Optimal k-copies Task Assignment Reliability in Distributed Systems, via a Genetic Algorithm

Chin-Ching Chiu* and Chung-Hsien Hsu

Department of Management Information System

Private Takming College, Taipei, Taiwan, R.O.C.

TEL: 886-2-26585801 ext 393

E-mail: chiu@mis.takming.edu.tw

*Correspondence and reprints should be sent to Chin-Ching Chiu.

Abstract

The reliability-oriented task assignment problem, which is NP-hard, is to find a task distribution such that the program reliability or system reliability is maximized. In this paper, we have developed a reliability oriented task allocation scheme, based on a genetic algorithm, for distributed systems to find an approximate solution. The simulation shows that, in most test cases, the algorithm finds sub-optimal solutions efficiently; therefore, it is a desirable approach to solve these problems.

Keywords: Distributed system reliability; Task assignment; Distributed program reliability; Genetic algorithm

1. INTRODUCTION

Distributed systems (DS) have become increasingly popular in recent years. The potential reliability improvement of a distributed system is possible because of program and data-file redundancy. Reliability evaluations of distributed systems have been widely published [1-8]. To evaluate the reliability of a distributed system, including a given distribution of programs and data-files, it is important to obtain a global reliability measure that describes the degree of system reliability [10-15].

For a given distribution of programs and data files in a DS, distributed program reliability (DPR) [9] is the probability that a given program can be run successfully and will be able to access all of the files it requires from remote sites in spite of faults occurring among the processing elements and communication links. The second measure, DSR, is defined as the probability that all of the programs in the system can be run successfully. Kumar, Hariri, Raghavendra [9] has also shown that redundancy in resources such as computers, programs, and data-files can improve the reliability of distributed systems [9]. Therefore, the study of program and data-file assignment with redundancy considerations is important in improving the DSR.

Assume that there are n processing nodes, P programs, F data files and k copies. Then the total number of possible assignment are $n^{k(P+F)}$. Thus, the optimal allocation of programs and files on the processing nodes is a problem of exponential complexity [15]. This implies that optimum solutions can be found only for small problems. For larger problems it is necessary to introduce heuristics to produce algorithms, which generate near-optimum solutions. Genetic algorithm (GA) can be applied to search large, complex problem spaces [19]. The main steps for GA are reproduction, selection, crossover, and mutation. The process of

selection, crossover, and mutation is repeated until the termination condition is satisfied [16-20].

Hwang and Tseng proposed the *k*-DTA problem. The *k*-DTA models the assignment of *k* copies of both distributed programs and their data-files to maximize the DSR under some resource constraints. Since the *k*-DTA problem is NP-hard [13], this study proposed an algorithm based upon genetic algorithm [19] to find an approximate solution.

2. COMPUTING OPTIMAL RELIABILITY

In this section, we describe the problem addressed herein for convenience and clarification of our research objectives.

2.1 Notations and Definitions

The following notations, and definitions are used hereinafter.

NOTATIONS.

$G=(V,E)$	an undirected DS graph where V denotes a set of processing elements, and E represents a set of communication links.	$MFST(p_i)$	set of minimal file spanning trees Associated with program p_i.
		FST	file spanning tree consisting of the root node (processing element that runs the program) and some other nodes which hold all the files needed for the program held in the root node under consideration.
n	the number of nodes in G, $n=\|V\|$.		
v_i	an i^{th} node represents the i^{th} processing element.		
$c(v_i)$	the capacity of the i^{th} node.		
e	the number of links in G, $e=\|E\|$	$s_{need}(v_i)$	the total capacity requires from v_i.
$e_{i,j}$	an edge represents a communication link between v_i and v_j.	ng	generation of GA, for $ng=0,\ldots,tng$.
		mr, cr	mutation rate, crossover rate.

$a_{i,j}$	the probability of success of link $e_{i,j}$	**mc, cc**	mutation count, crossover count.
$b_{i,j}$	the probability of failure of link $e_{i,j}$	**ps**	population size. represents total
$d(v_i)$	the number of links connected to the	**tng**	number of generation when GA end.
	node v_i.	$x_{n(P+F)-},\ldots,x_0$	a binary string with a length $n(P+F)$
$w(v_i)$	the weight of the i^{th} node.		represent an chromosome, where x_i=1
$w(e_{i,j})$	the weight of the link $e_{i,j}$.		if v_i is selected, otherwise x_i=0,
F	total number of files in the DS.		for i=0, …,$n(P+F)$-1.
P	total number of programs in the DS.	FT_i	the actual fitness value of a chromo-
p_i	distributed program i.		some i in a generation.
f_i	file i.	$SUMFT$	total fitness value of all the chromo-
$s(p_i)$	the size of program p_i.		some in a generation, i.e.,
$s(f_i)$	the size of data file f_i.		$SUMFT=\sum_{i=1}^{ps} FT_i$.
pf_i	distributed program or file i.	RFT_i	a proportion of a roulette-wheel slot-
k	the number of copies of pf_i.		sized of chromosome i in a generation,
AFL(p_i)	list of files required for p_i to		i.e., $RFT_i = FT_i / SUMFT$.
	complete its execution.	$roulette_i$	accumulation of FT_i,i.e.,
DPR(p_i)	distributed program reliability of p_i.		$roulette_i = \sum_{k=1}^{i} FT_i$.
MFST	minimal FST containing no subset	$AVGFT$	average fitness value, i.e.,
	file spanning tree.		$AVGFT= SUMFT /ps$.

Definitions

Definition 1. A dependent set is defined as a set S of distributed programs and files such that there does not exist a partition which divides S into two disjoint subsets S_1 and S_2, where $S_1 \cup S_2=S$, and $S_1 \cap S_2= \varnothing$, such that each program and the files required are within the same subset [13].

Definition 2. A DTA problem is defined as to find an assignment for a dependent set under some resource constraints in the distributed system such that the distributed system reliability is maximal [13].

Definition 3. A *k*-DTA problem is defined as to determine assignments for *k* copies of a dependent set to maximize the DSR under some resource constraints in the distributed system [13].

Definition 4. A mask string is defined as a string with a length $n(P+F)$ in which each bit indicates whether the capacity of a node is sufficient to be allocated a program or data file.

2.2 Problem Statements

A distributed system can be modeled by a simple undirected graph. For a topology of the DS with four nodes and five links, if two programs and three data files that should be allocated, the number of different combinations of programs and data files for allocation is 4^5, that is 1024. The program p_1 requires data files f_1, f_2, and p_2 requires data files f_1, f_2, f_3, for completing execution. Assume that these files are allocated as shown in figure 1.

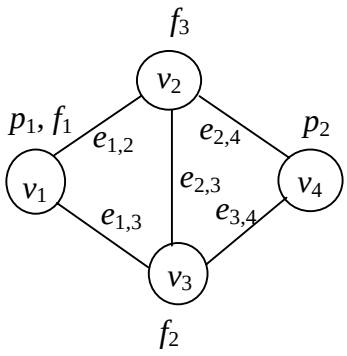


Fig. 1. The DS with four nodes and five links

For program p_1, there are three MFSTs, such as $v_1 v_3 e_{1,3}$, $v_1 v_3 e_{1,2} e_{2,3}$, $v_1 v_3 e_{1,2} e_{2,4} e_{3,4}$. Therefore, DPR($p_1$) = pr($\bigcup_{i=1}^{3} MFST_i$). Once all of the minimal files spanning trees have been generated, the next step is to find the probability that at least one MFST is working

which means that all of the edges and vertices included in it are operational. The reliability of program p_1 can be computed by means of a sum of mutually disjoint terms [4].

DPR(p_1) = $b_{1,2}\,a_{1,3} + a_{1,2}\,a_{1,3}\,b_{2,3}\,b_{2,4} + a_{1,2}\,a_{1,3}\,b_{2,3}\,a_{2,4}\,b_{3,4} + a_{1,2}\,a_{2,3}\,a_{2,4}\,b_{3,4} + a_{1,2}\,a_{2,3}\,b_{2,4} + a_{1,2}\,a_{2,4}\,a_{3,4}$.

Assume that the probability of each link is 0.9. Then DPR(p_1) = 0.98829.

In the same way, DPR(p_2) = pr($\bigcup_{i=1}^{8} MFST_i$) = 0.97686.

Therefore, the results of DSR is pr($\cap_{i=1}^{p} p_i$) = pr($\cap_{i=1}^{p} MFST(p_i)$) = 0.97686.

A reliability-oriented task assignment problem can be characterized as follows:

Given : distributed system parameters, memory capacity of each node, memory requirement of each program and data file, number of copies of each program and data file, files required by each distributed program for execution.

Object: Maximize DSR = $\Pr\left[\bigcap_{i=1}^{p} E(p_i)\right]$

subject to: $(\sum_{j=0}^{P} ps_{j-1}x_{n\times j} + \sum_{j=P}^{P+F} fs_j x_{n\times j}) \le C_i$, i=1, …, n

$\sum_{i=1}^{n} x_{j\times n+i} = k,\ x_{i,j} = 0\ or\ 1,\ j = 1,...,P+F,\ k$ *is the number of copies of* pf_i.

3. GENETIC ALGORITHM BASED RELIABILITY-ORIENTED TASK ASSIGNMENT METHODOLOGY

The GA search space is composed of possible solutions (chromosomes) to the problem. Each chromosome has an associated objective function value called a fitness value which denotes its strength. A set of chromosomes and their associated fitness values are called the population. This population at a given stage of GA is referred to as a generation.

3.1 Development of GAROTA

The development of GAROTA is described in the following subsections.

3.1.1 Chromosomal-coding scheme

If P, F and n represent the number of programs, data files and nodes, respectively, that a chromosome, say X, with a length $n\times(P+F)$ of the following type:

$$\underbrace{\quad\quad f_F \quad\quad}\ \cdots\ \underbrace{\quad\quad f_1 \quad\quad}\ \underbrace{\quad p_P \quad}\ \cdots\ \underbrace{\quad p_1 \quad}$$

$$v_{n-1}\ v_{n-2}\ \ldots v_0\ \ldots v_{n-1}\ v_{n-2}\ \ldots v_0\ v_{n-1}\ v_{n-2}\ \ldots v_0\ \ldots v_{n-1}\ v_{n-2}\ \ldots v_0$$

$$x_{(P+F)n-1}x_{(P+F)n-2}\ \ldots x_{(P+F-1)n}\ \ldots x_{(P+1)n-1}\ x_{(P+1)n-2}\ \ldots x_{Pn}\ x_{Pn-1}\ x_{Pn-2}\ \ldots x_{(P-1)n}\ \ldots\ x_{n-1}\ x_{n-2}\ \ldots\ x_0$$

Each bit indicates whether a program or file is allocated in a node, where x_i =1, if it is occurrs, otherwise x_i =0. For example, if n=5, P=1, F=2, k=2

<----------f_2------- > <----------f_1------- > <----------p_1------- >

x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
1	0	0	1	0	0	1	0	1	0	0	0	1	0	1

This chromosome shows p_1 is allocated in node v_2 and v_0, f_1 is allocated in node v_3 and v_1, and f_2 is allocated in node v_4 and v_1.

3.1.2 Valid Chromosomes

The most reliable assignment for k copies of some program or data file is to assign these copies to k distinct nodes [13].

For a valid chromosome, it must possess two characteristics as follows:

(1) For each sub-string, say $s_i = x_{(i+1)\times n+1}, \ldots, x_{i\times n}$, of a chromosome, where i is in the range (0, $P+F$-1), the number of bits whose value is 1 is equal to the k.

(2) The summation of the programs and files size, which is assigned to same node is at most as large as the capacity of the node.

3.1.3 Initialization approach

The initial population can be randomly created or well adapted [20]. GAROTA was randomly created to generate an unbiased population at initialization.

The size and format of a mask string, say M, is same as those of a chromosome. This string can avoid generating an invalid chromosome and speed up the GA generation initialization. In our simulation case, if the mask string is omitted, about 80~100 invalid strings shall be generated before obtain a valid chromosome. For each bit of M, say $m_{i\times n+j}$, the value is set as follows:

$$m_{i\times n+j} = \begin{cases} 1 \; if \;\; c_j \geq ps_i, \; and \;\; i < P, \\ 1 \; if \;\; c_j \geq fs_{i-P}, \; and \;\; i \geq P, \\ 0 \quad otherwise \end{cases},$$

where $0 < i < P+F$, $0 < j < N$.

When a chromosome is generated according to the mask string M, we could always obtain a valid chromosome and could omit checking whether it is a valid chromosome. The chromosome satisfies our requirement and is appended to the population.

3.1.4 The object function

The number of ports at each node (degree of a node) and number of links directly impact the system reliability. Reliability decreases with a decrease in the number of links [5]. Therefore, we employed a simple means for computing the node weight, which takes less time and can quickly compute the weight of every node.

$$w(v_i) = 1 - \prod_{z=1}^{d(v_i)} b_{i,t_z} \quad (1)$$

In the network, two nodes may contain many paths between them. The length of a path is between one and *n*-1. To reduce the computational time, we considered a path in which the length is not greater than three. The following recursive formula was used to evaluate the weight of 2-terminals.

$$w2term(v_i, v_j) = \begin{cases} 0, & initialize, \\ w2term(v_i, v_j) + (1 - w2term(v_i, v_j)) \times A, \end{cases} \quad (2)$$

where

A is a probability of a path that select from the sets S_1, S_2 and S_3 one after another,

$S_1=\{a_{i,j} \mid \text{if } e_{i,j} \text{ is exists}\}$, $S_2=\{a_{i,t}a_{t,j} \mid \text{if } e_{i,t}, e_{t,j} \text{ is exist}\}$, $V_{s2}=\{v_t \mid \text{if } e_{i,t}, e_{t,j} \text{ is exist}\}$,

$S_3=\{a_{i,x}a_{x,y}a_{y,j} \mid \text{if } e_{i,x}, e_{x,y}, e_{y,j} \text{ is exist}, v_x, v_y \notin V_{s2} \text{ and } v_x, v_y \text{ has not occurred}\}$.

We can obtain the 2-terminal access weight of all pairs according to formula 2.

$$w2termaccess(v_i, v_j) = [w2term(v_i, v_j) + w(v_i) + w(v_j)]/3 \quad (3)$$

If a program p_i is allocated in v_x and a file f_i is allocated in v_y, the file f_i is needed when p_i is run. If the set SS(p_i) represents all of the pairs (v_x, v_y), i.e. $SS(p_i)=\{ (v_x, v_y) \mid v_x$ holds a copy of program p_i, v_y hold a copy of data file f_j, $f_i \in \text{AFL}(p_i)$, $x \neq y\}$, we can use the following formula to compute the access weight of each program as follows.

$$AW(p_x) = \prod w2termaccess(v_x, v_y) \quad (4)$$

where $(v_x, v_y) \in SS(p_i)$.

Therefore, the access weight of chromosome X which denotes all the programs and data files allocated in some node is computed as follows.

$$AW(X) = \prod w2termaccess(v_x, v_y) \quad (5)$$

where $(v_x, v_y) \in \bigcup_{i=1}^{p} SS(p_i)$ and if both of pairs (v_x, v_y), (v_y, v_x) are exist, discard (v_y, v_x).

According to the access weight of chromosome X, the object function to compute the fitness value of X is constructed as follows:

$$FT_i = AW(X) \times AW(X) \quad \textbf{(6)}$$

The object function used to compute the fitness value of each i^{th} chromosome X in generation j. Fitness values indicate which chromosomes are to be carried into the next generation. The reason for using the value of the square of $AW(X)$ as the fitness value is expansion of the difference between two chromosomes. This will lead to a speed up of the population convergence.

3.1.5 Genetic reproduction and selection

The process for selecting potentially good strings from the current generation is to be carried into the next generation. This is achieved by assigning a proportionately higher fitness value [16,18]. A "biased" roulette wheel [20] is used for chromosome selection into the mating pool.

3.1.6 Genetic crossover operators

The crossover is performed at the boundaries of the node bits. First, two chromosomes are randomly selected from the mating pool. Next, using a random number generator, an integer is generated in the range (1, P+F-1). This number is used as the crossover site. The result produces two new chromosomes with information from their parents. For example,

<---------f_2------- > | <---------f_1------- > <---------p_1------- >

x_{14} x_{13} x_{12} x_{11} x_{10} | x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0

String1	1	0	0	1	0	0	1	0	1	0	0	0	1	0	1
String2	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1

If the crossover site is 2, the information exchange occurs as:

	<---------f_2-------- >					<---------f_1-------- >					<---------p_1-------- >				
	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Child1	1	0	0	1	0	0	0	1	1	0	0	0	0	1	1
Child2	0	1	1	0	0	0	1	0	1	0	0	0	1	0	1

The crossover operator sometimes generates an invalid chromosome. For example, If the size of each node is 5, the size of p_1 is 2, f_1 is 3 and f_2 is 2, respectively. Because p_1, f_1 and f_2 should be allocated to v_1 in child1, the summation size is 7 which is exceeds the capacity of v_1. This anomaly is just discarded.

3.1.7 Genetic mutation operator

This operator is used to improve the global optimal solution, if it is appreciably reduced by the crossover operation. First, using a random number generator, three integers (say x, y, z) are generated. The value of x is in the range (0, ps-1), which indicates the mutation chromosome. The value of y is in the range (0, P+F-1) which indicates the bits between $y \times n$ and $((y+1) \times n)$-1 of the mutation chromosome. The value of z is in the range (0, n-1) which indicates the mutation bit, i.e., the $(y \times n+z)^{th}$ bit of the mutation chromosome and mutates it. Ensures that the copy of each program and data file is correct, we select a bit randomly in the range ($y \times n$, $y \times (n+1)$-1) bit of the mutation chromosome and adjust it.

The mutation operator sometimes generates a chromosome, which does not represent a valid task assignment. When this situation occurs, the original chromosome is reserved and another chromosome is selected for mutation.

3.1.8 Replacement strategy and termination rules

The most common replacement strategy is to probabilistically-replace the poorest performing chromosome in the previous generation [20]. On the other hand, the elitist strategy appends the best performing chromosome from the previous generation to the current population and, thereby, ensures that the chromosome with the best objective function value always survives to the next generation. Our GAROTA combines both of these concepts. Each offspring generated after crossover is added to the new generation if it has a better objective function value than both of its parents. If the objective function value of an offspring is better than only one of the parents, a chromosome is randomly selected from the better of the two parents and the offspring. If the offspring is worse than both parents, any one of the parents is selected at random for the next generation. This ensures that the best chromosome is carried into the next generation, while the worst is not.

GAROTA execution can be terminated when the average and maximum fitness values of the strings in a generation become the same.

3.2 Complete Algorithm of GAROTA

The algorithm begins with an initial generation of valid chromosomes, which satisfy the constraint. The initial generation contains a finite number of valid strings selected by random. The number of strings in any generation, population size, is kept an even number to ease the crossover. The detail steps for GAROTA are described as follows:

Algorithm GAROTA

step 0 /*Initialize DS, GA, and task parameters */

Read parameters: n, e, $a_{i,j}$, $c(v_i)$, ps, cr, mr, ng, tgn, k, P, F, $s(p_i)$, $s(f_i)$, AFL(p_i).

step 1 Compute each node weight using Eq. (1), all 2-terminal weight using Eq. (2) and all 2-terminal access weight using Eq. (3).

step 2 /* Generate each chromosome of the initial population. */

2.1 Initialize the mask string *M* for generating chromosomes quickly.

2.2 *ng*=0. /* generation 0 */

2.3 /* generate each valid chromosome and compute its fitness value for generation 0. */

for (i=0; $i < ps$; i++){

generate a valid chromosome $chrom_i$.

FT_i = evaluate access weight of $chrom_i$ using Eq. (5) }

step 3 Generate roulette-wheel area of each chromosome according to its fitness value.

step 4 /*Reproduction/Selection for mating pool*/

Select *ps* chromosomes randomly according to roulette-wheel and generate the mating pool.

step 5 /*Crossover for next generation. */

$cc = \lceil cr \times ps/2 \rceil$. /* set the crossover count. */

Dowhile (*cc*-- > 0)

generate two valid chromosomes, say *tmp*1 and *tmp*2, by crossover the two chromosomes, say $pool\_chrom_x$ and $pool\_chrom_y$, which are random select from mating pool.

evaluate access weight of *tmp*1 and *tmp*2 using Eq. (5).

relpace $pool\_chrom_x$, $pool\_chrom_y$ by the chromosomes which are related to the two maximized fitness values of these four chromosomes.

Enddowhile

step 6 /* Mutate for next generation. */

$mc = \lceil mr \times ps \rceil$. /* Set the mutation count. */

Dowhile (*mc*-- > 0)

generate a valid chromosome, say *tmp*3, by mutate an arbitrary bit of the chromosome, say $pool\_chrom_x$, which are random select from mating pool.

evaluate access weight of *tmp*3 using Eq. (5)

replace $pool\_chrom_x$ by *tmp*3 if its fitness value better.

Enddowhile

step 7 Replacement and creation a new population

step 8 /*Test for terminating condition*/

$ng = ng + 1$. /* next generation */

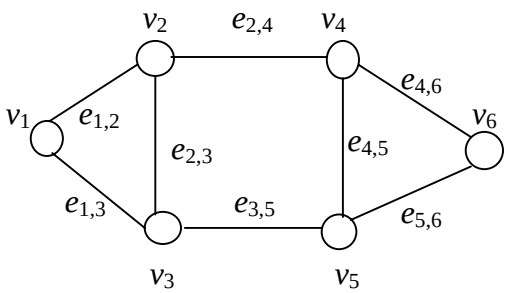
if ($ng \leq tng$ and some $FT_i \neq AVGFT$) go to step 3.

step 9 /*Compute the DSR and output the best task assignment.*/

Compute the reliability of the final result task assignment indicate at the first chromosome of the population using SYREL [4] and Output the task assignment.

End GAROTA

3.3 An Illustrate Example



$c(v_1)=5$ $c(v_2)=4$ $c(v_3)=6$
$c(v_4)=6$ $c(v_5)=5$ $c(v_6)=5$

$a_{1,2}=0.70$ $a_{1,3}=0.80$ $a_{2,3}=0.90$ $a_{2,4}=0.75$
$a_{3,5}=0.95$ $a_{4,5}=0.85$ $a_{4,6}=0.90$ $a_{5,6}=0.95$

Fig. 2. The DS with six nodes and eight links

In figure 2, if there are two programs, p_1, p_2, and three data files, f_1, f_2, f_3, the size of p_1, p_2, f_1, f_2, f_3 is 2, 3, 2, 3 and 3, respectively. The program p_1 needs f_1, f_2, and program p_2 needs f_1, f_2,

f_3 for complete execution, e.g., AFL(p_1) is {f_1, f_2}, AFL(p_2) is {f_1, f_2, f_3}. Our problem is to find the maximal distributed system reliability under the allocated programs and files.

In step 1, after evaluating each 2-terminal pair's weight using Eq. 3, the weight of (v_1 ,v_2), (v_1 ,v_3), …, (v_1 ,v_6), (v_2 ,v_3), …, (v_2 ,v_6), …, (v_5 ,v_6) is 0.9495, 0.9550, 0.9227, 0.9356, 0.9295, 0.9914, 0.9735, 0.9799, 0.9754, 0.9776, 0.9924, 0.9852, 0.9960, 0.9907 and 0.9943, respectively.

In step 2, initialize population.

In step 3, each chromosome's fitness value, ratio of fitness value, and roulette-wheel area are derived. The average fitness value is 0.490881.

The algorithm executes statements between step 4 and step 8. They are reproduction and selection for the mating pool, crossover and mutation for the next generation, replacement and creation of the new generation, and testing for the termination condition. In addition, step 3 is executed again. The average fitness value of generation 1 is 0.537424.

In the same way, the average fitness value of generation 10, …, 80 is obtained as shown in Fig. 3. In generation 80, all chromosomes are 10010001100010001000110001001O and the fitness value is 0.748307. The average fitness value is 0.748307. Figure 3 illustrates the average fitness value for every generation. Because the termination condition is satisfied, the algorithm goes to step 9.

In step 9, the program and file assignments in the chromosomes of the population are shown in Fig. 4. The algorithm computes the DSR using SYREL and Outputs the task assignment. The reliability is 0.9994969. The reliability count is equal to 1.

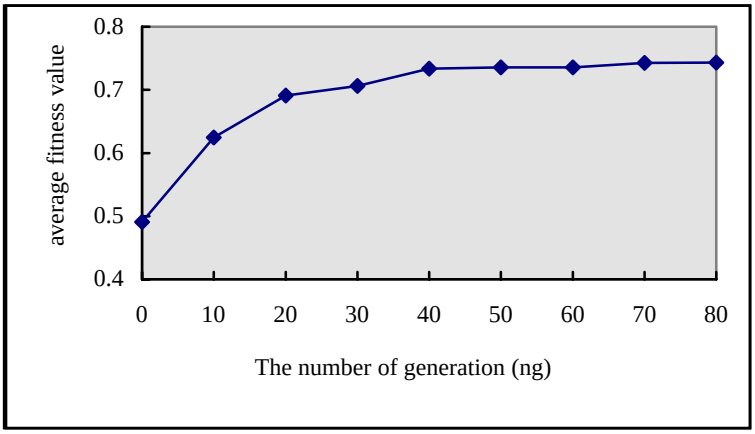


Fig. 3. The average fitness value of every generation of DS with six nodes and eight links with $P = 2$, $F = 3$, AFL(p_1)={f_1, f_2}, AFL(p_2)={ f_1, f_2, f_3}.

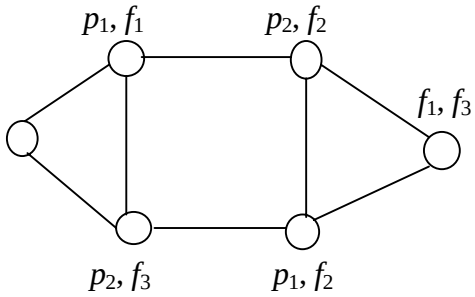


Fig. 4. The results of programs and data files assignment

4. RESULTS AND DISCUSSION

Table 1 presents the data on the results obtained using three different methods for various DS topologies with different allocated programs and data files.

Table 1. The results obtained using the exhaustive method, Hwang&Tseng method and our proposed method for various DS topologies and *k*-DTAs.

Size	*k*-DTA	AFL	Global	Exhaust	Hwang&Tseng	Proposed method

n	e	k	P	F	p_1	p_2	p_3	Optimal solution	NRC	time (sec)	time (sec)	absolute err	time (sec)	NRC	tng	ps	absolute err
6	8	1	2	3	f_1,f_2	f_1,f_3	-	0.9883041	4032	16	0.11	0.0099509	0.97	1	190	100	0
6	8	1	2	4	f_1,f_2,f_3	f_3,f_4	-	0.9883041	18756	145	0.11	0.0099509	0.72	1	90	100	0
6	8	1	3	3	f_1,f_2	f_2,f_3	f_1,f_3	0.9883041	13968	55	0.16	0.0207249	1.39	1	100	100	0
6	8	1	3	5	f_1,f_3,f_4	f_2,f_3,f_5	f_1,f_4	0.9745220	210168	1072	0.22	0.0055219	1.07	1	70	100	0
6	8	2	2	3	f_1,f_2,f_3	f_1,f_3	-	0.9998719	21312	97	0.16	0.0146482	0.77	1	190	100	0.0002047
6	8	2	2	3	f_1,f_2,f_3	f_1,f_2,f_3	-	0.9992175	21312	34	0.22	0.0137254	0.99	1	90	100	0.0001706
6	8	2	2	4	f_1,f_2,f_3	f_1,f_2,f_3,f_4	-	0.9984149	11691	19	0.38	0.0003736	0.72	1	100	100	0.0002877
6	8	2	2	3	f_2,f_3	f_1,f_2	-	0.9998719	21312	104	0.22	0.0002890	0.54	1	60	100	0.0002574
6	8	2	3	3	f_2,f_3	f_1,f_2	f_1,f_3	0.9998698	3699	24	0.27	0.0053454	0.38	1	100	100	0.0004042

NRC: the number of reliability computation Exhaust: the exhaustive method

k: the number of copies of programs and data files

if k=1, then $c(v_1)$=5, $c(v_2)$=4, $c(v_3)$=6, $c(v_4)$=6, $c(v_5)$=5, $c(v_6)$=5, $s(p_1)$=2,$s(p_2)$=3, $s(p_3)$=3, $s(f_1)$=2, $s(f_2)$=3, $s(f_3)$=3, $s(f_4)$=2, $s(f_5)$=2,

if k=2, then $c(v_1)$=6, $c(v_2)$=5, $c(v_3)$=7, $c(v_4)$=7, $c(v_5)$=6, $c(v_6)$=6, $s(p_1)$=2,$s(p_2)$=3, $s(p_3)$=3, $s(f_1)$=2, $s(f_2)$=3, $s(f_3)$=3, $s(f_4)$=2, $s(f_5)$=2,

In this paper, we proposed a new technique for solving the k-DTA reliability problem. The complexity of the proposed algorithm is $O(n^3+tng\times ps\times k\times n(P+F)^2+m^2)$. Results obtained from our algorithm were compared with those from the exhaustive method and the Hwang&Tseng method [13]. Although the exhaustive method, which the time complexity is $O(m^2n^{k(P+F)})$, can yield the optimal solution, it cannot effectively reduce the number of reliability computations and the time complexity. An application occasionally requires an efficient algorithm for computing reliability owing to resource considerations. Under this circumstance, deriving the optimal reliability may not be a promising option. Instead, an efficient algorithm yielding approximate reliability is preferred. The time complexity of the method of Hwang and Tseng [13] is $O(n^3+k\times n(P+F)+m^2)$, which is slightly quicker than our

method, but the deviation from exact solution is not ideal [13].

The accuracy and efficiency of the proposed algorithm were verified by implementing simulation programs in C language executed in Pentium III with 128M-DRAM on MS-Windows 98. In our simulation case, the number of reliability computations for the proposed algorithm was constant. The exact solution can be obtained when the number of copies of programs and files is one. In almost every case, if the number of copies of programs and files exceeds one, the proposed method can obtain an approximate solution, in which the average deviation from the exact solution is under 0.001. Because the proposed algorithm uses the elitist strategy at replacement and uses the access weight instead of the reliability of two-terminal for computing fitness value, in a few cases, it cannot obtain the exact solution.

5. CONCLUSION

This paper has presented a genetic algorithm based reliability-oriented task assignment methodology for computing the *k*-DTA reliability problem. Our numerical results show that the proposed algorithm may obtain the exact solution in most cases and the computation time seems to be significantly shorter than that needed for the exhaustive method. When the proposed method fails to give an exact solution, the deviation from the exact solution appears very small.

REFERENCES

[1] K.K. Aggarwal, S. Rai, "Reliability evaluation in computer communication networks", *IEEE trans. Reliability*, vol R-30, 1981 Jun, pp 32-35.

[2] K.K. Aggarwal, Y.C. Chopra, J.S. Bajwa, "Topological layout of links for optimizing the S-T reliability in a computer communication system", *Microelectron. Reliab.*, vol 22, num 3, 1982, pp 341-345.

[3] A. Satyanarayna, J.N. Hagstrom, "New algorithm for reliability analysis of multiterminal networks", *IEEE Trans. Reliability*, vol R-30, 1981 Oct, pp 325-333.

[4] S. Hariri, C.S. Raghavendra, “SYREL: A symbolic reliability algorithm based on path and cuset methods”, *IEEE Trans. Computers*, vol C-36, 1987 Oct, pp 1224-1232.

[5] F. Altiparmak, B. Dengiz, A.E. Smith, “Reliability optimization of computer communication networks using genetic algorithms”, Proc *IEEE int conf syst man cybern 5*, 1998 Oct, pp 4676-4680.

[6] D.W. Coit, A.E. Smith, “Reliability optimization of series-parallel systems using a genetic algorithm”, *IEEE Trans. Reliability*, vol R-45, 1996 Jun, pp 254-260.

[7] D. Torrieri, “Calculation of node-pair reliability in large networks with unreliable nodes”, *IEEE Trans. On Reliability*, vol R-43, 1994 Sep, pp 375-377.

[8] D.J. Chen, T.H. Huang, “Reliability analysis of distributed systems based on a fast reliability algorithm”, *IEEE Trans. on Parallel and Distributed Systems*, vol 3, 1992 mar, pp 139-154.

[9] V.K.P. Kumar, C.S. Raghavendra, Hariri, “Distributed program reliability analysis”, *IEEE Trans. Software Engineering*, vol SE-12, 1986 Jan, pp 42-50.

[10] A. Kumar, D.P. Agrawal, “A generalized algorithm for evaluation distributed program reliability”, *IEEE Trans. Reliability*, vol R-42, 1993 Sep, pp 416-426.

[11] D.J. Chen, R.S. Chen, T.H. Huang, “Heuristic approach to generating file spanning trees for reliability analysis of distributed computing systems”, *Journal of Computers Math. with Applic.* vol 34, 1997 Nov, pp 115-131.

[12] P. Tom, C.R. Murthy, “Algorithms for reliability-oriented module allocation in distributed computing systems”, *Journal of Systems and Software*, vol 40, 1998 Feb, pp 125-138.

[13] G.J. Hwang, S.S. Tseng, “A heuristic task assignment algorithm to maximize reliability of a distributed system”, *IEEE Trans. Reliability*, vol R-42, 1993 Sep, pp 408-415.

[14] M. Kafil, I. Ahmad, “Optimal task assignment in heterogeneous distributed computing systems”, *IEEE Concurrency*, vol 6, 1998 Sep, pp 42-51.

[15] A. Kumar, R.M. Pathak, Y.P. Gupta, “Genetic algorithm based approach for file allocation on distributed systems”, *Computers Ops Res.*, vol 22, num 1, 1995, pp 41-54.

[16] L. Davis, *et al*, “Genetic algorithms and simulated annealing: An overview”, *Genetic Algorithms and Simulated Annealing* (L. Davis, Ed), 1987, Morgan Kaufman.

[17] K.A. De Jong, “An analysis of the behavior of a class of genetic adaptive systems”, *PhD Thesis*, 1975, (microfilm -76-9381); Univ. of Michigan.

[18] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1989; Addison-Wesley.

[19] J.H. Holland, “Genetic algorithm and the optimal allocation of trials”, *SIAM J. Comput.*, vol 2, num 2, 1973 Jun, pp 88-105.

[20] G.E. Liepins, M.R. Hilliard, “Genetic algorithms: Foundations and applications”, *Annals of Operations Research*, vol 21, 1989, pp 31-58.