Adaptive Application Component Mapping for Parallel Computation Offloading in Variable Environments

Wenhao Fan[1,2], Yuan'an Liu[1,2] and Bihua Tang[1,2]
[1] School of Electronic Engineering,
Beijing University of Posts and Telecommunications, Beijing 100876, China
[2] Beijing Key Laboratory of Work Safety Intelligent Monitoring,
Beijing University of Posts and Telecommunications, Beijing 100876, China
[e-mail: whfan@bupt.edu.cn]
*Corresponding author: Wenhao Fan

Received May 26, 2015; revised August 6, 2015; accepted September 2, 2015; published November 30, 2015

Abstract

Distinguished with traditional strategies which offload an application's computation to a single server, parallel computation offloading can promote the performance by simultaneously delivering the computation to multiple computing resources around the mobile terminal. However, due to the variability of communication and computation environments, static application component multi-partitioning algorithms are difficult to maintain the optimality of their solutions in time-varying scenarios, whereas, over-frequent algorithm executions triggered by changes of environments may bring excessive algorithm costs. To this end, an adaptive application component mapping algorithm for parallel computation offloading in variable environments is proposed in this paper, which aims at minimizing computation costs and inter-resource communication costs. It can provide the terminal a suitable solution for the current environment with a low incremental algorithm cost. We represent the application component multi-partitioning problem as a graph mapping model, then convert it into a pathfinding problem. A genetic algorithm enhanced by an elite-based immigrants mechanism is designed to obtain the solution adaptively, which can dynamically adjust the precision of the solution and boost the searching speed as transmission and processing speeds change. Simulation results demonstrate that our algorithm can promote the performance efficiently, and it is superior to the traditional approaches under variable environments to a large extent.

Keywords: parallel computation offloading, application multi-partitioning, graph mapping, combinatorial optimization, genetic algorithm

This work is support in part by National Natural Science Foundation of China (Grant No. 61170275 and 61502050), Civil Aerospace Science and Technology Project, YangFan Innovative & Entrepreneurial Research Team Project of Guangdong Province, Fundamental Research Funds for the Central Universities.

http://dx.doi.org/10.3837/tiis.2015.11.005 **ISSN : 1976-7277**

1. Introduction and Related Works

Computation offloading migrates the computation of an application from resource-constrained mobile terminals to external relative resourceful computation resources [1]. It can effectively enhance the capacities of mobile terminals to support diverse computation-occupying and energy-consuming mobile applications, whereas, whose requirements can not sufficiently satisfied by the embedded systems of mobile terminals with limited computation and energy capacities. The contradition between mobile terminal and mobile application becames severe especially in current and future periods that the scale of mobile internet industry increases explosively.

Traditional works for computation offloading mainly focus on the computation offloading strategies that offload the computation of an application to a single remote server [2][3][4]. However, the performance can be further promoted by simultaneously offloading the computation to multiple computation resources outside of the terminal, which is called parallel computation offloading. In this way, the degree of parallelism in the application can be utilized to a great extent, so that the computation and energy efficiency of the application can be then improved. In this area, most of the existing research basically considers the computation resources as multiple remote severs [5][6][7]. In the scenarios of pervasive computing, which becomes more and more popularized along with the development of IoT technologies, the generalized computation devices surround the terminal, such as laptops, PCs, tablets, wireless routers, air conditioners, printers, TVs, base stations, etc., can be taken as the computation resources for parallel computation offloading [8][9]. These computation devices are connected with the terminal via multiple heterogeneous network access technologies, such as WiFi, Bluetooth, Zigbee, 3G/LTE, etc.

Application component multi-partitioning algorithm is the core in parallel computation offloading. The application is abstracted as multiple components according to its structure. Based on the algorithm, these components are partitioned properly into multiple clusters, and each cluster is offloaded to its corresponding computation device.

In above scenarios, the computation and communication environments are variable. On the one hand, the computation capabilities of computation devices are diverse. The devices are impacted by the scale of computation that they are coping with currently. On the other hand, the qualities of the communication connections between the terminal and the computation devices are diverse, and they are influenced by the changes of wireless environments. Thus, static application component multi-partitioning algorithms are difficult to keep the optimality of their solutions in time-varying envirionments, whereas, over-frequent algorithm executions triggered by the changes of environments may bring excessive algorithm costs. Thus, a good application partitioning algorithm for these scenarios should provide the solution adaptively base on current environment, in order to prolong the effectiveness of the solution and maintain the level of the algorithm cost.

In regard to the existing research in this area, [10] proposes a computation offloading middleware for Android platform, where the algorithm takes transmission cost, memory cost and CPU cost as parameters, and models a 0-1 linear programming optimization problem. When the environmental parameters change, the solving process of the optimization problem is triggered to obtain the optimal solution for the new environment. However, the optimization for the algorithm cost, which is generated frequently and may consume a lot, is not focused.

[11] decides whether a function in the application should be offloaded based on a time threshold, which are computed according to the current environmental parameters. Still, the time threshold for each function is computed at every time when the environment changes, thus it leads to a high algorithm cost. [5] designs an adaptive $k + 1$ application partitioning algorithm. It considers memory cost, CPU utility and bandwidth. Based on graph partitioning theory, it partitions an application into one cluster running locally and k clusters be offloaded to multiple remote severs. The adaptivity of the algorithm for environment changes is not mentioned. [12] proposes an adaptive computation offloading engine, which employs a fuzzy logic model. The model evaluates the memory consumption of the application. Base on the model, it decides if the application should be offloaded. Although, the algorithm only considers the memory consumption, and its execution is triggered by the change of the memory, so over-frequent executions may appear when the variation of memory usage increases. [13] designs two application partitioning algorithms for small-scale and large-scale applications. The partitioning result is based on the variation of bandwidth. If the value of the bandwidth in current environment falls into the interval of the bandwidth threshold, then the algorithm re-execution will not be triggered. However, only the adaptivity of bandwidth is considered, and the adaptivity inside of the algorithm is still not investigated.

In this paper, we propose an adaptive application component mapping algorithm for parallel computation offloading in variable environments, which aims at minimizing computation costs and inter-resource communication costs and can provide the terminal a suitable solution for the period of the current environment with a low incremental algorithm cost. The algorithm abstracts the application component multi-partitioning problem as a graph mapping model, which consists of an application component graph and a computation device graph. Thus, the problem is converted into a pathfinding problem that finds out a proper path from the starting node to the end node in the search network. A genetic algorithm enhanced by an elite-based immigrants mechanism is designed to obtain the solution adaptively, which can dynamically adjust the precision of the solution and boost the searching speed as communication and computation parameters vary. The transmission speeds of network connections and the processing speeds of computation devices are chosen as variable parameters to represent the varying communication and computation environments. Simulation results validate the high adaptivity of our algorithm, demonstrating that the algorithm reduces the computation costs and inter-resource communication costs significantly, and it only takes a low incremental algorithm cost compared with traditional approaches. Our algorithm can be applied to the computation offloading frameworks and middlewares such as [2][10][14], etc.

The major contributions of our paper are as follows: (a) we abstract the the application component multi-partitioning problem as a graph mapping model with the transmission speeds of network connections and the processing speeds of computation devices considered as variable parameters, and convert the problem into a pathfinding problem; (b) we design a genetic algorithm enhanced by an elite-based immigrants mechanism to obtain the solution of the problem adaptively through dynamically adjusting the precision of the solution and boosting the searching speed, which can provide the suitable solution only with a low incremental algorithm cost.

The rest of this paper is organized as follows: Section II presents the graph mapping model for the application component multi-partitioning problem, where the application component graph and the computation device graph are defined. The genetic algorithm enhanced by elite-based immigrants is described in Section III, including all steps of the algorithm for

solving the pathfinding problem. Section IV shows the simulation results and the evaluations of our algorithm. Our work is concluded in Section V.

2. Graph Mapping Model for Application Component Multi-partitioning

An application running in a mobile terminal can be expressed as an undirected weighted graph [15] according to its program structure. The graph is called application component graph (ACG), which consists of vertices and edges. The vertices denote the components of the application with a certain granularity, such as classes, objects, modules, interfaces, functions or threads. Additionally, the vertices are from two categories: offloadable vertices or unoffloadable vertices. The formers are the ones that can be executed either in the terminal or in any one of the outside computation devices, whereas, the latter are the ones that can only run in the terminal, such as the components which operate the terminal's I/O hardware or are in charge of user interfaces, etc. An edge connecting two vertices in the ACG represents the communication between the two components that the two vertices correspond to. The weight of a vertex is defined as the amount of the computation that the corresponding component generates, and the weight of an edge is defined as the amount of data that needs to be transmitted between the two corresponding components that it connects with, if the two components are allocated to different computation devices in parallel computation offloading.

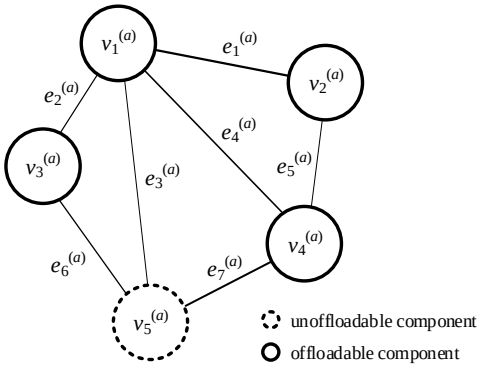
We use $G^{(a)} = (\mathbf{V}^{(a)}, \mathbf{V}_o^{(a)}, \mathbf{V}_u^{(a)}, \mathbf{E}^{(a)}, \boldsymbol{\delta}^{(a)}, \boldsymbol{\theta}^{(a)})$ to express the ACG of an application with m vertices and n edges. $\mathbf{V}^{(a)} = \{v_1^{(a)}, \dots, v_m^{(a)}\}$ is the vertex set of $G^{(a)}$. $\mathbf{V}_o^{(a)} \subseteq \mathbf{V}^{(a)}$ is the subset including all offloadable components, and $\mathbf{V}_u^{(a)} \subseteq \mathbf{V}^{(a)}$ is the subset including all unoffloadable ones. $\mathbf{E}^{(a)} = \{e_1^{(a)}, \dots, e_m^{(a)}\}$ is the edge set of $G^{(a)}$. The weight sets of $\mathbf{V}^{(a)}$ and $\mathbf{E}^{(a)}$ are denoted by $\boldsymbol{\delta}^{(a)} = \{\delta_1^{(a)}, \dots, \delta_m^{(a)}\}$ and $\boldsymbol{\theta}^{(a)} = \{\theta_1^{(a)}, \dots, \theta_n^{(a)}\}$, which contain the amount of computation of each computation device and the amount of data transmitted between two components if allocated differently, respectively.

In order to structurally express the relationships among vertices and edges in an ACG, an upper triangular matrix $H^{(a)}$ with m rows and m columns is employed to represent the existences and weights of edges between vertices. h_{kl} represent the edge between $v_k^{(a)}$ and $v_l^{(a)}$. $h_{kl} = 0$ if $k = l$ or there is no edge between $v_k^{(a)}$ and $v_l^{(a)}$, whereas, $h_{kl} \neq 0$ if $k \neq l$ and there is an edge between $v_k^{(a)}$ and $v_l^{(a)}$. Here, the value of h_{kl} is equal to the weight of the edge between $v_k^{(a)}$ and $v_l^{(a)}$. As an instance, an ACG is shown in **0**, which consists of 5 vertices and 7 edges, and its $H^{(a)}$ is

$$H^{(a)} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{1}$$

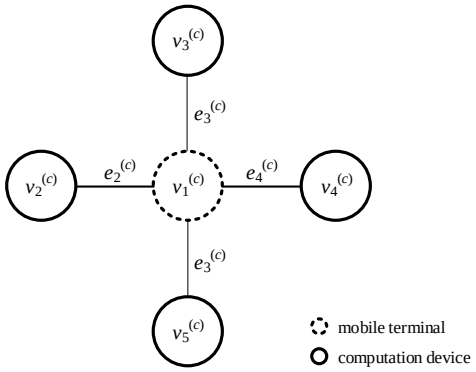
In the scenarios of the mobile terminal-centric ambient intelligence [16] developed by IoT technologies, the topology of the mobile terminal and its surrounding computation devices is actually a star network, where the computation devices are connected with the terminal via heterogenous networks, and the terminal is the center of the network. In the same way, the star network can be also expressed as an undirected weighted graph, called computation device graph (CDG), where the vertices denote the terminal and the computation devices, and the

edges denote the network connections between the terminal and the computation devices. The weight of a vertex represents the processing speed of the computation device that the vertex corresponds to. Here, we use MIPS (Million Instructions Per Second) to quantify the processing speed. The weight of an edge represents the data transmission speed of the network connection between the terminal and the corresponding computation device. Here, we use bandwidth (MB/s, Million Bytes per second) to quantify the transmission speed.



Application Component Graph

Fig. 1. An ACG with 5 vertices and 7 edges



Computation Device Graph

Fig. 2. A CDG with 4 computation devices around the terminal

$G^{(c)} = (\mathbf{V}^{(c)}, \mathbf{E}^{(c)}, \boldsymbol{\delta}^{(c)}, \boldsymbol{\theta}^{(c)})$ is adopted to express a CDG with p vertices and $p-1$ edges. We express the vertex set of $G^{(c)}$ as $\mathbf{V}^{(c)} = \{v_1^{(c)}, \ldots, v_p^{(c)}\}$, where the mobile terminal is denoted by $v_1^{(c)}$ fixedly. The edge set of $G^{(c)}$ is expressed as $\mathbf{E}^{(c)} = \{e_2^{(c)}, \ldots, e_p^{(c)}\}$. The weight sets of $\mathbf{V}^{(c)}$ and $\mathbf{E}^{(c)}$ are defined as $\boldsymbol{\delta}^{(c)} = \{\delta_1^{(c)}, \ldots, \delta_p^{(c)}\}$ and $\boldsymbol{\theta}^{(c)} = \{\theta_2^{(c)}, \ldots, \theta_p^{(c)}\}$, respectively, note that, the index of $\mathbf{E}^{(c)}$ and $\boldsymbol{\theta}^{(c)}$ starts from 2 in order to maintain the correspondence between the indexes of $\mathbf{V}^{(c)}$ and $\mathbf{E}^{(c)}$. As an instance, the topology of an CDG with 4 computation devices surrounding the terminal is illustrated in **0**.

The cost of a computation device is employed to measure the effect for offloading the allocated components to the device, which is combined linearly by the computation time consumed for the device processing the components, and the transmission time used for transmitting the components to the device. For a certain computation device $v_j^{(c)} \neq v_1^{(c)}$, its cost C_j is formulated by

$$C_j = C_j^{(x)} + C_j^{(y)} = \frac{\sum_{\delta_i^{(a)} \in \boldsymbol{\pi}_j} \delta_i^{(a)}}{\delta_j^{(c)}} + \frac{\sum_{\theta_i^{(a)} \in \boldsymbol{\phi}_j} \theta_i^{(a)}}{\theta_j^{(c)}} \tag{2}$$

where $C_j^{(x)}$ is the computation time of $v_j^{(c)}$, and $C_j^{(y)}$ is the transmission time of $v_j^{(c)}$. $\boldsymbol{\pi}_j$ and $\boldsymbol{\phi}_j$ are defined as the subsets of $\boldsymbol{\delta}^{(a)}$ and $\boldsymbol{\theta}^{(a)}$, respectively. $\boldsymbol{\pi}_j$ includes the weights of the vertices corresponding to all components that are offloaded to $v_j^{(c)}$. Similarly, $\boldsymbol{\phi}_j$ contains the weights of the edges corresponding to all the data that needs to be transmitted between $v_j^{(c)}$ and the terminal, note that, the data transmitted between two components are omitted if the both of them are offloaded to the same $v_j^{(c)}$, since the cost of inter-component

communications inside a computation device is very tiny and can be neglected. $C_j^{(x)}$ is obtained by summing all weights in $\boldsymbol{\pi}_j$ and then dividing the sum by $\delta_j^{(c)}$, because the sum of all weights in $\boldsymbol{\pi}_j$ is the total amount of computation hosted by $v_j^{(c)}$ in parallel computation offloading, and $\delta_j^{(c)}$ is the processing speed of $v_j^{(c)}$. $C_j^{(y)}$ is obtained by summing all weights in $\boldsymbol{\phi}_j$ and then dividing the sum by $\theta_j^{(c)}$, because the sum of all weights in $\boldsymbol{\phi}_j$ is the total amount of data needing to be transmitted from or to $v_j^{(c)}$ in parallel computation offloading, and $\delta_j^{(c)}$ is the data transmission speed of the network that $v_j^{(c)}$ corresponds to. If $v_j^{(c)} = v_1^{(c)}$, it means the computation device is actually the mobile terminal. In this case, the components that are allocated to the terminal run locally, and they need no data transmission in parallel computation offloading, so $C_1^{(y)} = 0$. Therefore, the cost C_1 of $v_1^{(c)}$ can be given by

$$C_1 = C_1^{(x)} = \frac{\Sigma_{\delta_i^{(a)} \in \boldsymbol{\pi}_1} \delta_i^{(a)}}{\delta_1^{(c)}} \tag{3}$$

After parallel computation offloading, the components of the application are offloaded to different computation devices or remain in the terminal. In the star network, the components hosted by each device are executed in parallel, thus the total cost is actually the maximum time consumed among the computation devices and the terminal to complete the whole computation of the application, and it can be formulated by

$$C = \max_{1 \le j \le p} C_j \tag{4}$$

It can be seen that the factors that impact C are $\boldsymbol{\pi}_j$ and $\boldsymbol{\phi}_j$, which form an application component multi-partitioning result that decides whether a component should be offloaded, and designates which component should be offloaded to which computation device. In this paper, we convert the application component multi-partitioning problem into a graph mapping problem. Thus, the multi-partitioning result is an mapping result that maps the vertices of the ACG $G^{(a)}$ to the vertices of the CDG $G^{(c)}$. The graph mapping can be defined as $W: \mathbf{V}^{(a)} \rightarrow \mathbf{V}^{(c)}$, which must obey the following rules: (a) all vertices in $\mathbf{V}_u^{(a)}$ are mapped to $v_1^{(c)}$ fixedly because they belong to unoffloadable components that can only run in the terminal; (b) A vertex in $\mathbf{V}_o^{(a)}$ can be only mapped to a unique vertex in $\mathbf{V}^{(c)}$ since duplications of components are forbidden, which may disturb the synchronization of the application's execution if the same component runs at different locations in parallel.

A matrix Z is employed to express the correspondences between vertices from $\mathbf{V}^{(a)}$ and vertices from $\mathbf{V}^{(c)}$ in a mapping result. Z is with m (the number of vertices in $\mathbf{V}^{(a)}$) rows and p (the number of vertices in $\mathbf{V}^{(c)}$) columns, where the value of an element z_{ij} at the ith row and jth column is defined as

$$z_{ij} = \begin{cases} 1, & if\ v_i^{(a)}\ is\ mapped\ to\ v_j^{(c)} \\ 0, & otherwise \end{cases} \tag{5}$$

Therefore, with Formula (2), (3) and (5) substituted, the cost of a certain computation device $v_j^{(c)}$or the terminal $v_1^{(c)}$can be rewritten as

$$C_j = \begin{cases} C_j^{(x)} + C_j^{(y)} \\ C_j^{(s)}, \end{cases} = \begin{cases} \frac{\sum_{i=1}^{m}\left(\delta_i^{(a)} z_{ij}\right)}{\delta_j^{(c)}} + \frac{\sum_{k=1}^{m}\sum_{l=1}^{m}\left(h_{kl}\left|z_{kj}-z_{lj}\right|\right)}{y_i^{(r)}}, & j \neq 1 \\ \frac{\sum_{i=1}^{m}\left(\delta_i^{(a)} z_{ij}\right)}{\delta_j^{(c)}}, & j = 1 \end{cases} \tag{6}$$

It can be observed that $\boldsymbol{\pi}_j, \ldots, \boldsymbol{\pi}_p$ and $\boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_p$ in Formula (2) and (3) are replaced by z_{ij} of Z in Formula (6).

The $\left|z_{kj} - z_{lj}\right|$ in Formula (6) can be explained as follows: (a) if $z_{kj} \neq z_{lj}$, namely, $z_{kj} = 0$, $z_{lj} = 1$ or $z_{kj} = 1$, $z_{lj} = 0$, which means only one of the two components is mapped to $v_j^{(c)}$, then the data transmission time between them is considered; (b) if $z_{kj} = z_{lj} = 0$, which means none of the two components is mapped to $v_j^{(c)}$, so the data transmission time between them is not considered since it is irrelevant with $v_j^{(c)}$; (c) if $z_{kj} = z_{lj} = 1$, which means both of the two components are offloaded to $v_j^{(c)}$, so the data transmission time can be omitted.

The objective of the parallel computation offloading is to find the optimal mapping result $Z^{(*)}$ which minimizes the total cost C, which can be formulated by

$$\min_Z C = \min_Z\left(\max_{1\leq j\leq p} C_j\right) \tag{7}$$

Formula (7) belongs to an combinatorial optimization problem [17], and is NP-hard according to Formula (6). It is very complicated to solve the problem directly. Enormous costs of searching and traversing need to be paid to find out the optimal solution from a large number of feasible solutions. Thus, high-efficiency algorithms are required to handle the complexity of the problem.

Besides, when the computation and communication environments are time-varying, a vital issue that is hard to avoid is the effectiveness of the solutions provided by the algorithm. A mapping result is generated based on the environment at the time when the algorithm launches, so it may not keep optimal as the environment changes, and the solution from a last algorithm execution will possibly expire very soon in a time-varying environment, at the moment, the algorithm needs to be re-executed to obtain the solution for current environment. However, if the frequency of the algorithm execution is too high, the algorithm cost will become a major burden that, on the contrary, counteracts the performance promotion brought by the parallel computation offloading; if the frequency is too low, the mapping result will deviate the optimality for current environment, and the performance of parallel computation offloading will deteriorate correspondingly. Therefore, there is a contradiction between the effectiveness of solution and algorithm cost in time-varying environments. A high-efficiency algorithm should balance the above two factors according to the environment characteristics adaptively, and provide suitable mapping results with low algorithm costs.

3. Design of the Genetic Algorithm Enhanced by Elite-based Immigrants

A genetic algorithm enhanced by elite-based immigrants is designed specifically to handle the optimization problem described in Formula (7). Firstly, we transform the optimization problem into a pathfinding problem. A search network is constructed, which contains all feasible paths. Then, we propose the chromosome representation, and describe each step used

in the algorithm. Finally, we present the adaptivity mechanism employed by the algorithm, which can dynamically adjust the precision of the solution and boost the searching speed as the processing speeds of computation devices and the transmission speeds of network connections change.

3.1 Pathfinding Problem

The solving process of the optimization problem is actually a searching process which finds out the optimal solution from all feasible solutions. Here, the optimization problem is transformed into a pathfinding problem, that is, a certain feasible solution is described by a corresponding path. The components in $\mathbf{V}_u^{(a)}$ are unoffloadable and are mapped to the terminal fixedly, so only the components in $\mathbf{V}_o^{(a)}$, which are offloadable, need to be considered in the pathfinding problem. The number of components in $\mathbf{V}_o^{(a)}$ is defined as m'. The search space of the pathfinding problem can be denoted by a search network, which is a layered graph consisting of multiple nodes according to the number of vertices in $\mathbf{V}_o^{(a)}$ and the number of vertices in $\mathbf{V}^{(c)}$. These nodes are organized into m' levels, and in each level, there are p nodes locating from left to right. A node, which is at the ith $(1 < i < m')$ level and the jth $(1 < j < p)$ location in the search network, represents a mapping from the ith component in $\mathbf{V}_o^{(a)}$ to $v_j^{(c)}$. A certain node at an upper level is associated with every node at the supper level. Thus, a solution consists of the mappings for all vertices in $\mathbf{V}_o^{(a)}$, so it can be described as a path which connects with the node at each level ordered from the 1st level to the m'th level in the search network.

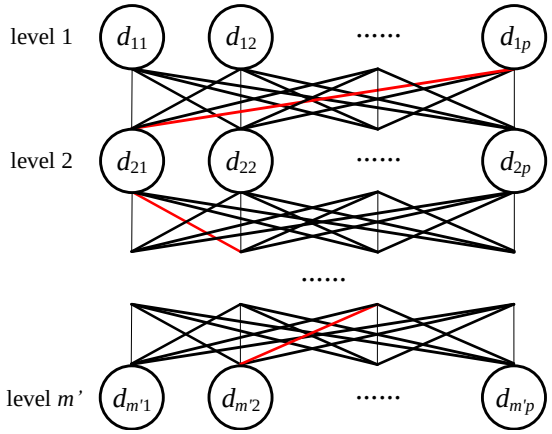


Fig. 3. The search network and a solution path

A path can be formulated by a sequence S with length m', which consists of the nodes that the path passes by in the search network. For example, a search network and a path in it are shown in **0**, where the path $S =< d_{1p}, d_{21}, \ldots, d_{m'2} >$ is drawn by the red lines.

3.2 Genetic Algorithm Enhanced by Elite-based Immigrants

Our genetic algorithm enhanced by elite-based immigrants is developed based on a standard genetic algorithm [18]. It involves several key steps: chromosome representation, population initialization, selection, crossover, mutation, fitness evaluation and an adaptivity mechanism

via elite-based immigrants. The solution of the optimization problem can be obtained by executing the algorithm's workflow which is composed of the above steps.

As a heuristic search inspired from the process of natural evolution, our genetic algorithm enhanced by elite-based immigrants manages the evolution process of a population. It iteratively looks for the solution of the problem, updates the population and makes it denser around the optimal solution. An individual in the population, called a chromosome, is an arbitrary feasible solution for the problem, and gradually gets improved through the fitness evaluation, crossover, mutation in every iteration. The adaptivity of the algorithm is based on an elite-based immigrants mechanism. It dynamically adjusts the performance of the algorithm according to the current environment, in this way, the solving process is promoted by speeding up the searching for the solution with a proper precision. Finally, the optimal solution (or near optimal solution) can be obtained when multiple iterations complete.

3.2.1 Chromosome Representation

A chromosome is used to represent a path in the search network, and its content corresponds to the sequence of the path. A chromosome contains m' genes, which are use to express the nodes in the path. If node d_{ij} is chosen by a path, its gene is at the ith location in the chromosome, and the content of the gene is j. For example, the correspondence between the sequence S of a path with length of 5 and its chromosome with 5 genes is shown in **Fig. 4**.

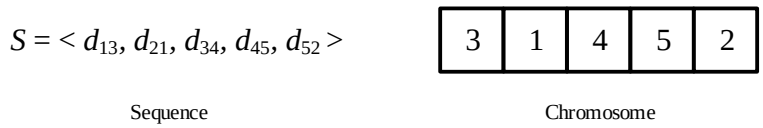


Fig. 4. The sequence of a path and its corresponding chromosome

For a search network with m' levels and p locations at each level, we define a chromosome R_k with index k in current population as

$$R_k =< r_{1k}, r_{2k}, \dots, r_{m'k} >, \qquad r_{ik} \in \{1,2, \dots, p\} \tag{8}$$

3.2.2 Population Initialization

A population is initialized at the start of the algorithm. Multiple different chromosomes are randomly generated. Each of them contains a combination of genes. The number of chromosomes in a population is defined as γ, and the population **U** can be formulated by

$$\mathbf{U} =< R_1, R_2, \dots, R_\gamma > \tag{9}$$

3.2.3 Fitness Evaluation

The fitness evaluation aims at evaluating the qualities of the chromosomes in **U**. The fitness of a certain chromosome R_k, which is expressed by f_k, is the value computed from Formula (7) with the values of its genes substituted. Note that, the lower f_k is, the higher the quality of R_k will be.

3.2.4 Selection

The chromosomes with low fitness values are chosen from the current population through the selection process, in order to promote the average quality of the population. Based on the

fitness value, a stochastic tournament method is used to randomly and multiply generate the subsets of chromosomes from the population. The subsets are called the groups of competitors. The chromosome which possesses the best fitness in each group of competitors is selected according to the following formula

$$R_{\lambda}^{(q)} = \arg\min\left(\left\{f_{\lambda_1}.f_{\lambda_2},\ldots,f_{\lambda_\eta}\right\}\right) \tag{10}$$

where the number of groups is denoted by σ. q is the index number of group, and it satisfies $1 \le q \le \sigma$. The number of chromosomes in a group is denoted by η. λ is the index of the best chromosome in the group, and it satisfies $\lambda \in \{\lambda_1, \lambda_2, \ldots, \lambda_\eta\}$. Thus, $R_{\lambda}^{(q)}$ is the best chromosome with index λ in the qth group.

3.2.5 Crossover

The chromosomes, which are the bests in the groups of competitors, are chosen and bisected into two sets **A** and **B**. The offspring of the population can be generated by the crossover process. Two new chromosomes are formed through recombining one chromosome selected from **A** and one chromosome selected from **B**. In the process of the recombination, some locations in the chromosome are marked based on a probability ρ, and the genes of the two chromosomes at these locations are exchanged correspondingly. As is shown in **0**, two offspring chromosomes are generated from the two parent chromosomes via the crossover.

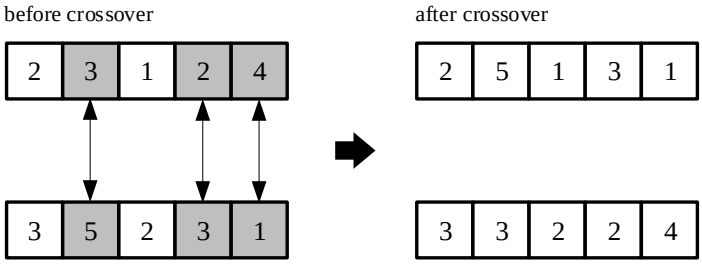


Fig. 5. An instance of the crossover of two chromosomes

3.2.6 Mutation

In order to avoid the solutions represented by the chromosomes in the population from converging into a local optimal point, random mutations at random genes in some chromosomes are carried out with a probability ϵ in the process of mutation. The value of the gene that is assigned to be mutated is replaced by a random value from 1 to p.

3.3 Adaptivity Mechanism

In a time-vary environment, the transmission speeds of the network connections between the terminal and computation devices, the processing speeds of computation devices may change, that is, the values of $\boldsymbol{\delta}^{(c)}$ and $\boldsymbol{\theta}^{(c)}$ in the optimization problem are variable, even during the algorithm execution. Thus, an elite-based immigrants mechanism is used to adaptively adjust the performance of the genetic algorithm. The idea of the mechanism is based on that the current environment is relevant with its previous environment since the values of the parameters in the current environment are the changes of those in its previous to some extent. In most cases, the changes are relatively slight since the environment change is a continuous

process, whereas, in rare cases, the changes may be very severe caused by some bursty factors, such as wireless inferences, device overloads, etc. For the former, the feasible solutions for the previous environment still have certain effectiveness in the current environment. The elite-base immigrants mechanism chooses the elite chromosomes with low fitness values from the chromosomes in the previous population with a proportion, and migrates them to the current population to replace the same amount of bad chromosomes, which have high fitness values. Thus, the searching speed for the current solution can be boosted since the quality of chromosomes are improved, and the convergence to the optimal solution is promoted. For the latter, the algorithm should be re-executed immediately since the solutions for the previous environment have completely expired for the current environment with a huge difference. Therefore, aiming at establishing a continuous mechanism to control relationship between the proportion of elites and the intensity of environment change, which are denoted by ζ and ξ, respectively, ζ should decrease with the increase of ξ, and vice versa.

In addition, the number of iterations in the algorithm can be also adjusted adaptively. The more the number of iterations is, the higher the precision of the solution will be, conversely, the less the number of iterations is, the lower the precision of the solution will be. In an environment with frequent changes, the effectiveness of the solution is prior to the precision of the solution, whereas, in an environment with occasional changes, the precision of the solution is prior to the effectiveness of the solution. Therefore, the number of iterations, which is denoted by τ, should decrease with the increase of ξ, and properly increase with the decrease of ξ.

Thus,the relationships between ζ, τ and ξ are formulated by

$$\zeta \propto \frac{1}{\xi}, \qquad \tau \propto \frac{1}{\xi} \tag{11}$$

where ξ is described by the change of the values of $\boldsymbol{\delta}^{(c)}$ and $\boldsymbol{\theta}^{(c)}$. There are multiple ways to formulate ξ according to different scenarios. Here, considering a time sequence expressed as $< t_1, t_2, \ldots >$, for a certain time t_ω, we give a formulation of ξ as

$$\xi = \beta \frac{\sum_{j=1}^{p} \left(\frac{\left| \delta_j^{(c)}(t_\omega) - \delta_j^{(c)}(t_{\omega-1}) \right|}{\tilde{\delta}_j^{(c)}} \right)}{p} + (1-\beta) \frac{\sum_{j=2}^{p} \left(\frac{\left| \theta_j^{(c)}(t_\omega) - \theta_j^{(c)}(t_{\omega-1}) \right|}{\tilde{\theta}_j^{(c)}} \right)}{p-1} \tag{12}$$

where $\delta_j^{(c)}(t_{\omega-1})$ and $\theta_j^{(c)}(t_{\omega-1})$ are the weights at time $t_{\omega-1}$, and $\delta_j^{(c)}(t_\omega)$ and $\theta_j^{(c)}(t_\omega)$ are the weights at time t_ω. $\tilde{\delta}_j^{(c)}$ and $\tilde{\theta}_j^{(c)}$ are the upper bounds of any $\delta_j^{(c)}(t_\omega)$ and $\theta_j^{(c)}(t_\omega)$, respectively. $0 \leq \beta \leq 1$ is used to balance the values of $\boldsymbol{\delta}^{(c)}$ and $\boldsymbol{\theta}^{(c)}$. It can be seen that ξ is the average proportion of the change of $\boldsymbol{\delta}^{(c)}$ and $\boldsymbol{\theta}^{(c)}$ from time $t_{\omega-1}$ to t_ω.

The numeric relationship between ζ, τ and ξ should be based on Formula (11), and it needs to be configured experientially according to the practical implementation of the algorithm. Here, ζ is given by

$$\zeta = 1 - \xi \tag{13}$$

Similarly, τ is formulated by

$$\tau = \tau^{(base)} + (1-\xi)\tau^{(inc)} \tag{14}$$

where $\tau^{(base)}$ is the basic number of iterations needed by the algorithm to obtain a solution, and $\tau^{(inc)}$ is the upper bound of the incremental iterations. Thus, we can see that τ varies in the range $\left[\tau^{(base)}, \tau^{(base)} + \tau^{(inc)}\right]$ according to ξ.

3.4 Workflow of the Algorithm

The workflow of our genetic algorithm enhanced by elite-based immigrants is described in **Algorithm 1**.

Algorithm 1 **Genetic Algorithm Enhanced by Elite-based Immigrants**

INPUT: ξ, $\boldsymbol{\delta}^{(c)}$ and $\boldsymbol{\theta}^{(c)}$

OUTPUT: $Z^{(\#)}$

1. compute ζ and τ from the formulae based on Formula (11);
2. initialize **U** with $(1-\zeta)\gamma$ randomly generated chromosomes and $\zeta\gamma$ elites from the population at $t_{\omega-1}$;
3. **loop** for τ times
4. carry out fitness evaluation for each chromosome in **U**;
5. carry out selection via the stochastic tournament method;
6. carry out crossover with ρ;
7. carry out mutation with ϵ;
8. **end loop**
9. obtain $Z^{(\#)}$ represented by the best chromosome in **U**.

$Z^{(\#)}$ is the solution obtained by the algorithm, which is possibly a sub-optimal solution, although, it is considered as a solution good enough that is suitable for the current environment at time t_ω.

The terminal monitors $\boldsymbol{\delta}^{(c)}$ and $\boldsymbol{\theta}^{(c)}$ continuously. The execution of **Algorithm 1** is triggered by the value of ξ, which is computed according to Formula (12). We define $\xi^{(th)}$ as the threshold of ξ. If $\xi > \xi^{(th)}$, then the algorithm is executed, whereas, if $\xi \leq \xi^{(th)}$, then the solution obtained from the last execution remains in use.

4. Simulation Results and Evaluations

The performance of our genetic algorithm enhanced by elite-based immigrants is evaluated through simulations under different scenarios and from different aspects. We first measure the performance via the parallel computation offloading and the traditional computation offloading which offloads the computation of an application to the best computation device. The comparison is carried out between them based on the performance with no computation offloading. Then, the adaptivity of our algorithm is evaluated under the scenarios with variable intensities of environment, and its precision of solution and its number of iterations are measured and compared with a standard genetic algorithm without enhancements. The parameters used in simulations are illustrated in **Table 1**.

Table 1. The parameters used in simulations

Name	Meaning	Value
m	the number of vertices in $G^{(a)}$	[3,9]
n	the number of edges in $G^{(a)}$	[3,9]
m'	the number of unoffloadable components	$90\%m$
$\delta_i^{(a)}$	the values of weights in $\boldsymbol{\delta}^{(a)}$	[10,50]MI
$\theta_j^{(a)}$	the values of weights in $\boldsymbol{\theta}^{(a)}$	[0.05,1]MB
p	the number of vertices in $G^{(c)}$	[3,9]
$\delta_i^{(c)}$	the values of weights in $\boldsymbol{\delta}^{(c)}$	[2,256]MIPS
$\theta_j^{(c)}$	the values of weights in $\boldsymbol{\theta}^{(c)}$	[0.256,128]MB/S
γ	the number of chromosomes in $\mathbf{U}$	100
δ	the number of groups of competitors	γ
η	the number of chromosomes in a group of competitors	5
ρ	the probability of crossover	30%
ϵ	the probability of mutation	5%
$\tau^{(base)}$	the basic number of iterations	100
$\tau^{(inc)}$	the upper bound of the incremental iterations	500
$\xi^{(th)}$	the threshold of ξ	0.1

4.1 Performance of the Parallel and Traditional Computation Offloading

The total costs via the parallel computation offloading and the traditional computation offloading are measured under different scales of $G^{(a)}$ and $G^{(c)}$, and they are compared with the total cost via the approach which offloads no components. The topologies and weights of $G^{(a)}$ and $G^{(c)}$ are randomly generated based on **Table 1** unless stated clearly.

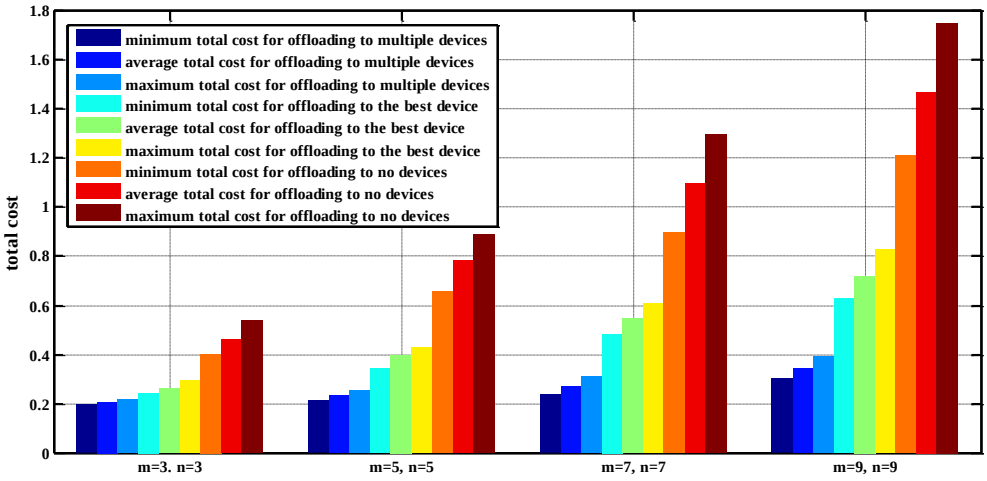


Fig. 6. The average costs via the 3 approaches with variable $G^{(a)}$

When the parameters of $G^{(a)}$ are variable and those of $G^{(c)}$ are fixed, the total costs via the 3 approaches are measured from 100 random cases, the minimum, maximum and average costs are shown in **0**. In the simulations, m and n are picked up from 3 to 9, respectively, and $p = 6$, $m - m' = 1$ (the number of unoffloadable vertices). Generally, the total costs of the 3 approaches all increase with the increase of the scale of $G^{(a)}$, because the amount of the whole computation increases with the increase of the scale of $G^{(a)}$. It can be observed that the average total costs for offloading to no computation devices are the highest for all $G^{(a)}$, since all components of the application are executed in the terminal. On the contrary, the average total costs by the other two approaches that use computation offloading are 30.71% and 51.54% of those via the former approach on average. The parallel computation offloading is better than the traditional approach that offloads the components to the best device. Its average total cost for each $G^{(a)}$ is 79.86%, 58.2%, 49.75% and 47.61% of the corresponding one via the approach that only considers a single device, respectively. The performance promotion is due to the parallelism employed in the computation offloading, so the cost decreases when components are offloaded to different computation devices. It can be also found that the gap between the costs of the latter two approaches increase with the increase of the scale of $G^{(a)}$, because the load of the computation device may rise as the number of components that are offloaded to it increases, it will worsen the performance of computation offloading to a large extent if the components are offloaded to a single device, whereas, the parallel computation offloading allocates the components to different devices, so it can balance the loads of computation devices and alleviate the performance deterioration brought by the increase of the scale of $G^{(a)}$.

When the parameters of $G^{(a)}$ are fixed and those of $G^{(c)}$ are variable, the total costs via the 3 approaches are measured from 100 random cases, the minimum, maximum and average costs are shown in **0.** The value of p is chosen from 3 to 9 in the simulations, and $m = 8$, $n = 8$, $m - m' = 1$ (the number of unoffloadable vertices). It can be seen that the performance via the parallel computation offloading is still the best, and that via the traditional approach is the following, and that via the approach without computation offloading is the worst. Generally, the average total cost via the approach without computation offloading is nearly invariant with the increase of the scale of $G^{(c)}$, since all components run in the terminal and m, n are fixed in the simulations. The average total costs via the other two approaches all decrease with the increase of the scale of $G^{(c)}$, because there are more chances that a component is chosen to offload to a more proper computation devices as the scale of $G^{(c)}$ increases. The average total costs by the two approaches that use computation offloading are 20.03% and 27.19% of those via the former approach on average, and the average total by the parallel computation offloading for each $G^{(c)}$ is 87.19%, 70.25%, 69.35% and 66.17% of the corresponding one via the approach only considering a single device, respectively. It can be found that the gap between the two approaches grows as the scale of $G^{(c)}$ increases, since parallelism is promoted with the increase of the number of computation devices, so there are more choices that components can be offloaded to multiple devices.

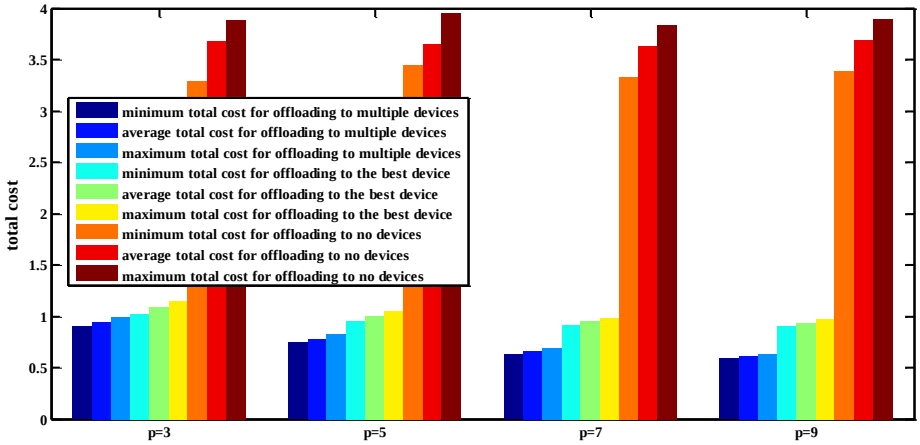


Fig. 7. The average costs via the 3 approaches with variable $G^{(c)}$

4.2 Performance of the Algorithm with Variable Intensities of Environment Change

In order to evaluate the adaptivity of the algorithm, three scenarios with different intensities of environment change are instanced: low, medium and high. The variations of ξ of the three scenarios are shown in **0**, and they are triggered at each time with interval 20s, 10s and 5s, respectively. The values of $\boldsymbol{\delta}^{(c)}$ and $\boldsymbol{\theta}^{(c)}$ for the scenarios first increase before 60s, 30s and 15s, and then decrease after 80s, 40s and 20s, respectively.

In the simulations of the three scenarios, the parameters of $G^{(a)}$ and the topology of $G^{(c)}$ are fixed. The total costs via our algorithm and the standard genetic algorithm are measured every 50 iterations, and the total cost of the corresponding optimal solution at each time interval is also marked as the reference.

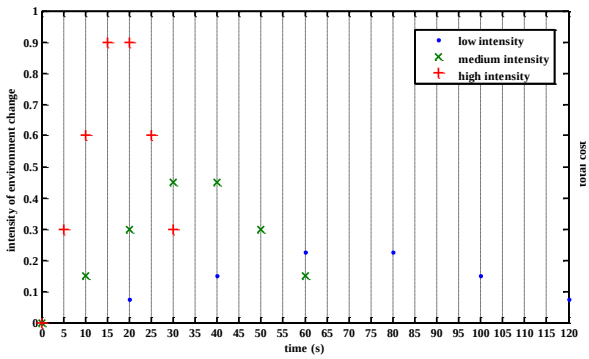


Fig. 8. The value of ξ at different time

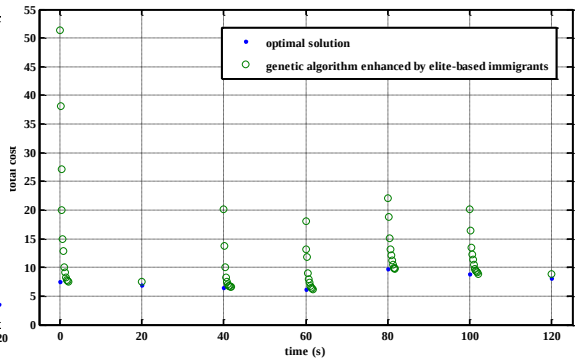


Fig. 9. The total costs in scenarios with low ξ via the genetic algorithm enhanced by elite-based immigrants

4.2.1 Low Intensities of Environment Change

In the scenarios with low intensities of environment change, the values of ξ are chosen from 0% at 0s, increase by 7.5% at 20s, increase by 7.5% at 40s, increase by 22.5% at 60s, decrease by 22.5% at 80s, decrease by 22.5% at 100s, decrease by 22.5% at 120s. The variation of ξ describes the intensity of a relatively stable environment. The total costs via the two algorithms are shown in **0** and **0**. It can be found that the variation of the total costs during the iterations of our algorithm are lower than that during the iterations of the standard genetic algorithm, which demonstrates the fast convergence of our algorithm. As shown in **0**, the standard genetic algorithm takes 600 iterations ($\tau^{(base)} + \tau^{(inc)} = 600$) at each time when the change of ξ is triggered (0s, 20s, 40s, 60s, 80s, 100s, 120s), whereas, according to Formula (14) the iterations consumed by our algorithm is 600 at 0s, 0 at 20s, 488 at 40s, 488 at 60s, 488 at 80s, 525 at 100s, 0 at 120s. At time 0s and 120s, our algorithm is not executed due to $\xi = 0.075 < \xi^{(th)} = 0.1$, thus, verbose executions for slight variation of ξ are avoided, and algorithm cost is alleviated in this way. The average errors between the optimal value and the values computed from the two algorithm are 0.37% and 0.31%, respectively, which prove that, in environments with low intensities, through the mechanism of elite-based immigrants, our algorithm can approach a high precise solution which is quite close to that via the standard genetic algorithm that uses more iterations.

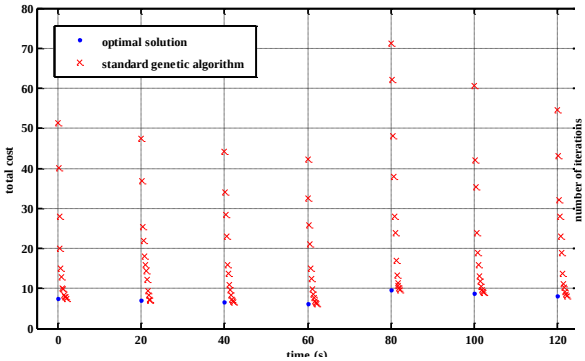


Fig. 10. The total costs in scenarios with low ξ via the standard genetic algorithm

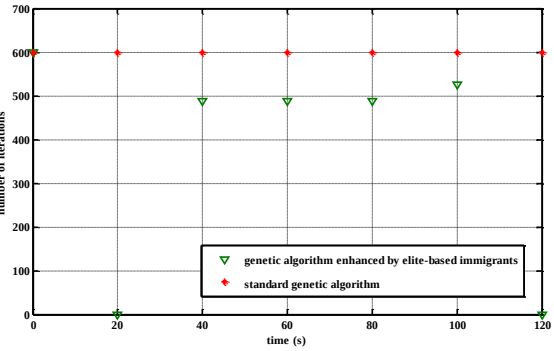


Fig. 11. The number of iterations with low ξ via the our algorithm and the standard genetic algorithm

4.2.2 Medium Intensities of Environment Change

In the scenarios with medium intensities of environment change, the values of ξ are chosen from 0% at 0s, increase by 15% at 10s, increase by 30% at 20s, increase by 45% at 30s, decrease by 45% at 40s, decrease by 45% at 50s, decrease by 30% at 60s. The total costs via the two algorithms are shown in **0** and **0**. In the simulations, our algorithm is triggered at every time interval since the corresponding $\xi > 0.1$. The convergence speed of our algorithm is faster than that of the standard genetic algorithm due to the elite-based immigrants mechanism. As regards to our algorithm, it can be found that the variations of the total costs during the iterations are diverse for different time. Referred to the performance of the standard genetic algorithm at each time interval, the variations of our algorithm is low at the time with low ξ (10s, 20s, 50s, 60s), whereas, the variations increase properly at the time with high ξ (30s,

40s). This is because that ζ decreases with the increase of ξ according to Formula (13). The immigrants become less valuable when the intensity of the environment increases, conversely, they may disturb the convergence of the solving process, and make the solution converge at a suboptimal location. As shown in **0**, the standard genetic algorithm takes 600 iterations at each time when the change of ξ is triggered (0s, 10s, 20s, 30s, 40s, 50s, 60s), whereas, the iterations consumed by our algorithm is 600 at 0s, 525 at 10s, 450 at 20s, 375 at 30s, 375 at 40s, 450 at 50s, 525 at 60s. The average errors between the optimal value and the values computed from the two algorithms are 1.62% and 1.18%, respectively. It demonstrates that in environments with medium intensities, our algorithm can reach a precise solution with less iterations.

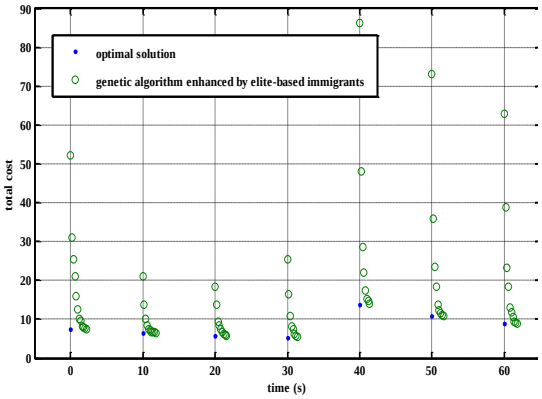


Fig. 12. The total costs in scenarios with medium ξ via the genetic algorithm enhanced by elite-based immigrants

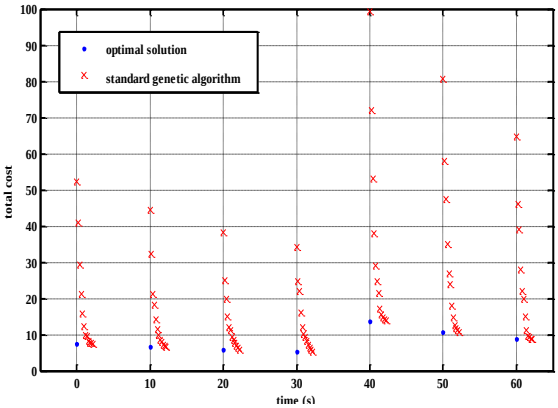


Fig. 13. The total costs in scenarios with medium ξ via the standard genetic algorithm

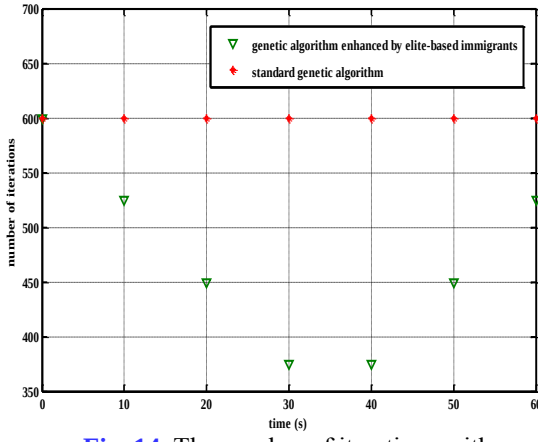


Fig. 14. The number of iterations with medium ξ via the our algorithm and the standard genetic algorithm

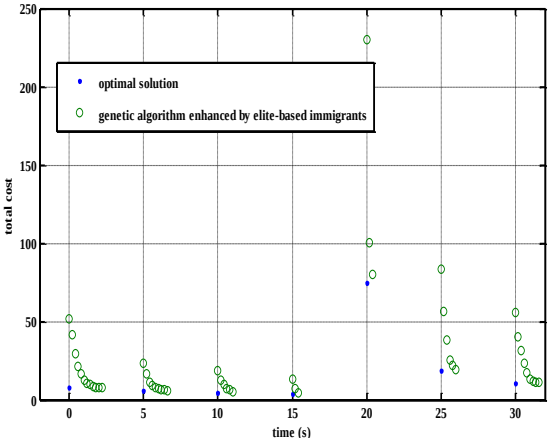


Fig. 15. The total costs in scenarios with high ξ via the genetic algorithm enhanced by elite-based immigrants

4.2.2 High Intensities of Environment Change

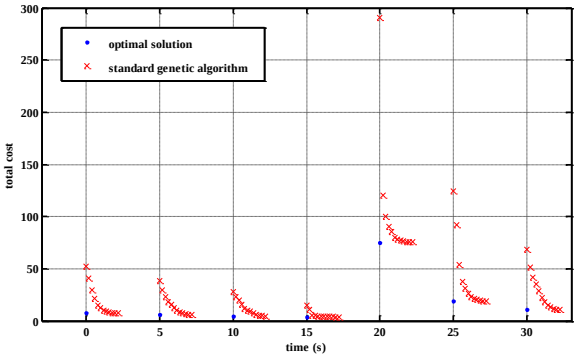


Fig. 16. The total costs in scenarios with high ξ via the standard genetic algorithm

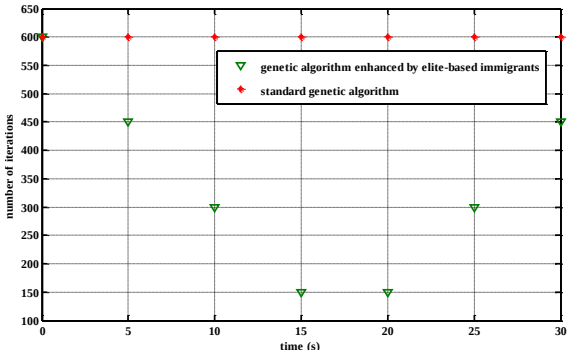


Fig. 17. The number of iterations with high ξ via the our algorithm and the standard genetic algorithm

In the scenarios with high intensities of environment change, the values of ξ are chosen from 0% at 0s, increase by 30% at 5s, increase by 60% at 10s, increase by 90% at 15s, decrease by 90% at 20s, decrease by 60% at 25s, decrease by 30% at 30s. The simulation results via the two algorithms are shown in **0** and **0**. In the environments with high ξ, the time consumed by the algorithms is vital since the effectiveness is more important than the precision of the solution. In a highly variable environment, the solution may expire if too long time is taken by the algorithm to obtain it, so the execution time of the algorithm needs to decrease to prolong the life time of the solution. In the simulation, It can be found that τ decreases with the increase of ξ, this is due to the mechanism of iteration control in our algorithm according to Formula (12). As shown in **0**, the iterations used by our algorithm for each time interval are 600 at 0s, 450 at 5s, 300 at 10s, 150 at 15s, 150 at 20s, 300 at 25s, 450 at 30s, whereas, the iterations of the standard genetic algorithm are all 600. The average errors between the optimal value and the values computed from the two algorithm are 5.16% and 4.25%, respectively, which proves that in the environments with high intensities, our algorithm still keeps a relative precious solution while restricting the number of iterations.

5. Conclusion

Computation offloading is a promising technology to alleviate the contradiction between computation-occupying applications and resource-constrained terminals. This paper focuses on the application multi-partitioning problem for parallel computation offloading, and proposes an adaptive application component mapping algorithm for parallel computation offloading in variable environments. The algorithm is under the scenarios that the components of an application are offloaded in parallel to multiple computation devices around the terminal, and it models the multi-partitioning problem as a graph mapping model, converts it into a pathfinding problem, then uses a genetic algorithm enhanced by elite-based immigrants to obtain suitable mapping results for the environments with variable transmission speeds of network connections and processing speeds of computation devices. An adaptivity mechanism is designed to adaptively adjust the precision of the solution and promote the searching speed through change the number of iterations and the proportion of elite immigrants. Simulation results demonstrate that our algorithm can promote the performance of parallel computation offloading efficiently, and its adaptivity in variable environments outperforms traditional

approaches to a large extent.

References

[1] X. Ma, Y. Zhao, L. Zhang, H. Wang, and L. Peng, "When mobile terminals meet the cloud: computation offloading as the bridge," *IEEE Network*, vol. 27, no. 5, pp. 28–33, 2013. Article (CrossRef Link)

[2] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proc. of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62, ACM, 2010. Article (CrossRef Link)

[3] H. Wu, Q. Wang, and K. Wolter, "Tradeoff between performance improvement and energy saving in mobile cloud offloading systems," in *Proc. of Communications Workshops (ICC), 2013 IEEE International Conference on*, pp. 728–732, IEEE, 2013. Article (CrossRef Link)

[4] J. Oueis, E. C. Strinati, and S. Barbarossa, "Multi-parameter decision algorithm for mobile computation offloading," in *Proc. of Wireless Communications and Networking Conference (WCNC), 2014 IEEE*, pp. 3005–3010, IEEE, 2014. Article (CrossRef Link)

[5] S. Ou, K. Yang, and A. Liotta, "An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems," *Pervasive Computing and Communications, 2006, PerCom 2006. Fourth Annual IEEE International Conference on*, pp. 116–125, IEEE, 2006. Article (CrossRef Link)

[6] K. Sinha and M. Kulkarni, "Techniques for fine-grained, multi-site computation offloading," in *Proc. of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 184–194, IEEE Computer Society, 2011. Article (CrossRef Link)

[7] C. Wang, Y. Li, and D. Jin, "Mobility-assisted opportunistic computation offloading," *Communications Letters, IEEE*, vol. 18, pp. 1779–1782, Oct. 2014. Article (CrossRef Link)

[8] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, pp. 14–23, Oct 2009. Article (CrossRef Link)

[9] H. Wu, Q. Wang, and K. Wolter, "Methods of cloud-path selection for offloading in mobile cloud computing systems," in *Proc. of the 4th IEEE International Conference on Cloud Computing Technology and Science*, pp. 443–448, 2012. Article (CrossRef Link)

[10] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud," in *Proc. of the 2012 IEEE10th International Symposium on Parallel and Distributed Processing with Applications, ISPA '12*, pp. 784–791, IEEE Computer Society, 2012. Article (CrossRef Link)

[11] C. Xian, Y.-H. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *Proc. of the13th International Conference on Parallel and Distributed Systems -Volume 01, ICPADS '07*, pp. 1–8, IEEE Computer Society, 2007. Article (CrossRef Link)

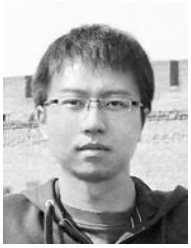
[12] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic,"Adaptive offloading inference for delivering applications in pervasive computing environments," *Pervasive Computing and Communications, 2003. (PerCom 2003)*, in *Proc. of the First IEEE International Conference on*, pp. 107–114, March 2003. Article (CrossRef Link)

[13] J. Niu, W. Song, L. Shu, and M. Atiquzzaman, "Bandwidth-adaptive application partitioning for execution time and energy optimization," in *Proc. of Communications (ICC), 2013 IEEE International Conference on*, pp. 3660–3665, June 2013. Article (CrossRef Link)

[14] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," *INFOCOM, 2012 Proceedings IEEE*, pp. 945–953,IEEE, 2012. Article (CrossRef Link)

[15] L. Wang and M. Franz, "Automatic partitioning of object-oriented programs for resource-constrained mobile devices with multiple distribution objectives," *Parallel and Distributed Systems, 2008. ICPADS'08. 14thIEEE International Conference on*, pp. 369–376, IEEE, 2008. Article (CrossRef Link)

[16] F. Sadri, "Ambient intelligence: A survey," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 36, 2011. Article (CrossRef Link)
[17] C. H. Papadimitriou and K. Steiglitz, "Combinatorial optimization: algorithms and complexity," Courier Corporation, 1998. Article (CrossRef Link)
[18] L. Davis et al., "Handbook of genetic algorithms," vol. 115. Van Nostrand Reinhold New York, 1991. Article (CrossRef Link)



Wenhao Fan received the B.E. and Ph.D. degree from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2008 and 2013, respectively. He is currently an assistant professor and a supervisor of postgraduate at the School of Electronic Engineering in BUPT. His main research topics include mobile computing, parallel computing and transmission, information security for mobile terminals, and software engineering for mobile internet.

Yuan'an Liu received the B.E., M.Eng., and Ph.D. degrees in electrical engineering from the University of Electronic Science and Technology, Chengdu, China, in 1984, 1989, and 1992, respectively. In 1984, he joined the 26th Institute of the Electronic Ministry of China to develop the inertia navigating system. In 1992, he began his first Postdoctoral position with the EMC Laboratory, Beijing University of Posts and Telecommunications (BUPT), Beijing, China. In 1995, he started his second Postdoctoral position with the Broadband Mobile Laboratory, Department of System and Computer Engineering, Carleton University, Ottawa, ON, Canada. Since 1997, he has been a Professor with BUPT. He is currently the Dean of the School of Electronic Engineering, BUPT. His research interests include pervasive computing, wireless communications and electromagnetic compatibility. Dr. Liu is a Fellow of the Institution of Engineering and Technology, U.K.; the Vice Chairman of Electromagnetic Environment and Safety of the China Communication Standards Association; the Vice Director of the Wireless and Mobile Communication Committee, Communication Institute of China; and a Senior Member of the Electronic Institute of China.

Bihua Tang received the B.E. degree from the Sichuan University, Chengdu, China, in 1984, and M.Eng. degree from the University of Electronic Science and Technology, Chengdu, China, in 1989, respectively. She is currently a professor at the School of Electronic Engineering in Beijing University of Posts and Telecommunications, Beijing, China. Her main research topics include wireless sensor networks, wireless networks and hardware engineering for sensors. She has published more than 50 papers.