all nodes, and is defined as follows.

[3]In order to enhance the search efficiency of GA, the penalty function (function which reduces it somewhat, instead of setting an evaluation value to 0 when not satisfied condition) about each constrained condition and the instability of a multicast tree is used.

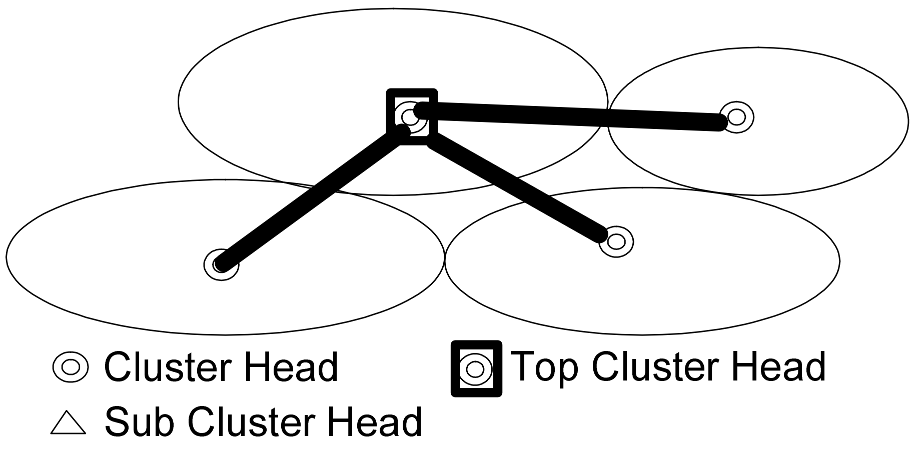

Figure 7. Global Tree

$$\mathrm{Delay\_Global}(T') = max_{C_i \in \mathscr{C}} |\mathrm{path_c}(C_i)| / |V|$$

Here, $|\mathrm{path_c}(C_i)|$ represents the number of hops in $\mathrm{path_c}(C_i)$. $\mathrm{path_c}(C_i)$ is the route between the cluster head of a cluster which s belongs to, and the cluster head of the cluster C_i, by way of several intermediate nodes. If there is no such path, $hopcount(C_i)$ is defined as 0.

The function $\mathrm{Instability\_Global(T')}$ is a criterion for tree instability and defined as the ratio of highest speed of nodes on tree T' over the maximum speed $SPEED$, as follows.

$$\mathrm{Instability\_Global}(T') = max_{C_i \in \mathscr{C}} (max_{v \in \mathrm{path_c}(C_i)} \frac{speed(v)}{SPEED})$$

3.3.3 Problem of Optimizing Local Tree

For local tree T''_i of cluster C_i with user nodes U_i, we maximize evaluation function $f\_Local(T''_i, U_i)$ (see Fig. 8) defined below.

$$f\_Local(T''_i, U_i) = \mathrm{Satisfied\_User\_Local}(T''_i, U_i) - \epsilon \cdot \mathrm{Instability\_Local}(T''_i, U_i)$$

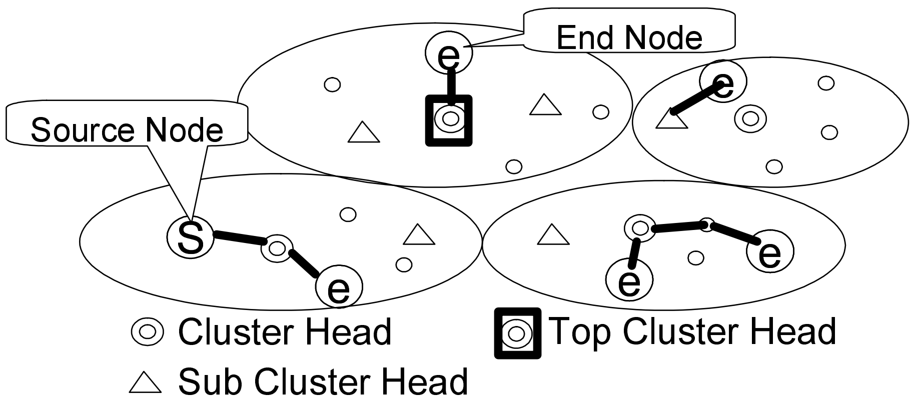

Figure 8. Local Tree

Here, $\mathrm{Satisfied\_User\_Local}(T_i'', U_i)$ is a criterion regarding to the number of user nodes, and $\mathrm{Instability\_Local}(T_i'', U_i)$ is a criterion for tree instability. ϵ is a weighted coefficient.

Function $\mathrm{Satisfied\_User\_Local}(T_i'', U_i)$ represents the number of user nodes which satisfy the following constraints.

$$\begin{aligned}\mathrm{Band}(h_i, u, T_i'') &\geq B_r \\ \mathrm{Delay}(h_i, u, T_i'') &\leq (1-\delta)D_r\end{aligned}$$

Here, $\mathrm{Band}(h_i, u, T_i'')$ is the minimum bandwidth in the path from cluster head h_i to node u on multicast tree T_i'', $\mathrm{Delay}(h_i, u, T_i'')$ is the total delay on this path, and $(1-\delta)D_r$ is allowable delay in the cluster.

$\mathrm{Instability\_Local}(T_i'', U_i)$ is a criterion for instability of tree T_i'', and is defined as follows.

$$\begin{aligned}\mathrm{Instability\_Local}(T_i'', U_i) &= \mathrm{max}_{e \in T_i''}(\mathrm{instability}(e)) \\ \mathrm{instability}(e) &= \mathrm{distance}(v1(e), v2(e)) \\ &\quad \cdot (\mathrm{speed}(v1(e)) + \mathrm{speed}(v2(e)))\end{aligned}$$

where $v1(e)$ and $v2(e)$ denote endpoints of edge e, $\mathrm{distance}(u, v)$ denotes the estimated distance between nodes u and v, and $\mathrm{speed}(v)$ denotes moving speed of node

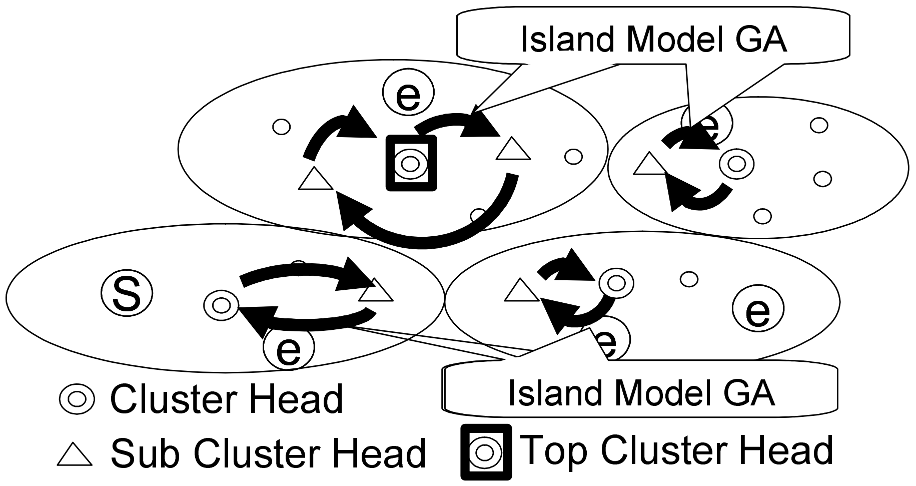

Figure 9. Island Model GA

v.

3.3.4 Detailed Description of The Proposed Method

The proposed method repeats the following three steps.

1. Divide all nodes in the network into clusters.

2. Compute global tree, which will be used as a backbone.

3. Compute local tree for each cluster.

The whole multicast tree is constructed as a concatenation of local trees and a global tree. Each local tree is constructed using IGA [37] in corresponding cluster (see Fig. 9). In each cluster, a cluster head and sub cluster heads correspond to islands of IGA for computing the local tree of the cluster. Global tree is constructed using IGA in the cluster to which the top cluster head belongs. In the cluster, a top cluster head and sub cluster heads correspond to islands of IGA for computing the global tree.

IGA is a kind of parallel GA. It is a technique to find a solution using several GAs with independent groups of candidate solutions where they cooperate by exchanging a few candidate solutions between adjacent groups (arrows of Fig. 9). The IGA has some

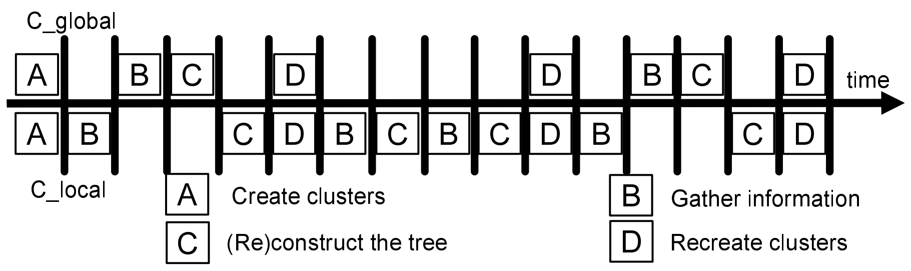

Figure 10. Computation Procedure of Multicast Tree

advantages such that (i) it does not require frequent communications between computation nodes, and that (ii) it is effective to maintain diversity of candidate solutions and thus there is relatively small chance to fall into local optima during computation.

According to node movements, the multicast tree has to be reconstructed. The proposed method reconstructs the tree by repeating the three steps periodically.

Furthermore, step 2 and step 3 are divided into an information gathering phase and a tree computation phase, respectively. According to the above discussion, the proposed method is carries out by six phases, as listed below. (A) *cluster division*, (B-global) *information gathering between clusters*, (C-global) *construction of a global tree*, (B-local) *information gathering in a cluster*, (C-local) *construction of a local tree*, and (D) *reconstruction of all clusters and trees*. The whole procedure is shown in Fig. 10. If the change of inter-cluster topology is smaller than change of intra-cluster topology, phase (B-local) and phase (C-local) are repeated more frequently than other phases.

Hereafter, we explain the details of each phase.

(A) Cluster Division

In this phase, nodes are divided into clusters. Also, a cluster head and sub cluster heads are selected for each cluster. The IGA is executed on the cluster head and sub cluster heads. A cluster head of the cluster with a source node s is selected as the top cluster head. The top cluster head governs the information regarding to all clusters.

There are some algorithms for clustering and deciding cluster heads. [38] is an algorithm to re-construct clusters with relatively small load. MAX-MIN D-cluster algorithm [2] is an algorithm which constructs clusters within D radius by exchanging information $3D$ times between adjacent nodes.

In the proposed method, each node generates a random number, and the nodes generated highest k numbers are chosen as a cluster head and $k-1$ sub cluster heads. This can be realized by a flooding with $D/2$ of TTL. Here, k and D are integer numbers given in advance.

(B-local) Information Gathering in a Cluster

In this phase, topology information of each cluster is gathered to the cluster head. The following information is gathered.

- Path from each node in the cluster to the cluster head.
- IDs of user nodes requesting data stream in the cluster.
- State of each link in the cluster (unused bandwidth, delay, estimated distance between nodes)
- Moving speed of each node
- Paths between the cluster head and cluster heads of adjacent clusters.

First, all nodes in the cluster C_i and paths between the cluster head h_i and cluster heads of adjacent clusters are searched by flooding messages (shown as REQ Message M_a in Fig. 11). Each node which received a message sends a reply packet (REP Message M_b in Fig. 11) toward the cluster head h_i. The reply packet is delivered to h_i by traversing the backward path. The reply packet contains ID of a node requesting data stream and information regarding to states of links to its neighbor nodes.

If a node receives a REQ message originated from the cluster head h_j of an adjacent cluster C_j, it sends the path between h_i and h_j toward h_i and h_j (the path is called an *adjacent path*, hereafter).

(B-global) Gathering Information between Clusters

In this phase, information regarding to bandwidth and delay of the adjacent path is gathered to a cluster head (see Fig. 12). This information and the number of data delivery requests from user nodes are gathered to a top cluster head from all cluster heads.

Nodes on each adjacent path send information on delay and bandwidth of its links to the cluster head. The information of adjacent paths (including available bandwidth

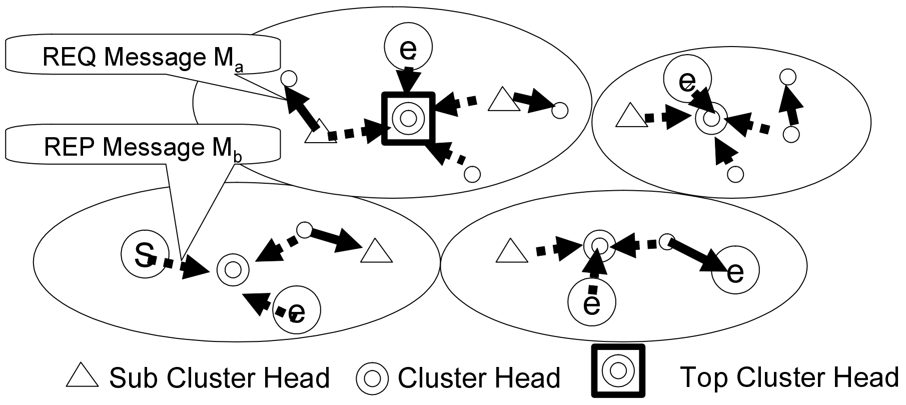

Figure 11. Gathering Information in a Cluster

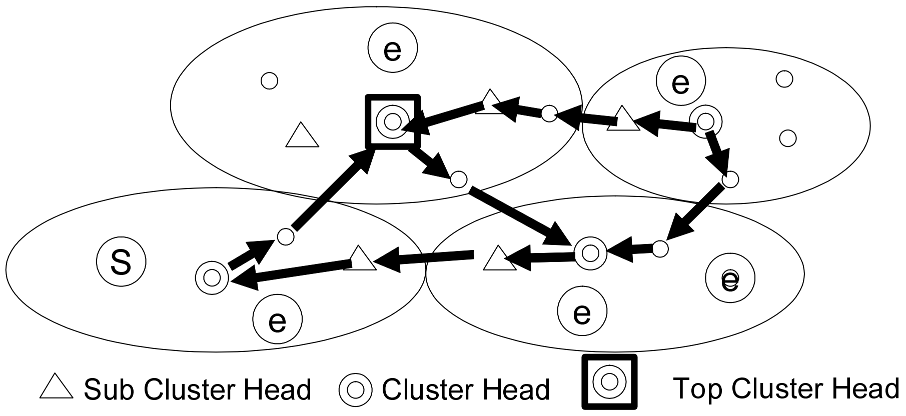

Figure 12. Gathering Information between Clusters

and delay) is sent from each cluster head to the top cluster head. In the proposed method, the information about bandwidth and delay on path between cluster heads are gathered in the top cluster head while this phase. Here, information of adjacent paths which do not satisfy the constraints regarding to required bandwidth and delay is not sent to the top cluster head. Therefore, all the gathered adjacent paths satisfy the constraint of bandwidth (for delay, each adjacent path satisfies the delay constraint Dr, but concatenation of paths may not satisfy it). Each cluster head can transmit a packet to the next hop nodes of the global tree and the first child nodes of the local tree by one broadcast. So, bandwidth competition between local tree and global tree does not occur, and thus the constraint regarding to bandwidth is not needed when constructing global tree.

(C-global) Computation of Global Tree

In this phase, the top cluster head and sub cluster heads of the same cluster compute a global tree (see Fig. 7) by solving the problem defined in Section 3.3.2 using the IGA. Then, the information on global tree is sent to each cluster head in the tree. When each cluster head receives this information, it maintains only information regarding to the paths to the next (downstream) cluster heads in the tree. The global tree is a multicast tree of cluster heads and its root is the top cluster head (cluster head of the cluster with s). The genetic operators used in GA are described in Section 3.3.5.

(C-local) Computation of local tree

In this phase, cluster head and sub-cluster heads of each cluster compute a local tree for the cluster by solving the problem defined in Section 3.3.3. The local tree is a multicast tree whose root is the cluster head h_i. IGA is used to compute the tree as shown in Fig. 9. The genetic operators used in GA are described in Section 3.3.5. The information on the computed local tree is sent to each node in the tree, and when each node receives this information, it maintains only information regarding to the next (downstream) nodes on the tree.

Delivery of Computed local tree:

We describe how the information of local tree is delivered to nodes in the cluster. First, a cluster head broadcasts a transmission packet. The node which receives the packet confirms the sender of the packet. Only if the sender node is the upstream node of the local tree, the packet is broadcasted further. By repeating this process, a packet can be transmitted to the nodes of lower stream in a tree.

(D) Cluster Reconstruction

In phase (D), in order to cope with topology change according to node movement, clusters are reconstructed. A new node or a disconnected node broadcasts a data delivery request to neighboring nodes. The node which received the request message forwards the message toward the cluster head.

The cluster head which received this message reconstructs the cluster using the same method as the phase (A).

3.3.5 Genetic Operator

Here, we describe a method to code and decode a candidate solution, and genetic operators used in the proposed method. When solving tree optimization problems using GA, a general coding method which can represent any trees may be used. However, when we use a general coding scheme, solution space tends to be large, and thus longer computation time may be required until the candidate solutions converge. On the other hand, if we use a coding scheme with narrow solution space, it would be difficult to maintain diversity of the candidate solutions. In the proposed method, we devised an efficient coding scheme with appropriate size of solution space. In this coding scheme, a tree is encoded as a sequence of nodes. A tree is decoded from a sequence of nodes so that nodes near the cluster head or the top cluster head are added to the tree in prior to other nodes. Therefore, the number of hops in the tree tends to become small.

Genetic Operators for Global Tree

First, we give ID numbers to clusters, where ID 0 is given to a cluster with a source node. $|\mathscr{C}|$ denotes the number of clusters in the network. A candidate solution (chromosome) of a multicast tree is encoded as a sequence $< g_1, ..., g_i, ..., g_{|\mathscr{C}|-1} >$, where g_i represents the ID number of cluster adjacent to cluster C_i if such a cluster exists (g_i becomes -1 if no such cluster exists).

The decoding scheme is shown in Algo. 4. Here, $T' = (V', E')$ denotes a global tree.

---

1. Let the initial value of the vertex set V' of T' be an empty set. Add the cluster (to which s belongs) to V'. Let the initial value of the edge set E' of T' be an empty set.
2. Assign 1 to j. Assign $false$ to flag.
3. If g_j is -1, go to step 7. Otherwise go to step 4.
4. If cluster g_j is not in V', go to step 7.
5. If cluster j is in V', go to step 7.
6. Add j to V'. Add the edge connecting cluster j and cluster g_j to E'. Assign $true$ to flag.

7. Add 1 to j. If j is $|\mathscr{C}| - 1$ or less, go to step 3.

8. if flag is $false$, the algorithm terminates. Otherwise go to step 2.

Algorithm 4. Pseudo code of Genetic Operators for Global Tree

As genetic operators for global tree, we use the uniform crossover and a mutation method which randomly replaces a gene (node) by another one in chromosome of a candidate solution.

Genetic Operators for Local Tree

For local tree construction, we use the same coding scheme, crossover method, and mutation method as the global tree construction. A candidate solution (chromosome) of cluster C's local tree is a sequence $< n_1, ..., n_i, ..., n_{|C|-1} >$, where $|C|$ represents the number of nodes in C. We give integer ID numbers to all nodes in C, where a cluster head has ID of 0. n_i denotes the ID of a node adjacent to node i. If there is no adjacent node to node i, n_i becomes -1.

The decoding scheme is shown in Algo. 5. Here, $T'' = (V'', E'')$ denotes a global tree.

---

1. Let the initial value of the vertex set V'' of T'' be an empty set. Add 0 (cluster head) to V''.

2. Assign 1 to j. Assign $false$ to flag.

3. If n_j is -1, then go to step 7. Otherwise go to step 4.

4. If n_j is in V'', then go to step 5. Otherwise go to step 7.

5. If node j is not in V'', then go to step 6. Otherwise go to step 7.

6. Add j to V''. Add the edge connecting node j with node n_j to E''. Assign $true$ to flag.

7. Add 1 to j. If j is $|C| - 1$ or less, then go to step 3. Otherwise go to step 8.

8. if flag is $false$, the algorithm terminates. Otherwise go to step 2.

Algorithm 5. Pseudo code of Genetic Operators for Local Tree

3.4 Evaluation Experiment

We conducted experiments to investigate the performance of the proposed method in following terms.

(1) computation time of a multicast tree

(2) communication cost

(3) efficiency of multicast tree

Here, each experiment is explained.

3.4.1 Computation Time of Multicast Tree

In order to investigate the scalability of the proposed method, we measured computation time of a tree, changing the number of nodes in MANET. We also measured the recomputation time to investigate whether GA can quickly recompute a new multicast tree by using solution candidates used for the last computation. We assumed that candidate solutions converge when the best candidate solution is not updated for 30-generations. In each trial, we measured computation time until convergence. Here, we suppose that all nodes belong to a local tree. Assuming the operation on a portable terminal, we used a note book PC as a mobile terminal for computation[4].

The specification of the PC and other experimental environment are as follows: CPU Intel(R) Pentium(R) M processor 1500MHz, Windows XP Pro, cygwin 1.5.18, and gcc version 3.4.4.

The experimental result is shown in Table 6. This result is the average value of 30 trials. According to Table 6, we see that the proposed method can compute a multicast tree with 30 nodes (for a cluster) in 0.02 seconds. Since the number of nodes of each cluster is less than 15 when we use 5 hops as maximum radius of clusters, this result is short enough for practical use. In addition, the computation time for global tree was similar to that of local tree as long as the number of nodes is same.

Moreover, we see that computation time could be shortened to 60% by using solution candidates used for the last computation. Therefore, we confirmed that in order to

[4]Here, we run several IGAs in one machine for simplicity of experiment. If we use several PCs which run IGAs in parallel, the performance will be improved.

Table 6. Computation time of local tree

	Time (sec.)		Fitness	
node	computation	recomputation	computation	recomputation
10	0.010400	0.006867	0.800000	0.800000
30	0.022300	0.009200	1.000000	0.993750
50	0.040467	0.018533	0.989409	0.942153
68	0.053100	0.032600	0.953586	0.986990
100	0.104100	0.087500	0.894136	0.957880
113	0.129533	0.077867	0.914843	0.908532
225	0.600333	0.326400	0.858166	0.845311
270	0.865200	0.446333	0.835762	0.902104
450	2.417233	1.374933	0.729587	0.843629
900	6.822000	4.999367	0.561283	0.627084
1000	9.097533	6.095467	0.583697	0.643176

reduce computation time, it was effective to reuse solution candidates used for the last computation.

Table 6 suggests that the computation time of a multicast tree is almost proportional to the square of the number of nodes. Since the computation time of local tree is almost the same as that of global tree, the total computation time for entire MANET is minimized when the number of nodes in each cluster and the number of clusters are $\sqrt{N}$, where N is the number of nodes in MANET. However, since global tree is constructed using aggregated information, it is thought that the size of the local tree is necessary to some extent if there are few total nodes.

3.4.2 Communication Cost

To investigate the communication cost of the proposed method, we measured per-node control traffic required for the proposed method by simulation.

In the experiment, we used GTNetS [27] as a network simulator. The simulation configuration is as follows: simulation space is 3000 $\times$ 3000 m^2; the number of nodes is 1000; MAC layer protocol is IEEE802.11 (maximum transmission speed is 2Mbps);

the radio range of all nodes is 160m; all nodes do not move; clustering algorithm is Max-Min D-Cluster [2]; and the maximum radius of cluster is changed among 3, 5 and 10 hops.

The experimental results are shown in Table 7 and Table 8. Here, each value is the sum of incoming and outgoing traffic per node (or per cluster head), and is calculated as an average of 10 trials. From these tables, we see that control traffic of clustering is the largest. For the traffic of information gathering in a cluster, the traffic of a cluster head is higher than that of a normal node, since all the information in a cluster is gathered to a cluster head. The traffic for clustering of a cluster head is lower than that of a normal node. The reason is that there are some cluster heads isolated from other nodes and they reduce average values. From Table 8, the control traffic becomes larger as cluster radius increases. However, the entire control traffic is only 4.4K byte even if cluster radius is 10 (average number of nodes in a cluster is 36). According to the above results, even when reconstructing a tree with 36 nodes once every 20 seconds, the required control traffic at a normal node and a cluster head are 1.8Kbps and 1.7Kbps, respectively. As for these values, this traffic is small enough to be used at mobile terminals. On the other hand, as shown in Table 8, along with the increase of the number of nodes in a cluster (or the number of clusters), the amount of computation increases.

Therefore, as for MANET with the number of N nodes, under the consideration of communication traffic and the amount of computation, required resource for a mobile terminal can be minimized, by dividing MANET so that the number of nodes contained in each cluster is closer to $\sqrt{N}$, and the number of clusters contained in the whole is closer to $\sqrt{N}$ clusters in the whole.

From the experimental results of computation cost and communication cost, the proposed method can compute a multicast tree for 1000 nodes in 0.004 seconds and the required traffic is about 1.8Kbps per node, by dividing them into about 30 clusters with about 30 nodes. We think that this result is practical enough. In this case, the proposed method can hold down the communications traffic of each cluster head in order to compute the local tree to about 4.2 K bytes. This result shows that if reconstruction of a multicast tree is performed every 20-30 seconds, the proposed method can be realized in an actual environment by assigning the terminal which has a performance comparable to note PC and can use the radio communications complying with 802.11

Table 7. Required control traffic per node (Bytes)

Cluster radius (the num. of nodes in a cluster)	3(7.7)	5 (15.2)	10(36.0)
Comm. traffic in a clustering	910.2	1603.6	3640.2
Comm. traffic of info. gathering between clusters	341.8	291.6	259.8
Comm. traffic of info. gathering between clusters	161.8	187.3	291.0
Comm. traffic of delivery of global tree	2.9	1.72	0.59
Comm. traffic of delivery of local tree	30.2	70.25	185.8
Sum of comm. traffic	1446.9	2154.5	4377.4

to a cluster head.

3.4.3 Efficiency of Multicast Tree

In order to evaluate the stability of the multicast tree constructed by the proposed method, we measured the transition of the packet arrival rate at user nodes in the tree as time passes. We defined the packet arrival rate to be the ratio of the number of received packets over the number of packets which were transmitted from the source node. The average packet arrival rate decreases as time passes since the multicast tree was computed based on the past topology and some links in the tree are broken as the network topology changes.

In this experiment, we used the same configuration as the previous section except bellow:

(1) we used two kinds of moving speed of each node: 0 Km/hour and 4 Km/hour

(2) the random waypoint (RWP) was used as mobility model

(3) data stream of 64Kbps is transmitted through multicast tree[5]

[5]It corresponds to 320Kbps when we use IEEE802.11b

Table 8. communications traffic per one cluster head (Bytes)

Cluster radius (the num. of nodes in a cluster)	3(7.7)	5 (15.2)	10(36.0)
Comm. traffic in a clustering	804.8	1416.2	2787.9
Comm. traffic of info. gathering between clusters	302.8	236.7	185.6
Comm. traffic of info. gathering between clusters	410.9	624.9	1208.1
Sum of comm. traffic	1518.5	2277.8	4181.6

(4) each node can change the radius of radio range for power saving when transmitting a packet

In order to show the applicability of the proposed method, we measured the packet arrival rate for the following three different objective functions:

(1) objective functions in Sect. 2 with larger γ and larger ϵ values to regard tree stability more important (hereafter referred to as *stable*)

(2) objective functions with smaller γ and smaller ϵ values (hereafter referred to as *non-stable*)

(3) objective functions with new terms for power consumption [6] (referred to as *power-saving*)

The following setting was used as the power consumption minimization technique. Since most of mobile terminals drive with battery in the MANET, it is an important topic to save power consumption. In the experiment, the power consumption of packet transmission in the terminal is proportionate to the square of transmission distance of the radio wave, and the term which minimizes total of this power consumption was added to the evaluation function. For the objective function (3), we allow each node to adjust the strength of the transmission power so that radius of radio range will be 1.1

[6]In these terms, power consumption of a packet transmission is defined to be proportional to the square of the distance between nodes.

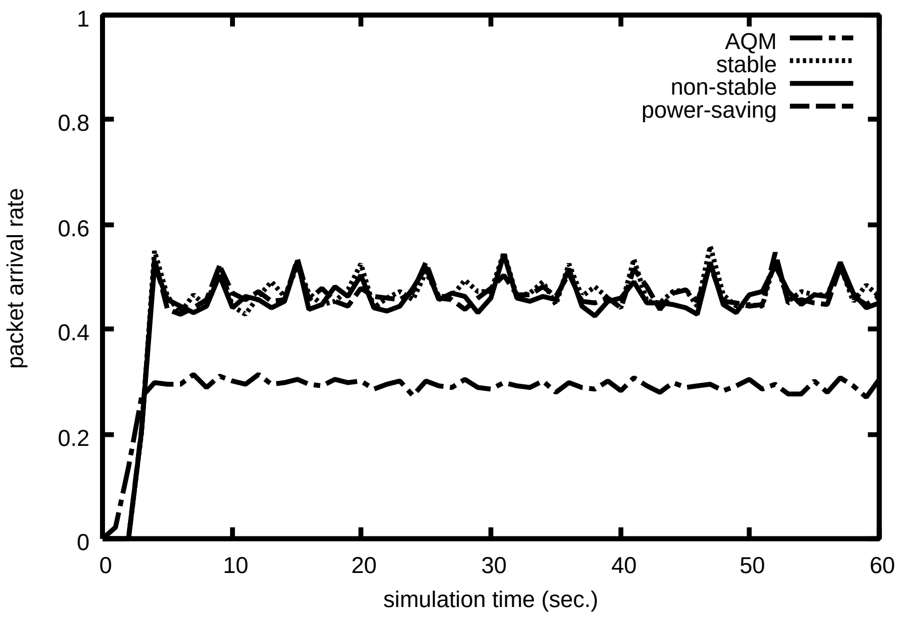

Figure 13. Transition of The Packet Arrival Rate over Time:0km/h

times larger than the minimum distance required reaching the destination to keep the tree stability [7]. This is because the stability of the path at the time of node movement was taken into consideration.

Since there does not exist the protocol which constructs semi-optimal multicast tree to the specified arbitrary evaluation metrics such as the proposed method, also, we compared our method with AQM [6]. AQM is an on-demand multicast routing method which satisfies bandwidth constraint by efficiently investigating the bandwidth usage of adjacent nodes. In order to evaluate the stability of the constructed multicast tree, the reconstruction function of the delivery tree was not used. In Fig. 13 and Fig. 14, we show experimental results when the maximum speed of nodes are 0km/h and 4km/h, respectively. Here, note that the initial tree was not re-constructed during simulation to see the stability of the computed tree.

The packet arrival rate of our experimental results is less than 50 % throughout simulation time, and it may look inefficient. This is because some of the nodes in

[7]We suppose that each node can guess the distance to other nodes from the strength of radio wave signal.

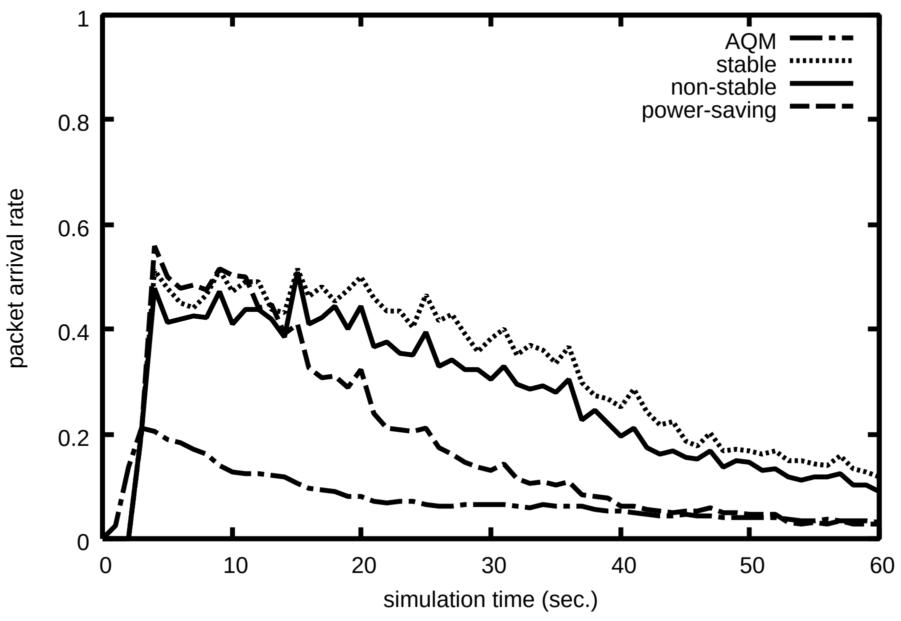

Figure 14. Transition of The Packet Arrival Rate over Time:4km/h

MANET are geographically separated off the radio ranges of the nodes included in the multicast tree, and packets cannot reach them at all. Fig. 13 shows that the proposed method with any objective function is superior to AQM in terms of packet arrival rate when nodes do not move. There is no big difference of performance among three objective functions of our method. On the other hand, as shown in Fig. 14, when a node moves, the packet arrival rate decreases as time passes. We see that the method *stable* constructed the most stable multicast tree.

Next, we compared the power consumption for data streaming through the multicast trees constructed by our methods (*power-saving* and *stable*) and AQM. The ratio of the power consumption among those methods was that *stable* : AQM : *power-saving* = 1 : 0.67 : 0.61. Furthermore, we compared the total power consumption in reconstruction interval of multicast tree (for 20 seconds). The transmission power consumption in the interval among those methods was that *stable* : AQM : *power-saving* = 10.095 Watt-second : 6.7639 Watt-second : 6.04317 Watt-second [8]. However, the proposed

[8]We suppose that transmitted power is 0.28183 Watt and the number of the transmitting nodes of AQM is 750.

method need the power consumption of CPU to recompute a tree by using solution candidates used for the last computation. When Note PC (CPU Intel(R) Pentium(R) M processor 1500MHz, Thermal Design Power 24.5 Watt) is used, computation time is required for 0.009200 seconds, and power consumption is 7.66360 Watt-second. When PDA (Intel XScale (R) PXA270 (624MHz), Thermal Design Power 0.747 Watt) is used, computation time is required for 0.0038 seconds [9], and power consumption is 0.56168 Watt-second. Added these, total power consumption in reconstruction interval of multicast tree (for 20 seconds) between AQM and *power-saving* was that AQM : *power-saving*(note PC): *power-saving*(PDA)= 6.7639 Watt-second : 13.70677 Watt-second : 6.60485 Watt-second. Consequently, the result of the power-saving method showed power consumption smaller than AQM and the stable method.

As mentioned above, we conformed that according to various objects (such as stability and lower power consumption, etc), the proposed method was able to construct the semi-optimal trees for each object, respectively.

As a demerit of the proposed method, high calculation cost and shortage of readiness is raised. Since, in the proposed method, the computation power of a cluster head is assumed about note PC, it is difficult to apply at present when MANET environment is constituted with terminals with lower computation ability, such as PDA and a cell phone unit. Moreover, multicast tree need to be re-compute even if a path is cut. It is considered that this can be improved using hybrid technique which combined the on-demand routing technique.

[9]We suppose that computation time is proportional to clock frequency.

3.5 Conclusion

In this chapter, in order to realize multimedia streaming on MANET, we proposed a new multicast routing method for MANET. The proposed method constructs the semi-optimal multicast tree satisfying QoS constraints for any given objective. The proposed method constructs tree by two or more nodes on MANET. The features of our method are that it can treat arbitrary objective such as minimization of total power consumption for constructing optimized multicast tree, and that it is scalable to the number of nodes by distributed computation of the multicast tree based on clustering and IGA. Through simulation, we showed that the proposed method can compute more superior multicast trees than an existing on-demand multicast routing method AQM in terms of tree stability and power consumption.

4. Conclusion and Future work

4.1 Summary of This Thesis

In this thesis, focusing on island model GA, the following two research topics on improvement of exploration efficiency of island model GA and application to routing problem in mobile ad hoc network have been studied.

In Chapter 2, we proposed self adaptive island model GA (SAIGA) which can adjust multiple parameter value. By reducing the number of iterations in our previous work called A-SAGA (proposed agent-oriented self-adaptive GA), Computation cost with SAIGA can be lowered than that with A-SAGA. SAIGA can efficiently find a solution with automatic parameter adjustment, when solving only one instance in a problem. We also propose asynchronous SAIGA (called A-SAIGA, hereafter) which eliminates waiting time for synchronization among islands so that search efficiency is improved by reducing idle processor time. In order to evaluate effectiveness of the proposed method, we applied the proposed method to various problems and measured search efficiencies. Through experiments, we showed that the proposed method can find appropriate parameter values for various problems. We also showed that performance of the proposed method outperforms simple GA with standard parameter values, and is close to simple GA with manually adjusted optimal parameter values. Also, the processing time of A-SAIGA is able to be reduced to 35-75 percent compared with that of SAIGA. This means that A-SAIGA has reduced waiting time for synchronize among islands.

In Chapter 3, in order to realize multimedia streaming on MANET, we proposed a new multicast routing method for MANET. The proposed method constructs the semi-optimal multicast tree satisfying QoS constraints for any given objective. The proposed method constructs tree by two or more nodes on MANET. The features of our method are that it can treat arbitrary objective such as minimization of total power consumption for constructing optimized multicast tree, and that it is scalable to the number of nodes by distributed computation of the multicast tree based on clustering and IGA. To do so, it divides MANET into multiple clusters, and makes them construct sub-trees covering their clusters in parallel and computes a tree which connects the sub-trees of all clusters. Consequently, one semi-optimal multicast tree is constructed with them. Moreover HQMGA abstracts the topology information spanning all clusters so that a node can compute the tree connecting all clusters within small computation power and communication amount. Through simulation, we showed that the proposed method can compute more superior multicast trees than an existing on-demand multicast routing method AQM in terms of tree stability and power consumption.

4.2 Future work

The proposed method in Chapter2 might be extended toward the following three directions.

1. **Automatically optimizing distribution of processor resource in each island**
   Our experiments showed that A-SAIGA has better performance than SAIGA. This implies that distributing processor performance equally to islands does not always give the best result. We are going to investigate influences of distributing performance non-equally to islands, and make a method to optimize distribution.

2. **Introducing age model adopted in aGA [12] instead of FIFO deletion**
   The model considers both evaluation value and age of each individual to make selection. This should improve results in dynamic environment.

3. **Introducing other generation model (ex. MGG [29]) in low level GA of the proposed method.**
   In this experiment, SGA was used for low level GA, but generation model such as MGG model is able to be introduced to low level GA.

The proposed method in Chapter3 might be extended toward the following two directions.

1. **The improvement of the multicast tree**
   In the proposed method, computation of the global tree and local tree are performed independently. We can obtain the better multicast tree in shorter time by sharing the computation results of the global tree and local trees.

2. **The correspondence to two or more streams**
   In the proposed method, the single stream is assumed. When delivering two or more streams at same time, it is thought that load concentrates on a cluster head.

In order to avoid this problem, a technique of changing a cluster head for every stream should be examined.

Acknowledgements

First, I wish to express my sincere gratitude to my supervisor Professor Minoru Ito, for his continuous support and valuable advice.

I am also very grateful to my Co-supervisor Professor Kenji Sugimoto for valuable comments to this thesis.

I also want to thank to my Co-supervisor Associate Professor Keiichi Yasumoto for his valuable advice and constant encouragement. This work could not be achieved without his support and guidance.

I wish to thank Associate Professor of Shiga University Naoki Shibata and assistant professor Yoshihiro Murata for helpful discussion and valuable advice.

I thank to all members of Foundations of Software Lab. for the days I shared with them.

Finally, I would like to thank all people from which I have received a kindness.

References

[1] Alba, E. and Troya, J. M.: An analysis of synchronous and asynchronous parallel distributed genetic algorithms with structured and panmictic islands, *Proc. of the 11th conjunction with Int'l Parallel Processing Symposium and Symposium on Parallel and Distributed Processing workshops (IPPS/SPDP'99 workshops)*, pp. 248–256 (1999).

[2] Amis, A. D., Prakash, R., Huynh, D. and Vuong, T.: Max-Min D-Cluster Formation in Wireless Ad Hoc Networks, *Proc. of the 19th IEEE Conf. on Computer Communications (INFOCOM 2000) (1)*, Vol. 1, pp. 32–41 (2000).

[3] Bäck, T.: The Interaction of Mutation Rate, Selection, and Self-Adaptation Within a Genetic Algorithm, *Proc. of the 2nd Conf. on Parallel Problem Solving from Nature (PPSN II)*, pp. 85–94 (1992).

[4] Bäck, T.: Self-adaptation in genetic algorithms, *Proc. of the 1st European Conf. on Artificial Life (ECAL'92)*, pp. 263–271 (1992).

[5] Berntsson, J. and Tang, M.: A convergence model for asynchronous parallel genetic algorithms, *Proc. of the IEEE Cong. on Evolutionary Computation (CEC 2003)*, pp. 2627–2634 (2003).

[6] Bür, K. and Ersoy, C.: Ad Hoc Quality of Service Multicast Routing, *J. of Computer Communications (JCC)*, Vol. 29, No. 1, pp. 136–148 (2005).

[7] Chen, S. and Nahrstedt, K.: A Distributed Quality-of-Service Routing in Ad-Hoc Networks, *IEEE J. on Selected Areas in Communications (JSAC)*, Vol. 17, No. 8, pp. 1–18 (1999).

[8] Coltun, R. and Fuller, V.: The OSPF NSSA Option, RFC 1587 (Proposed Standard) (1994). Obsoleted by RFC 3101.

[9] D. Jong, K. A.: *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, PhD Thesis, University of Michigan (1975).

[10] D. Jong, K. A. and Sarma, J.: Generation Gaps Revisited, *Proc. of Foundations of Genetic Algorithms 2 (FOGA 2)*, Morgan Kaufmann, pp. 19–28 (1993).

[11] Espinoza, F., Minsker, B. S. and Goldberg, D.: A Self-Adaptive Hybrid Genetic Algorithm, *Proc. of the Genetic and Evolutionary Computation Conf. (GECCO 2001)* (2001).

[12] Ghosh, A., Tstutsui, S. and Tanaka, H.: Function Optimization in Nonstationary Environment using Steady State Genetic Algorithms with Aging of Individuals, *Proc, of the 1998 IEEE Int'l Conf. on Evolutionary Computation (ICEC'98)*, pp. 666–671 (1998).

[13] Glicman, M. R. and Sycara, K.: Reasons for Premature Convergence of Self-Adapting Mutation Rates, *Proc. of the Cong. on Evolutionary Computation (CEC 2000)*, pp. 62–69 (2000).

[14] Goldberg, D. E., Deb, K. and Thierens, D.: Towards a Better Understanding of Mixing in Genetic Algorithms, Technical Report IlliGAL No.92009, University of Illinois (1992).

[15] Grefenstette, J. J.: Optimization of control parameters for genetic algorithms, *IEEE Trans. on Systems, Man, and Cybernetics (SMC)*, Vol. 16, No. 1, pp. 122–128 (1986).

[16] Harik, G. and Lobo, F.: A parameter-less genetic algorithm, Technical Report IlliGAL No.99009, University of Illinois at Urbana-Champaign (1999).

[17] Hinterding, R., Michalewicz, Z. and Peachey, T. C.: Self-adaptive genetic algorithm for numeric functions, *Proc. of the 4th Conf. on Parallel Problem Solving from Nature (PPSN IV)*, pp. 420–429 (1996).

[18] Kee, E., Airey, S. and Cyre, W.: An Adaptive Genetic Algorithm, *Proc. of the Genetic and Evolutionary Computation Conf. (GECCO 2001)*, Morgan Kaufmann, pp. 391–397 (2001).

[19] Krink, T. and Ursem, R. K.: Parameter Control Using the Agent Based Patchwork Model, *Proc. of the Cong. on Evolutionary Computation (CEC 2000)*, pp. 77–83 (2000).

[20] Lawrence, S.: *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement)*, Graduate School of Industrial Administration, Carnegie-Mellon University (1984).

[21] Layuan, L. and Chunlin, L.: QoS Multicast Routing in Networks with Uncertain Parameters, *Proc. of the 5th Asian-Pacific Web Conf. (APWeb 2003)*, pp. 430–441 (2003).

[22] Liao, W.-H., Tseng, Y.-C., Wang, S.-L. and Sheu, J.-P.: A Multi-path QoS Routing Protocol in a Wireless Mobile ad Hoc Network, *Proc. of the First Int'l Conf. on Networking-Part 2 (ICN '01)*, London, UK, Springer-Verlag, pp. 158–167 (2001).

[23] Lin, C. R. and Liu, J.-S.: QoS Routing in Ad Hoc Wireless Networks, *IEEE J. Selected Areas in Communications (JSAC)*, Vol. 17, No. 8, pp. 1426–1438 (1999).

[24] Murata, Y., Shibata, N., Yasumoto, K. and Ito, M.: Agent-Oriented Self Adaptive Genetic Algorithm, *Proc. IASTED Int'l Conf. Communications and Computer Networks (CCN 2002)*, pp. 348–353 (2002).

[25] Perkins, C., Belding-Royer, E. and Das, S.: Ad hoc On-Demand Distance Vector (AODV) Routing, RFC 3561 (Experimental) (2003).

[26] Reinelt, G.: TSPLIB, `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

[27] Riley, G. F.: The Georgia Tech Network Simulator, *Proc. of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research (MoMeTools '03)*, New York, NY, USA, ACM Press, pp. 5–12 (2003).

[28] Sait, S. M. and Youssef, H.: *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*, IEEE Computer Society Press, Los Alamitos, CA, USA (1999).

[29] Sato, H., Ono, I. and Kobayashi, S.: A New Generation Alternation Model of Genetic. Algorithms and its Assessment, *J. of Japanese Society for Artificial Intelligence (JSAI)*, Vol. 12, No. 5, pp. 734–744 (1997). (in Japanese).

[30] Sinha, P., Sivakumar, R. and Bharghavan, V.: CEDAR: A Core-Extraction Distributed Ad Hoc Routing Algorithm, *Proc. of IEEE Conf. on Computer Communications (INFOCOM'99)*, pp. 202–209 (1999).

[31] Sinha, P., Sivakumar, R. and Bharghavan, V.: MCEDAR: Multicast core extraction distributed ad-hoc routing, *Proc. of IEEE Wireless Communications and Networking Conf. (WCNC'99)*, Vol. 3, pp. 1313–1317 (1999).

[32] Takashima, E., Murata, Y., Shibata, N. and Ito, M.: Self Adaptive Island GA, *Proc. of the 2003 Cong. on Evolutionary Computation (CEC 2003)*, Vol. 2, pp. 1072–1079 (2003).

[33] Takashima, E., Murata, Y., Shibata, N. and Ito, M.: Techniques to Improve Exploration Efficiency of Parallel Self Adaptive Genetic Algorithms by Dispensing Synchronization, *Proc. of the 5th Int'l Conf. on Simulated Evolution And Learning (SEAL2004)* (2004). (CD-ROM).

[34] Takashima, E., Murata, Y., Shibata, N. and Ito, M.: Techniques to Improve Exploration Efficiency of Parallel Self Adaptive Genetic Algorithms by Dispensing Iteration and Synchronization, *J. of the Institute of Electronics Information and Communication Engineers (IEICE journal)*, Vol. J88-D-II, No. 9, pp. 1934–1943 (2005). (in Japanese).

[35] Takashima, E., Murata, Y., Shibata, N., Yasumoto, K. and Ito, M.: A Method for Distributed Computaion of Semi-Optimal Multicast Tree for Efficient Video Distribution in MANET, *J. of the Information Processing Society of Japan (IPSJ journal)*. to appear, (in Japanese).

[36] Takashima, E., Murata, Y., Shibata, N., Yasumoto, K. and Ito, M.: A Method for Distributed Computaion of Semi-Optimal Multicast Tree in MANET, *Proc. of the IEEE Wireless Communications and Networking Conf. (WCNC 2007)* (2007). accepted.

[37] Tanese, R.: Distributed Genetic Algorithms, *Proc. of the Third Int'l Conf. on Genetic Algorithms (ICGA'89)* (Schaffer, J. D.(ed.)), Morgan Kaufmann Publishers, pp. 434–439 (1989).

[38] Taniguchi, H., Inoue, M., Masuzawa, T. and Fujiwara, H.: Clustering algorithms in ad hoc networks, *Trans. of the Institute of Electronics, Information and Communication Engineers (IEICE)*, Vol. J84-D-I, No. 2, pp. 127–135 (2001).

[39] Tongchim, S. and Chongstitvatana, P.: Parallel Genetic Algorithm with Parameter Adaptation, *J. of Information Processing Letters*, Vol. 82, No. 1, pp. 47–54 (2002).

[40] Weinberg, R.: *Computer simulations of a living cell*, PhD Thesis, University of Michigan (1970).

[41] Zhu, C. and Corson, M. S.: QoS Routing for Mobile Ad Hoc Networks, *Proc. the 21st Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM '02)*, Vol. 2 (2002).

List of Major Publications

Journal Papers

1. Takashima, E., Murata, Y., Shibata, N. and Ito, M.: Techniques to Improve Exploration Efficiency of Parallel Self Adaptive Genetic Algorithms by Dispensing Iteration and Synchronization, *J. of the Institute of Electronics Information and Communication Engineers (IEICE journal)*, Vol.J88-D-II, No.9, pp.1934-1943, September, 2005, (in Japanese).

2. Takashima, E., Murata, Y., Shibata, N., Yasumoto, K. and Ito, M.: A Method for Distributed Computation of Semi-Optimal Multicast Tree for Efficient Video Distribution in MANET, *J. of the Information Processing Society of Japan (IPSJ journal)*, to appear, (in Japanese).

International Conference

1. Takashima, E., Murata, Y., Shibata, N. and Ito, M.: Self Adaptive Island GA, *Proc. of 2003 Cong. on Evolutionary Computation (CEC 2003)*, Vol.2, pp.1072-1079, December, 2003.

2. Takashima, E., Murata, Y., Shibata, N. and Ito, M.: Techniques to Improve Exploration Efficiency of Parallel Self Adaptive Genetic Algorithms by Dispensing Synchronization, *Proc. of 5th Int'l Conf. on Simulated Evolution And Learning (SEAL2004)*, (CD-ROM), October, 2004.

3. Takashima, E., Murata, Y., Shibata, N., Yasumoto, K. and Ito, M.: A Method for Distributed Computation of Semi-Optimal Multicast Tree in MANET, *Proc. of the IEEE Wireless Communications and Networking Conf. (WCNC 2007)*, accepted.

Domestic Technical Reports

1. Takashima, E., Murata, Y., Shibata, N. and Ito, M.: Improvement of Coding Method for the Parameters using Self Adaptive Island Genetic Algorithm, *LA symposium (Summer)*, pp.S6.1-S6.5, July, 2003, (in Japanese).

2. Takashima, E., Murata, Y., Shibata, N. and Ito, M.: Asynchronous Implementation of Self Adaptive Island Genetic Algorithm, *IPSJ SIG Technical Reports*, 2003-MPS-46, pp.21-24, September, 2003, (in Japanese).

3. Takashima, E., Murata, Y., Shibata, N., Yasumoto, K. and Ito, M.: A Method for QoS Multicast Routing using Island Model GA in MANET, *Multimedia, Distributed, Cooperative and Mobile (DICOMO 2005) Symposium*, IPSJ Symposium Series Vol. 2005, No. 6, pp. 413–416, 2005, (in Japanese).

4. Takashima, E., Murata, Y., Shibata, N., Yasumoto, K. and Ito, M.: Performance Evaluation of Hierarchical QoS Multicast Routing in MANET, *IPSJ SIG Technical Reports*, 2005-MBL-35, pp.61–68, November, 2005, (in Japanese).