A Genetic Algorithm for Reliability-Oriented Task Assignment With $\widetilde{k}$ Duplications in Distributed Systems

Chin-Ching Chiu, Chung-Hsien Hsu, and Yi-Shiung Yeh

***Abstract*—A distributed system is a collection of processor-memory pairs connected by communication links. The reliability of a distributed system can be expressed using the distributed program reliability, and distributed system reliability analysis. The computing reliability of a distributed system is an NP-hard problem. The distribution of programs & data-files can affect the system reliability. 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. For example, efficient allocation of channels to the different cells can greatly improve the overall network throughput, in terms of the number of calls successfully supported. This paper presents a genetic algorithm-based reliability-oriented task assignment methodology (GAROTA) for computing the $\widetilde{k}$-DTA reliability problem. The proposed algorithm uses a genetic algorithm to select a program & file assignment set that is maximal, or nearly maximal, with respect to system reliability. 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 is very small. The technique presented in this paper would be helpful for readers to understand the correlation between task assignment reliability, and distributed system topology.**

***Index Terms*—Distributed program reliability, distributed system reliability, genetic algorithm, task assignment.**

Acronyms[1]

DPR	Distributed Program Reliability
DS	Distributed System
DSR	Distributed System Reliability
DTA	Distributed Task Assignment
FST	File Spanning Tree
GA	Genetic Algorithm
GAROTA	Genetic Algorithm-based Reliability-Oriented Task Assignment methodology
MFST	Minimal File Spanning Tree
NFT	Near-Feasibility Threshold

Manuscript received July 15, 2000; revised December 15, 2004 and August 31, 2005. Associate Editor: M. R. Lyu.

C.-C. Chiu and C.-H. Hsu are with the Department of Management Information System, Private Takming College, Taipei, Taiwan, R.O.C. (e-mail: chiu@takming.edu.tw).

Y.-S. Yeh is with the Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C. (e-mail: ysyeh@cs.nctu.edu.tw).

Digital Object Identifier 10.1109/TR.2005.863797

[1]The singular and plural of an acronym are always spelled the same.

Notation

$afl(p_\alpha)$	list of files required for p_α to complete its execution.
$a_{i,j}$	the probability of success of link $e_{i,j}$.
$avgft$	average fitness value, i.e., $avgft = sumft/ps$.
$b_{i,j}$	the probability of failure of link $e_{i,j}$.
$c(v_i)$	the capacity of the i^{th} node.
e	the number of links in G, $e = \|E\|$.
$e_{i,j}$	an edge represents a communication link between v_i and v_j.
F	total number of files in the DS.
ft_i	the actual fitness value of a chromosome i in a generation.
f_α	file α.
$G = (V, E)$	an undirected DS graph where V denotes a set of processing elements, and E represents a set of communication links.
k	the number of copies of pf_α.
mc, cc	mutation count, crossover count.
mr, cr	mutation rate, crossover rate.
n	the number of nodes in G, $n = \|V\|$.
P	total number of programs in the DS.
pf_α	distributed program or file α.
ps	population size.
p_α	distributed program α.
rft_i	a proportion of a roulette-wheel slot-sized of chromosome i in a generation, i.e., $rft_i = ft_i/sumft$.
$roulette_i$	accumulation of rft_i, i.e., $roulette_i = \sum_{k=1}^{i} rft_i$.
S_1	a set of $a_{i,j}$ of existing $e_{i,j}$.
S_2	a set of $a_{i,t}a_{t,j}$ of existing $e_{i,t}$, $e_{t,j}$.
S_3	a set of $a_{i,x}a_{x,y}a_{y,j}$ of existing $e_{i,x}$, $e_{x,y}$, $e_{y,j}$, in which v_x, $v_y \notin V_{s2}$ and v_x, v_y has not occurred. That is, the short path length from v_i to v_j, or from v_i to v_j is 3.
$s(f_\alpha)$	the size of data file f_α.
$s(p_\alpha)$	the size of program p_α.
$sumft$	total fitness value of all the chromosome in a generation, i.e., $sumft = \sum_{i=1}^{ps} ft_i$.
v_i	an i^{th} node represents the i^{th} processing element, $0 \le i < n-1$.
V_{s2}	a set of v_t of existing $e_{i,t}$, $e_{t,j}$.
Γ	a probability of a path created by selecting from the set S_1 if not empty, then from the set S_2 one after another, and same as the set S_3.

DEFINITIONS

Definition 1. **A distributed system** is defined as a system involving cooperation among several loosely coupled computers (processing elements). The system communicates (by links) over a network.

Definition 2. **A distributed program** is defined as a program of some distributed system which requires one or more files. For a successful distributed program, the local host possesses the program, the processing elements possess the required files, and the interconnecting links must be operational [9].

Definition 3. **A dependent set** is defined as a set S of distributed programs and files such that there does not exist a partition that divides S into two disjoint subsets S_x and S_y, where $S_x \cup S_y = S$, and $S_x \cap S_y = \varnothing$, such that each program and the files required are within the same subset [13].

Definition 4. **The 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 maximum [13].

Definition 5. **The *k*-DTA problem** is defined as determining assignments for k copies of a dependent set to maximize the DSR under some resource constraints in the distributed system [13].

Definition 6. **The $\widetilde{k}$-DTA problem** is defined as determining assignments for at least 1 copy and at most k copies of a dependent set to maximize the DSR under some resource constraints in the distributed system.

Definition 7. **The mask string** is defined as a string with a length $n \times (P+F)$ in which each bit indicates whether the capacity of a node is sufficient to be allocated a program or data file.

Definition 8. **The access weight** is defined as the probability that a program can access a data file to execute the program under consideration.

I. INTRODUCTION

DISTRIBUTED SYSTEMS (DS) have become increasingly popular in recent years. The advent of VLSI technology & low-cost microprocessors has made distributed computing economically practical. Distributed systems can provide appreciable advantages, including high performance, high reliability, resource sharing, and extensibility [1]. The potential reliability improvement of a distributed system is possible because of program and data-file redundancies. 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 & 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 & 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 & communication links. The second measure, distributed system reliability (DSR), is defined as the probability that all of the programs in the system can be run successfully. As Kumar, Hariri, & Raghavendra [9] pointed out, redundancy in resources such as computers, programs, and data-files can improve the reliability of distributed systems. Therefore, program & data-file assignment with redundancy considerations is important in improving a DSR.

It is assumed that there are n processing nodes, P programs, F data files, and k copies. The total number of possible assignments is then $n^{k(P+F)}$. Thus, the optimal program & file allocation for the processing nodes is an exponentially complex problem [15]. This fact indicates that optimum solutions can be found only for small problems. For larger problems, it is necessary to introduce heuristic algorithms that generate near-optimum solutions. The genetic algorithm (GA) can be adopted to search large, complex problem spaces [19]. The main steps for the GA are reproduction, selection, crossover, and mutation. The selection, crossover, and mutation processes are repeated until the termination condition is satisfied [16]–[20]. Typically, one can solve a constrained optimization using genetic algorithms, either by a penalty function for solutions outside the feasible, or by implementing special operators to ensure that all of the solutions are feasible. The penalty function used here employs the notion of a near-feasibility threshold (NFT) for each constraint [21]–[23]. The NFT is the threshold distance from the feasible region that is considered as being close to feasibility. It can be difficult to find a penalty function which is an effective, efficient surrogate for the missing constraints. We choose the latter course in this paper.

Hwang & Tseng [13] proposed the k-DTA (distributed task assignment with k copies) problem. The k-DTA models the assignment of k copies of both distributed programs, and their data-files to maximize the DSR within some resource constraints. We extend this problem to the $\widetilde{k}$-DTA problem which allows assignments for at least 1 copy, and at most k copies of a dependent set. Because the k-DTA problem is NP-hard [13], as is the $\widetilde{k}$-DTA problem, this study proposes an algorithm based upon the genetic algorithm [19] to find an approximate solution. The simulation results show that the exact solution can be obtained in most cases using the proposed algorithm. When the proposed algorithm fails to obtain an exact solution, the deviation from the exact solution is very small.

II. COMPUTING OPTIMAL RELIABILITY

In this section, we describe the problem addressed and clarify our research objectives.

Bi-directional communication channels operate between processing elements. A distributed system can be modeled using a simple undirected graph. For example, a DS topology with four nodes & five links is shown in Fig. 1. If two programs & three data files are allocated, the number of different program & data file combinations for allocation is $4^5 = 1024$. The program p_1

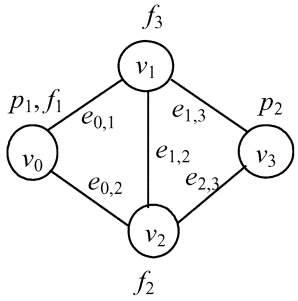

Fig. 1. The DS with four nodes, and five links.

requires data files f_1, and f_2; and p_2 requires data files f_1, f_2, and f_3 for completing execution. Assume that these files are allocated as shown in Fig. 1.

A file spanning tree (FST) consists 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. A minimal file spanning tree (MFST) contains no subset FST. Once all of the MFST 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 & vertices included in it are operational. Any terminal reliability evaluation algorithm based on path or cutset enumeration can be used to obtain the distributed program reliability $\mathrm{DPR}(p_\alpha)$ of the program p_α under consideration. For program p_1, there are three MFST, such as $v_0v_2e_{0,2}$; $v_0v_2e_{0,1}e_{1,2}$; and $v_0v_2e_{0,1}e_{1,3}e_{2,3}$. Therefore, $\mathrm{DPR}(p_1) = pr(\bigcup_{i=1}^{3} MFST_i)$. The reliability of program p_1 can be computed using a sum of mutually disjoint terms [4].

$$\begin{aligned}\mathrm{DPR}(p_1) = {} & b_{0,1}a_{0,2} + a_{0,1}a_{0,2}b_{1,2}b_{1,3} + a_{0,1}a_{0,2}b_{1,2}a_{1,3}b_{2,3} \\ & + a_{0,1}a_{1,2}a_{1,3}b_{2,3} + a_{0,1}a_{1,2}b_{1,3} + a_{0,1}a_{1,3}a_{2,3}.\end{aligned}$$

Assume that the probability of each link is 0.9. Then $\mathrm{DPR}(p_1) = 0.988\,29$.

In the same way, all the eight MFST of program p_2 are as follows.

$$\begin{aligned}& v_0v_1v_2v_3e_{0,1}e_{0,2}e_{1,3};\ v_0v_1v_2v_3e_{0,1}e_{1,2}e_{1,3};\\ & v_0v_1v_2v_3e_{0,1}e_{1,3}e_{2,3};\ v_0v_1v_2v_3e_{0,2}e_{1,2}e_{2,3};\\ & v_0v_1v_2v_3e_{0,2}e_{1,3}e_{2,3};\ v_0v_1v_2v_3e_{0,1}e_{0,2}e_{2,3};\\ & v_0v_1v_2v_3e_{0,2}e_{1,2}e_{2,3};\ v_0v_1v_2v_3e_{0,1}e_{1,2}e_{2,3}.\end{aligned}$$

$$\begin{aligned}\mathrm{DPR}(p_2) &= \mathrm{pr}\left(\bigcup_{i=1}^{8} MFST_i\right)\\ &= a_{0,1}a_{0,2}b_{1,2}a_{1,3}b_{2,3} + a_{0,1}b_{0,2}a_{1,2}a_{1,3}b_{2,3}\\ &\quad + a_{0,1}b_{0,2}b_{1,2}a_{1,3}a_{2,3} + b_{0,1}a_{0,2}a_{1,2}a_{1,3}b_{2,3}\\ &\quad + a_{0,1}a_{0,2}a_{1,2}a_{1,3}b_{2,3} + b_{0,1}a_{0,2}b_{1,2}a_{1,3}a_{2,3}\\ &\quad + a_{0,1}a_{0,2}b_{1,2}a_{2,3} + b_{0,1}a_{0,2}a_{1,2}a_{2,3}\\ &\quad + a_{0,1}a_{1,2}a_{2,3}.\\ &= 0.976\,86.\end{aligned}$$

$$\mathrm{DSR} = \mathrm{pr}\left(\bigcap_{\alpha=1}^{P} E(p_\alpha)\right) = \mathrm{pr}\left(\bigcap_{\alpha=1}^{P} MFST(p_\alpha)\right),$$

where $\mathrm{MFST}(p_\alpha)$ represents a set of MFST associated with program p_α. Therefore, the results of all the MFST of the intersection of the two programs are

$$\begin{aligned}& v_0v_1v_2v_3e_{0,1}e_{0,2}e_{1,3};\ v_0v_1v_2v_3e_{0,1}e_{1,2}e_{1,3};\\ & v_0v_1v_2v_3e_{0,1}e_{1,3}e_{2,3};\ v_0v_1v_2v_3e_{0,2}e_{1,2}e_{2,3};\\ & v_0v_1v_2v_3e_{0,2}e_{1,3}e_{2,3};\ v_0v_1v_2v_3e_{0,1}e_{0,2}e_{2,3};\\ & v_0v_1v_2v_3e_{0,1}e_{1,2}e_{2,3};\ v_0v_1v_2v_3e_{0,2}e_{1,2}e_{2,3}.\end{aligned}$$

$$\begin{aligned}\mathrm{DSR} &= \mathrm{pr}\left(\bigcap_{\alpha=1}^{P} E(p_\alpha)\right)\\ &= a_{0,1}a_{0,2}b_{1,2}a_{1,3}b_{2,3} + a_{0,1}b_{0,2}a_{1,2}a_{1,3}b_{2,3}\\ &\quad + a_{0,1}b_{0,2}b_{1,2}a_{1,3}a_{2,3} + a_{0,2}a_{1,2}a_{1,3}b_{2,3}\\ &\quad + b_{0,1}a_{0,2}b_{1,2}a_{1,3}a_{2,3} + a_{0,1}a_{0,2}b_{1,2}a_{2,3}\\ &\quad + a_{0,1}a_{0,2}b_{1,2}a_{2,3} + a_{0,1}a_{1,2}a_{2,3}.\\ &= 0.976\,86.\end{aligned}$$

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

1) Given:
   i) the topology of a DS
   ii) the reliability of each communication link
   iii) the available memory space of each processing element
   iv) a set of distributed programs
   v) a set of data files
   vi) the size of each distributed program
   vi) the size of each data File was copy-edited.
   vii) the files required by each distributed program for execution
2) Assumption:
   i) each node is perfectly reliable
   ii) each link is either in the working (ON) state, or failed (OFF) state
   iii) a graph G does not have any self-loops
   iv) the failure of a link is independent of the failure of other links
   v) each node on a DS can have only one copy of the data files
3) Constraint: the memory space limitation of each processing element
4) Goal: maximize the DSR of the system (or Maximize the DPR of a given program).

Reliability optimization can be defined as the maximum reliability for computing a given task under some constraints. For a given task, the reliability can be computed as $R_1, R_2, \ldots, R_x$ for x situations, where x may be an astronomical figure. By doing so, the reliability optimization for the task is the maximal reliability in $R_1, R_2, \ldots, R_x$. The genetic algorithm involves obtaining an approximate solution, which is close to the maximal reliability in $R_1, \mathrm{R}_2, \ldots, R_x$. Restated, a task assignment must be found under the given DS such that the DPR of a given program or DSR of the system is adequate. That is, when the algorithm fails to obtain an exact solution, the deviation from the exact solution is very small.

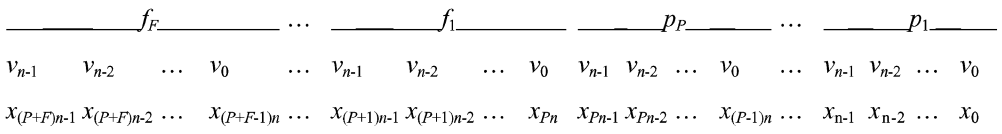

Fig. 2. The type of a chromosome, X, with a length $n \times (P + F)$.

The main problem can be mathematically stated as follows.

1) Given:
   i) distributed system parameters
   ii) memory capacity of each node
   iii) memory requirement of each program, and data file was copy-edited.
   iv) number of copies of each program, and data file
2) Objective: maximize $\mathrm{DSR} = \mathrm{pr}(\bigcap_{\alpha=1}^{P} E(p_\alpha))$
3) subject to:

$$\sum_{i=0}^{n-1}\left(\sum_{\alpha=1}^{P} s(p_j)x_{(\alpha-1)\times n+i} + \sum_{\alpha=P+1}^{P+F} s(f_j)x_{(\alpha-1)\times n+i}\right) \le c(v_i),$$

$$\sum_{i=0}^{n-1} x_{(\alpha-1)\times n+i} = \widetilde{k}$$

where $x_{(\alpha-1)\times n+i} = 0$ or 1, $\alpha = 1, \cdots, P+F$, and $\widetilde{k}$ is the number of copies of pf_α.

Obviously, the problem requires a large execution time. Herein, we develop an efficient method that allows task assignment optimization in the DS that achieves the desired performance. Owing to its computational advantages, our proposed method may be preferred to an exhaustive method.

III. 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 & their associated fitness values are called the population. This population at a given GA stage is referred to as a generation.

A. Development of the Approach

The development of our genetic algorithm-based reliability-oriented task assignment methodology (GAROTA) is described in the following subsections.

1) Chromosomal-Coding Scheme: Our problem involves program & data file assignments. The network topology is fixed. The size of every node is also fixed. We used a coding scheme with binary numbers. The length of a chromosome is equal to the number of network nodes multiplied by the sum of the number of programs & files. If P, F, and n represent the number of programs, data files, and nodes, respectively, then a chromosome, X, with a length $n \times (P + F)$ has the type as shown in Fig. 2.

Each bit indicates whether a program or file is allocated in a node, where $x_i = 1$, if it occurs; otherwise $x_i = 0$. That is, if program p_α is on node j, set $x_i = 1$ where the index $i = (\alpha - 1) \times n + j$ for $\alpha \le P$. For files, if file f_β is on node j, set $x_i = 1$ where the index $i = (\beta - 1 + P) \times n + j$ for $\beta \le F$.

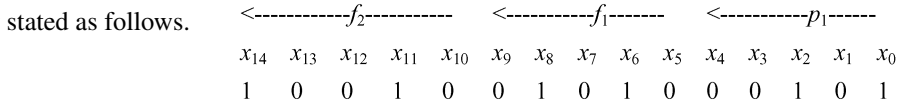

Fig. 3. A chromosome where $n = 5, P = 1, F = 2, \& \widetilde{k} = 2$.

For example, if $n = 5, P = 1, F = 2, \& \widetilde{k} = 2$ in Fig. 3 is one of the chromosomes, it indicates how the program & files are allocated. This chromosome shows that p_1 is allocated in nodes v_0 & v_2, f_1 is allocated in nodes v_1 & v_3, and f_2 is allocated in node v_1 & v_4.

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], the same characteristics as the $\widetilde{k}$-DTA problem.

For a valid chromosome, it must possess two characteristics:

1) For each sub-string, say $s_\alpha = x_{(\alpha+1)\times n-1}, \ldots, x_{\alpha\times n}$, of a chromosome, where α is in the range $(0, P+F-1)$, the number of bits whose value is 1 is under the $\widetilde{k}$ constraint.
2) The summation of the programs & files size, which is assigned to the same node, is at most as large as the capacity of the node.

The purpose of the CheckString function is to check that there are $\widetilde{k}$ copies of each program or file, and to ensure that the sum of the program & file memory used on each machine does not exceed its capacity. The detailed steps for CheckString are described in Fig. 4.

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.

a) Mask string generation: The size & format of a mask string, say M, is the same as those of a chromosome. This string can be used to avoid generating an invalid chromosome, and speed up the GA generation initialization. In our simulation case, if the mask string is omitted, approximately 80 to 100 invalid strings will be generated before obtaining a valid chromosome. For each bit of M, say $m_{\alpha\times n+i}$, the value is set at

$$m_{\alpha\times n+i} = \begin{cases} 1 & if\ c(v_i) \ge s(p_\alpha)\ and\ \alpha < P, \\ 1 & if\ c(v_i) \ge s(f_{\alpha-P})\ and\ \alpha \ge P, \\ 0 & otherwise \end{cases}$$

where $0 < \alpha < P + F$, $0 < i < n$.

The detailed steps for the InitMaskString function are described in Fig. 5.

b) Fast valid chromosome generation for population initialization: When a chromosome is generated according to the mask string M, we could always obtain a valid chromosome,

```
Function CheckString( X, P, F, k̃ , n ) /*check whether X is valid chromosome or not*/
    for (j=0; j < ( P + F ); j++ )
        s_j = x_{(j+1)×n-1},...,x_{j×n}.
        count = inspect the number of bits in sub-string s_j whose value is 1.
        if ( count <1 or count > k̃ )
            return false.
        end_if
    endfor
    for (i=0; i < n; i++)
        s_need(v_i) = 0. /*s_need(v_i) denotes the total capacity requires from v_i */
        for (j=0; j < ( P + F ); j++ )
            if ( x_{j×n+i} = 1)
                if ( j < P )
                    add s(p_j) to s_need(v_i).
                else
                    add s(f_{j-P}) to s_need(v_i).
                endif
            endif
        endfor
        if (s_need(v_i) > c(v_i))
            return false.
        endif
    endfor
    return true.
end CheckString
```

Fig. 4. The detailed steps of CheckString.

```
Function InitMaskString( P, F, n )
    length_M = n × ( P + F ). /*the number of bits in a mask string*/
    M = 0.    /*initial a string whose length is length_M bits*/
    for (j=0; j < ( P + F ); j++ )
        for (i=0; i < n; i++)
            if ( j < P, and c(v_i) > s(p_j) )
                Let m_{j×n+i} = 1.
            else if ( j ≥ P, and c(v_i) > s(f_{j-P}))
                Let m_{j×n+i} = 1.
            endif
        endif
        endfor
    endfor
    return M.
end GetMaskString
```

Fig. 5. The detailed steps of InitMaskString.

and omit checking whether it is a valid chromosome. The chromosome will satisfy our requirement, and can be appended to the population. The purpose of the GetString function is to generate a chromosome. The mask string would tend to put higher numbered files on higher numbered nodes in the initialization part of the GA (Function GetString). The detailed steps for GetString are described in Fig. 6.

4) Objective Function: To reduce the computational time, we construct the objective function carefully with a dependence on the weight of each node & each node pair, i.e. 2-terminal, for obtaining the fitness value of each chromosome.

a) Computing the weight of each node: The number of ports at each node (degree of a node), and the number of links would have a direct impact on the system reliability. Reliability decreases with a decrease in the number of links [5]. For any node, the degree of that node affects the number of the information paths that can be transferred from other nodes. The node with the higher degree is more likely to have more paths to the destination nodes than those with lower degrees. Therefore, we employed a simple means for computing the node weight, which takes less time, and quickly computes the process. The

```
Function GetString( M, P, F, k~ , n )
    Let c_tmp(v_i) = c(v_i), for 0 ≤ i < n.
    length_X = n × ( P + F ). /*the number of bits in a chromosome*/
    Let X = 0.      /*initialize a string X with a length length_X bits*/
    for (j=0; j < ( P + F ); j++ )
        Let t = 0.
        dowhile ( t <  k~ )
            i = generate a random number between 0, and n.
            if ( m_{j×n+i} = 1)
                Let m_{j×n+i} = 0.
                Let x_{j×n+i} = 1.
                if ( j < P )
                    Let c_tmp(v_i) = c_tmp(v_i) - s(p_j).
                else
                    Let c_tmp(v_i) = c_tmp(v_i) - s(f_{j-P}).
                endif
                Let a = j + 1.
                dowhile ( a < P + F ) /*tune the mask string M*/
                    if ( m_{a×n+i} = 1, and  x_{a×n+i} = 0)
                        if ( ( a < P and c_tmp(v_i) < s(f_{a-P}) ) or ( a ≥ P, and c_tmp(v_i) < s(f_{a-P}) ) )
                            Let m_{a×n+i} = 0.
                        endif
                    endif
                    Let a = a + 1.
                enddowhile
                Let t = t + 1.
            endif
        enddowhile
    endfor
    return X.
end GetString
```

Fig. 6. The detailed steps of GetString.

following formula is used to compute the weight, say $w(v_i)$, of node v_i.

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

where $d(v_i)$ denotes the number of links connected to the node v_i.

b) Computing the 2-terminal weight of all pairs: The reliability of a set of two nodes depends on their links, and the link reliability. In the network, two nodes may contain many paths between them. The length of a path is between 1, and $n-1$. To reduce the computational time, we considered a path in which the length is not greater than three. When the length of the path is three, paths were selected uniquely. The following formula with recursive description was used to evaluate the weight of 2-terminals.

$$\Phi(v_i, v_j) = \begin{cases} \Gamma \quad when\ initialize, \\ \Phi(v_i, v_j) + (1 - \Phi(v_i, v_j)) \times \Gamma \end{cases} \tag{2}$$

Because $0 \leq |S_1|, |S_2|, |S_3| < n$, in the worst case, the weight of 2-terminals can be computed in $n-2$ additions, $n-2$ subtractions, and $2n-1$ multiplications. Thus, in the worst case, when the graph is a complete graph, we can obtain all of the weights for each node-pair in $\mathrm{O}(n^3)$.

Function EvalFitnessValue(X)

/*compute the access weight of X using (5) and fitness value of X using (6)*/

$\Psi(X) = 0$.

firstflag=1.

for each program p_α do

for each $f_\beta \in afl(p_\alpha)$ do

if (there is a p_α copy in node v_x, and a f_β copy in node v_y)

if (pair (v_x, v_y) has not occurred, and pair (v_y, v_x) has not occurred)

if (*firstflag*=1)

$\Psi(X) = \Theta(v_x, v_y)$.

firstflag=0.

else

$\Psi(X) = \Psi(X) \times \Theta(v_x, v_y)$.

endif

endif

endif

enddo

enddo

return ($\Psi(X) \times \Psi(X)$).

end EvalFitnessValue

Fig. 7. The detailed steps of EvalFitnessValue.

c) Computing the 2-terminal access weight for all pairs: After the 2-terminal weight for all pairs & the weight of each node have been obtained, we can obtain the 2-terminal access weight of all pairs according to these values. The access weight can be obtained as

$$\Theta(v_i, v_j) = \frac{[\Phi(v_i, v_j) + w(v_i) + w(v_j)]}{3} \tag{3}$$

d) Computing the access weight of a program and a chromosome: If a program p_α is allocated in v_x, and a file f_β is allocated in v_y, the file f_β is needed when p_α is run. If the set $\Omega(p_\alpha)$ represents all of the pairs (v_x, v_y), i.e. $\Omega(p_\alpha) = \{(v_x, v_y) | v_x$ holds a copy of program p_α, v_y hold a copy of data file f_β, $f_\beta \in afl(p_\alpha), x \neq y\}$, we can use the following formula to compute the access weight of each program as follows.

$$\Psi(p_\alpha) = \prod \Theta(v_x, v_y) \tag{4}$$

where $(v_x, v_y) \in \Omega(p_\alpha)$.

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

$$\Psi(X) = \prod \Theta(v_x, v_y) \tag{5}$$

where $(v_x, v_y) \in \bigcup_{\alpha=1}^{P} \Omega(p_\alpha)$; and if both of pairs (v_x, v_y), (v_y, v_x) exist, discard (v_y, v_x).

e) Computing the fitness value of the chromosome using the objective function: According to the access weight of chromosome X, the objective function to compute the fitness value of X is constructed as

$$ft_i = (\psi(X))^2 \tag{6}$$

The objective function is 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 $\Psi(X)$ as the fitness value is for the expansion of the difference between two chromosomes. This will lead to an increase in the speed of population convergence. The purpose of the EvalFitnessValue function is to evaluate the fitness value of a chromosome. The detailed steps for EvalFitnessValue are described in Fig. 7.

5) Genetic Reproduction & 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.

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, and described in Fig. 8. Next, using a random number generator,

	<----------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

Fig. 8. The two chromosomes are randomly selected from the mating pool.

	<----------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

Fig. 9. The two new chromosomes with information from their parents.

Algorithm GAROTA

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

Read DS parameters: n, e, $a_{i,j}$, $c(v_i)$,

GA parameters: ps, cr, mr, ng, tng, /*tng denotes total no. of generation when GA end*/

task parameters: $\tilde{k}$, P, F, $s(p_\alpha)$, $s(f_\beta)$, $afl(p_\alpha)$.

step 1 /*compute each node weight using (1), all 2-terminal weight using (2) and all 2-terminal access weight using (3)*/

for (i=0; $i < n$; i++)

for (j=i+1; $j < n$; j++)

$\Theta(v_i, v_j)$ /*compute the pair (v_i, v_j) access weight using (3)*/

endfor

endfor

step 2 /*generate each chromosome of the initial population*/

2.1 /*initialize the mask string M for generating chromosomes quickly*/

M = **InitMaskString(** P, F, $\tilde{k}$, n **)**.

2.2 ng=0. /*generation 0, ng denotes generation of GA, ng=0,…,tng*/

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

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

$chrom_i$ = **GetString(** M, P, F, $\tilde{k}$, n **)**.

ft_i **= EvalFitnessValue(**$chrom_i$**)**. /*using (6)*/

endfor

step 3 /*generate roulette-wheel area*/

add each ft_i to $sumft$.

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

$roulette_i = ft_i / sumft$.

if (i != 0)

$roulette_i = roulette_{i-1} + roulette_i$.

endif

endfor

Fig. 10. The detailed steps of GAROTA.

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, if the crossover site is 2, the information exchange occurs as shown in Fig. 9.

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. Because p_1, f_1, and f_2 are usually allocated to v_1 in child1, the summation size is 7, which exceeds the capacity of v_1. This anomaly is just discarded.

```
step 4 /*reproduction/Selection for mating pool*/
    for (i=0; i < ps; i++)
        t = generate a random number between 0.0, and 1.0.
        j = 0.
        dowhile ( t > roulette_j )
            j = j + 1.
        enddowhile
        if ( j != 0 )
            j = j - 1.
        endif
        pool_chrom_i = chrom_j.
        pool_ft_i = ft_j.
    endfor
step 5 /*crossover for next generation*/
    cc = ⌈cr × ps / 2⌉. /*set the crossover count*/
    dowhile ( cc > 0 )
      generate random integer numbers x, y in the range (0, ps-1), and pos in the range (1,
          P+F-1).
      generate two chromosomes, say tmp1 & tmp2, by crossover the two chromosomes, say
      pool_chrom_x, and pool_chrom_y, which are selected at random from the mating pool.
      if (CheckString( tmp1, P, F, k̃ , n ) = True, and
          CheckString( tmp2, P, F, k̃ , n ) = True)
            tmp_ft1 = EvalFitnessValue( tmp1 ).   /* using (6) */
            tmp_ft2 = EvalFitnessValue( tmp2 ).   /* using (6) */
            replace pool_chrom_x, pool_chrom_y by the chromosomes which are related to the
              two maximized fitness values of { pool_ft_x, pool_ft_y, tmp_ft1, tmp_ft2 }.
            Let cc = cc - 1. /*decrement crossover count*/
      endif
    enddowhile
```

Fig. 10. *(Continued)* The detailed steps of GAROTA.

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 $\mathrm{y} \times \mathrm{n}$, 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 $(y \times n + z)^{\mathrm{th}}$ bit of the mutation chromosome, and mutates it. This mutation scheme ensures that the copy of each program & data file is correct by selecting a bit randomly in the range $(y \times n, y \times (n + 1) - 1)$ of the mutation chromosome.

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.

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. We randomly select a chromosome from the best two of the parents & the offspring. 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 & maximum fitness values of the strings in a generation become the same.

B. Complete Algorithm of GAROTA

The algorithm begins with an initial generation of valid chromosomes which satisfy the constraint. The initial generation

step 6 /* mutate for next generation*/

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

dowhile (*mc* > 0)

generate random integer number *x* in the range (0, *ps*-1), *j* in (0, *P*+*F*-1), and *t* in (0, *n*), respectively.

tmp_X = *pool_chrom*$_x$.

mutate the bit $x_{j\times n+t}$, and another bit by random select in $x_{j\times n}, \ldots, x_{j\times(n+1)-1}$.

if (**CheckString(** *tmp_X*, *P*, *F*, $\tilde{k}$, *n* **)** = True)

tmp_ft = **EvalFitnessValue(** *tmp_X* **)**. /*using (6)*/

if (*tmp_ft* > *pool_ft*$_x$)

pool_chrom$_x$ = *tmp_X*.

endif

Let *mc* = *mc* -1. /*decrement the mutation count*/

endif

enddowhile

step 7 /*replacement and creation a new generation*/

for (*i*=0; *i* < *ps*; *i*++)

chrom$_i$ = *pool_chrom*$_i$.

endfor

step 8 /*Test for terminating condition*/

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

if (*ng* ≤ *tng*, and some *ft*$_i$ ≠ *avgft*)

go to step 3.

endif

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

Fig. 10. *(Continued)* The detailed steps of GAROTA.

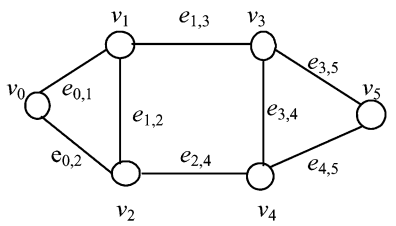

Fig. 11. The DS with six nodes, and eight links.

contains a finite number of valid strings selected at random. The number of strings in any generation, the population size, is kept to an even number to ease the crossover. The detailed steps for GAROTA are described in Fig. 10.

C. An Illustrative Example

The topology of the distributed system with six nodes & eight links is described as follows, and is shown in Fig. 11. The $c(v_i)$ represents the capacity of node v_i, and $a_{i,j}$ represents the reliability of link $e_{i,j}$.

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 $\{f_1, f_2\}$, and $afl(p_2)$ is $\{f_1, f_2, f_3\}$. Our task is to find the maximal distributed system reliability under the allocated programs & files.

In step 1, after evaluating each 2-terminal pair's weight using (3), the weight of $(v_0, v_1), (v_0, v_2), \ldots, (v_0, v_5)$, $(v_1, v_2), \ldots, (v_1, v_5), \ldots, (v_4, v_5)$ 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 the population.

In step 3, each chromosome's fitness value, ratio of fitness value, and roulette-wheel area are derived. The average fitness value is 0.490 881.

The algorithm executes statements between steps 4 & 8. They are reproduction & selection for the mating pool, crossover & mutation for the next generation, replacement & creation of the new generation, and testing for the termination condition. In

TABLE 1
THE RESULTS OBTAINED USING THE EXHAUSTIVE METHOD, HWANG & TSENG METHOD, AND OUR PROPOSED METHOD FOR VARIOUS DS TOPOLOGIES & $\tilde{k}$-DTA (WHERE $\tilde{k} = k$)

Size		$\tilde{k}$ -DTA			$afl(p_a)$			Global Optimal solution	Exhaustive Meth.[1]		Hwang&Tseng		Proposed method				
n	e	$\tilde{k}$[2]	P	F	p_1	p_2	p_3		NRC[3]	time (sec)	time (sec)	absolute err	time (sec)	NRC[4]	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	4	f_1,f_3,f_4	f_2,f_3	f_1,f_4	0.9745220	57624	266	0.22	0.0101783	1.48	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	2	3	f_2,f_3	f_1,f_3	-	0.9998719	21312	48	0.22	0.0002890	0.77	1	70	100	0.0007545
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
6	10[*4]	2	2	3	f_2,f_3	f_1,f_2	-	0.9999917	10260	113	0.22	0.0127541	1.14	1	60	100	0.0000811
6	10[*]	2	3	3	f_1,f_2	f_2,f_3	f_1,f_3	0.9999951	272157	3433	0.44	0.0274293	3.74	1	300	100	0.0000882
6	10[*]	2	3	4	f_1,f_2	f_1,f_2,f_3	f_1,f_3,f_4[+5]	1.0000000	5288498	38122	0.38	0.0291030	0.48	1	120	100	0.0000172

[1] Exhaustive Meth.: the exhaustive method

[2] $\tilde{k}$

k=1, set $c(v_0)$=5, $c(v_1)$=4, $c(v_2)$=6, $c(v_3)$=6, $c(v_4)$=5, $c(v_5)$=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.

k=2, set $c(v_0)$=6, $c(v_1)$=5, $c(v_2)$=7, $c(v_3)$=7, $c(v_4)$=6, $c(v_5)$=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.

[3] NRC: the number of reliability computation

[4] *:addition two links $e_{2,5}$, $e_{3,4}$ and $a_{2,5}$=0.9,$a_{3,4}$=0.95

[5] +: $s(f_4)$=1

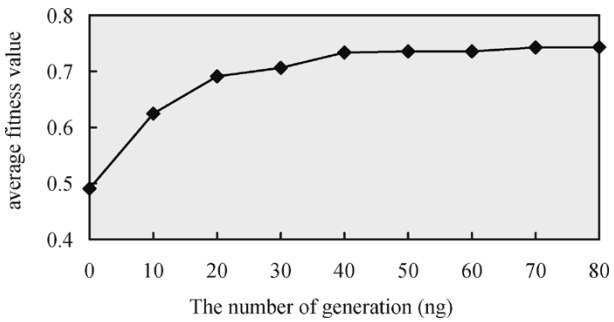

Fig. 12. The average fitness value of every generation of DS with six nodes, and eight links with $P = 2$, $F = 3$, $afl(p_1) = \{f_1, f_2\}$, and $afl(p_2) = \{f_1, f_2, f_3\}$.

addition, step 3 is executed again. The average fitness value of generation 1 is 0.537 424.

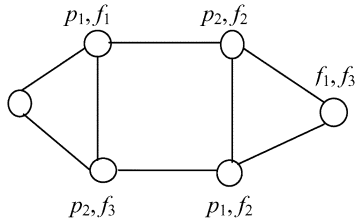

Fig. 13. The results of programs & data files assignment.

In the same way, the average fitness value of generation 10,. . ., 80 is obtained as shown in Fig. 12. In generation 80, all chromosomes are 10010001100010001000110001001O, and the fitness value is 0.748 307. The average fitness value is 0.748 307. Fig. 12 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 & file assignments in the chromosome population are shown in Fig. 13. The algorithm computes the DSR using SYREL [4], and outputs the task assignment. The reliability is 0.999 496 9. The number of reliability computations is equal to one, meaning that we only calculate the reliability of the optimal solution. Although we do have to compute the access weights & the fitness, the computational efficiency of our scheme is due to the fact that we are using access weights in the objective function as a proxy for reliability, so we do not have to explicitly compute the reliability of each solution.

IV. Results and Discussion

Table I presents the data on the results obtained using three different methods for various DS topologies with different allocated programs & data files. In contrast to the exhaustive method, the number of reliability computations grows rapidly when the size of the DS topology, or the number of programs & data files, is increased. The proposed method is constant, and independent of the size of the DS topology, and the number of programs & data files. The deviation is very small when the proposed method cannot obtain an optimal solution. These data show that the proposed method is more effective than the conventional methods.

In this paper, we proposed a new technique for solving the $\widetilde{k}$-DTA reliability problem. The complexity of the proposed algorithm in steps 0,. . ., 9 is $O(1)$, $O(n^3)$, $O(ps \times \widetilde{k}n(P+F)^2)$, $O(ps)$, $O(ps^2)$, $O(cc \times \widetilde{k}n(P+F))$, $O(mc \times \widetilde{k}n(P+F))$, $O(ps)$, $O(1)$, and $O(m^2)$, where $\widetilde{k}$ denotes the upper bound of the program & file copies, and m represents the number of paths in the assigned node set [4]. Therefore, the complexity of the proposed algorithm is $O(n^3 + tng \times ps \times \widetilde{k}n(P+F)^2 + m^2)$. Results obtained from our algorithm were compared with those from the exhaustive method, and the Hwang & Tseng's method [13]. Although the exhaustive method, which has a time complexity of $O(n^2 n^{\widetilde{k}(P+F)})$, 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 reliability computation will consume processor time. Thus, we focus on decreasing the number of reliability computations. When $\widetilde{k} = k$, the time complexity of the Hwang & Tseng method [13] is $O(n^3 + \widetilde{k}n(P+F) + m^2)$, which is slightly quicker than our method, but the deviation from the exact solution is not ideal [13].

In contrast to the computer reliability problem, which is static-oriented, the task assignment problems in the DS are dynamically-oriented because many factors such as DS topology, node capacity, link reliability, the size of the programs or files, files requested by each program, copies of programs & files, and the number of paths between each node can significantly affect the efficiency of the algorithm [4]. Thus, quantifying the time complexity exactly is extremely difficult. The accuracy & efficiency of the proposed algorithm were verified by implementing simulation programs in the C language executed on a Pentium III with 128 M-DRAM, using 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 & files is one. In almost every case, if the number of copies of programs & 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 two-terminal reliability of for computing the fitness value, in a few cases, it cannot obtain the exact solution.

References

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

[2] K. K. Aggarwal, Y. C. Chopra, and J. S. Bajwa, "Topological layout of links for optimizing the S-T reliability in a computer communication system," *Microelectron. Reliability*, vol. 22, no. 3, pp. 341–345, 1982.

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

[4] S. Hariri and C. S. Raghavendra, "SYREL: a symbolic reliability algorithm based on path and cuset methods," *IEEE Transactions on Computers*, vol. C-36, pp. 1224–1232, Oct 1987.

[5] F. Altiparmak, B. Dengiz, and A. E. Smith, "Reliability optimization of computer communication networks using genetic algorithms," in *Proceeding of theIEEE International Conference on System, Man, and Cybernetics 5*, Oct 1998, pp. 4676–4680.

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

[7] D. Torrieri, "Calculation of node-pair reliability in large networks with unreliable nodes," *IEEE Transactions on Reliability*, vol. 43, pp. 375–377, Sep 1994.

[8] D. J. Chen and T. H. Huang, "Reliability analysis of distributed systems based on a fast reliability algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, pp. 139–154, Mar. 1992.

[9] V. K. P. Kumar, C. S. Raghavendra, and C. S. Hariri, "Distributed program reliability analysis," *IEEE Transactions on Software Engineering*, vol. SE-12, pp. 42–50, Jan. 1986.

[10] A. Kumar and D. P. Agrawal, "A generalized algorithm for evaluation distributed program reliability," *IEEE Transactions on Reliability*, vol. 42, pp. 416–426, Sep. 1993.

[11] D. J. Chen, R. S. Chen, and T. H. Huang, "Heuristic approach to generating file spanning trees for reliability analysis of distributed computing systems," *Journal of Computers and Mathematics With Applications*, vol. 34, pp. 115–131, Nov. 1997.

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

[13] G. J. Hwang and S. S. Tseng, "A heuristic task assignment algorithm to maximize reliability of a distributed system," *IEEE Transactions on Reliability*, vol. 42, pp. 408–415, Sep. 1993.

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

[15] A. Kumar, R. M. Pathak, and Y. P. Gupta, "Genetic algorithm based approach for file allocation on distributed systems," *Journal of Computers and Operation. Research*, vol. 22, no. 1, pp. 41–54, 1995.

[16] L. Davis *et al.*, "Genetic algorithms and simulated annealing: an overview," in *Genetic Algorithms and Simulated Annealing*, L. Davis *et al.*, Ed: Morgan Kaufman, 1987.

[17] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," PhD Thesis, Univ. of Michigan, 1975.

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

[19] J. H. Holland, "Genetic algorithm and the optimal allocation of trials," *SIAM Journal on Computing.*, vol. 2, no. 2, pp. 88–105, Jun 1973.

[20] G. E. Liepins and M. R. Hilliard, "Genetic algorithms: foundations and applications," *Annals of Operations Research*, vol. 21, pp. 31–58, 1989.

[21] D. W. Coit and A. E. Smith, "Penalty guided genetic search for reliability design optimization," *Computers and Industrial Engineering*, vol. 30, no. 4, pp. 95–104, Jun 1996.

[22] ——, "Reliability optimization of series-parallel systems using a genetic algorithm," *IEEE Transactions on Reliability*, vol. 45, no. 2, pp. 254–260, Jun 1996.

[23] D. W. Coit, A. E. Smith, and D. M. Tate, "Adaptive penalty methods for genetic optimization of constrained combinatorial problems," *INFORMS Journal on Computing*, vol. 8, no. 2, pp. 173–182, Spring 1996.

Chin-Ching Chiu

Education: Sep. 1993–Oct. 2000 Ph.D. in Computer Science and Information Engineering, Department of EE & CS, National Chiao-Tung University. Sep. 1988–June 1991 M.S. in Computer Science, Department of EE & IE, Tamkang University. Chin-Ching Chiu received a BS degree in Computer Science from Soochow University.

Professional Background: Aug. 2001-Now Chairman, Department of Management Information System, Takming. Dec. 2000-Now Associate Professor, Department of Management Information System, Takming. Sep. 1991–Nov. 2000 Instructor, Department of Management Information System, Takming. April 1984–Aug. 1991 Computer Engineer in Data Communication Institute of Directorate General of Telecommunications. Oct. 1982–April 1984 System Analyst of SYSCOM computer company.

Research Interest: Reliability Analysis of Network & Distributed System, Genetic Algorithm, DNA computing, Ant algorithm.

Chung-Hsien Hsu

Education: Sep. 1993–June 1994 M.S. in Computer Science, Department of EE & IE, National Chun-Chung University. He received a BS degree in Mathematic Computer Science from National Chun-Shing University.

Professional Background: Sep. 1995-Now Instructor, Department of Management Information System, Takming. Aug. 1994–July 1995 System Analyst of SYSCOM computer company.

Research Interest: Reliability Analysis of Network and Distributed System, wireless application.

Yi-Shiung Yeh

Education: Sep. 1981–Dec 1985 Ph.D. in Computer Science, Department of EE & CS, University Of Wisconsin-Milwaukee. Sep. 1978–June 1980 M.S. in Computer Science, Department of EE & CS, University of Wisconsin-Milwaukee.

Professional Background: June 2002-Now Professor, Institute of CS & IE, National Chiao-Tung University. Aug. 1988–June 2002 Associate Professor, Institute of CS & IE, National Chiao-Tung University. Jul. 1986–Aug. 1988 Assistant Professor, Department of Computer & Information Science, Fordham University. Jul. 1984–Dec 1984 Doctorate Intern, Johnson Controls, Inc. Aug. 1980–Oct. 1981 System Programmer, System Support Div., Milwaukee County Gov.

Research Interest: Data security & Privacy, Information & Coding Theory, Game Theory, Reliability, and Performance.