Reliable Server Assignment in Networks using Nature Inspired Metaheuristics

Abdullah Konak[1], and Sadan Kulturel-Konak[2]

[1]Information Sciences and Technology, Penn State Berks, Reading, PA USA, konak@psu.edu

[2]Management Information Systems, Penn State Berks, Reading, PA USA, sadan@psu.edu

Cite as:

Konak, A. and Kulturel-Konak, S. (2011). "Reliable Server Assignment in Networks using Nature Inspired Metaheuristics," IEEE Transactions on Reliability, 60(2), 381-393.

Reliable Server Assignment in Networks using Nature Inspired Metaheuristics

Abstract

In this paper, a reliable server assignment problem in networks is defined as determining a deployment of identical servers to maximize a measure of service availability and solved using nature inspired metaheuristic approaches, namely Ant Colony Optimization, Particle Swarm Optimization and Clonal Selection Principle of Artificial Immune Systems. In networks, the communication between a client and a server might be interrupted because the server itself is offline or unreachable as a result of catastrophic network failures. Therefore, it is very important to deploy servers at critical network nodes so that the reliability of the system is maximized. A new reliability measure, called *critical service rate*, is defined to evaluate alternative server assignments with respect to the network's ability to provide services in the case of catastrophic component failures. The structure of the optimal server assignments is studied, and the performances of three nature inspired metaheuristics are investigated in a rigorous experimental study. Based on the computational studies, their advantages and disadvantages are discussed.

Keywords: *Ant Colony Optimization, Particle Swarm Optimization, Clonal Selection Algorithm, Reliable Network Design, Reliable Server Assignment*

ACRONYM AND ABBREVIATION

CSR	Critical Service Rate
DNS	Domain Name System
RSA	Reliable Server Assignment
MC	Monte Carlo
ACO	Ant Colony Optimization
CSA	Clonal Selection Algorithm
PSO	Particle Swarm Optimization
TSP	Traveling Salesperson Problem

TS	Tabu Search
GA	Genetic Algorithms
SL	Solution List
CL	Collision List
HT	Hash Table
EL	Elitist List
AIS	Artificial Immune Systems
MANOVA	Multivariate Analysis of Variance

NOMENCLATURE

$G=(V, E)$	an undirected network where $V=\{1,\ldots,n\}$ is the node set, $E=\{(i, j)\}$ is the edge set, and $(i, j)\equiv(j, i)$
$U(0,1)$	a Uniform number between 0 and 1
A	set of admissible nodes for server assignment
c_i	cost of deploying and maintaining a server at node i
$r_{(i, j)}$	reliability of edge $(i, j)\in E$
r_i	reliability of node $i\in V$
s_i	binary server assignment decision variable indicating whether a server is assigned to node i ($s_i=1$) or not ($s_i=0$)
s_{ji}	decision variable s_i of a solution j
n	number of nodes
$R()$	measure of the service availability
$\mathbf{X}$	a state vector of the network
$\delta_i(\mathbf{X}\|\mathbf{S})=$	1 if there exists at least one path between node i and a server node in state $\mathbf{X}$ for a given server assignment $\mathbf{S}$; 0, otherwise
$v_i(\mathbf{X})$	1 if node i is operational in state $\mathbf{X}$; 0, otherwise
α	critical service level, i.e., a predetermined fraction of the operational nodes
$\mathbf{S}$	a solution
$d(\mathbf{S})$	hash value of solution $\mathbf{S}$
ns	number of solutions searched so far
C	budget constraint
H	hash size
e_i	a prime number corresponding to decision variable s_i
K_1	a low number of simulation replications
K_2	a high number of simulation replications
K_3	a very high number of simulation replications
b	a predefined number of best solutions found so far
P	population
$\|A\|$	cardinality of set A

ACO Specific

$p_j(i)$	probability of assigning a server to node i for ant j
$\eta(i)$	problem specific heuristic information for node i
$\tau(i)$	pheromone value of node i
β	ACO parameter

PSO Specific

$\mathbf{S}_j=\{s_{j1},\ldots, s_{jn}\}$ current position of particle j in the solution space.
$\mathbf{V}_j=\{v_{j1},\ldots, v_{jn}\}$ velocity vector of particle j
$\mathbf{B}_j=\{b_{j1},\ldots, b_{jn}\}$ position where particle j has its best fitness so far
$\mathbf{G}=\{g_1,\ldots, g_n\}$ position of the global best fitness so far by any particle in the population
ω inertia coefficient
φ_1 and φ_2 cognition and social coefficients, respectively
$p_j(i)$ probability of assigning a server to node i

CSA Specific

μ number of cloned solutions
γ replacement ratio

I. Introduction

Telecommunications networks depend on several network services in order to provide users with the expected services. For example, a computer network will not function properly if the servers providing the Domain Name System (DNS) service to the network are inaccessible. Network users also use services that are provided by servers over the network in a distributed manner. Therefore, the availability and accessibility of the servers that provide critical services to the network as well as services to its users are critical. In this paper, a reliable server assignment (RSA) problem in networks is defined and solved using nature inspired metaheuristic approaches. The RSA problem has significant applications on telecommunications and logistics networks. Networks may become disconnected due to catastrophic component failures, and it is important for network users to access network services provided by the network servers even under such circumstances. Availability of network services can be increased by strategically allocating servers over a network. However, this problem has been addressed by a limited number of papers in the literature as summarized in Section II.

Consider an undirected network $G=(V, E)$, where $V=\{1,\ldots,n\}$ is the node set, $E=\{(i, j)\}$ is the edge set, and $(i, j)\equiv(j, i)$. The cost of deploying and maintaining a server at node i is given as c_i. For the successful operation of the network, each node must have access to at least one server. However, both nodes and edges of the network are subject to failure with known probabilities. Let $r_{(i, j)}$ be the reliability of edge $(i, j)\in E$ and r_i be the reliability of node $i\in V$. Edge and node failures are independent. When edge (i, j) fails, the communication between nodes i and j is interrupted if it cannot be rerouted through an alternative path. When a node fails, all of its incident edges become inoperative. Therefore, if a server node fails, the server located on this node becomes unreachable by all users. However, the edges of a failed node are not assumed to be in the failed state in terms of the state space. This assumption is important for computational tractability.

The RSA problem is defined as determining a deployment of servers to maximize a measure of service availability as follows:

$$\begin{aligned} &\max \quad z = R(\{s_1,\ldots,s_n\}, G) \\ &\sum_{i=1}^{n} c_i s_i \le C \\ &s_i \in \{0,1\} \end{aligned}$$

where s_i is the binary server assignment decision variable indicating whether a server is assigned to node i (s_i=1) or not (s_i=0), and $R()$ is a measure of the service availability.

A summary of the network reliability and availability measures defined in the literature is given in Section II. In this paper, we define a new reliability measure, called *critical service rate* (CSR), to evaluate alternative server assignments with respect to the network's ability to provide services in the case of catastrophic component failures. For a given server assignment

S=$\{s_1,\ldots,s_n\}$, CSR(**S**) is defined as the probability that more than a predetermined fraction (α) of the operational nodes have access to at least one server in case of a component failure as follows:

$$\mathrm{CSR}(\mathbf{S}) = \Pr\{\frac{\sum_{i\in V}\delta_i(\mathbf{X}\mid\mathbf{S})}{\sum_{i\in V} v_i(\mathbf{X})} \geq \alpha\} \tag{1}$$

where **X** denotes a state vector of the network; $\delta_i(\mathbf{X}|\mathbf{S})$=1 if there exists at least one path between node i and a server node, and $\delta_i(\mathbf{X}|\mathbf{S})$=0 otherwise; $v_i(\mathbf{X})$=1 if node i is operational in state **X**, $v_i(\mathbf{X})$=0 otherwise; and α is the critical service level. The CSR measure defined in (1) considers reachability of servers only by operational nodes because it is assumed that when a node fails, the users of that node cannot access network services. In practice, a network continues to operate even though several of its nodes fail or become disconnected, and CSR takes into account this operational aspect of networks. In this sense, CSR evaluates the ability of a network to continue providing network services in the case of catastrophic component failures.

By setting parameter α, CSR can model different reliability measures. For example, for α=1.0 and assuming perfectly reliable nodes, CSR is equal to network reachability [1], and if only a single server available, CSR is equal to the all-terminal reliability of the network. In Section IV-A, the structure of the optimal solution to the RSA are studied for different values of α.

II. Literature Survey

The RSA problem defined in this paper is closely related with the *p*-median problem. The deterministic *p*-median problem was originally defined by Hakimi [2] and is concerned with locating *p* identical services at *p* distinct nodes of a network to minimize the total weighted distance between the nodes and the closest servers. Several researchers ([3], [4], [5], [6], [7], and [8]) study the reliable *p*-median problem, which is concerned with the service unavailability due

to the infrastructure disruptions or component failures. In the reliable *p*-median problem, the nodes and/or edges of the underlying network are subject to failure with known probabilities. The overall objective is to maximize service availability, which is usually expressed in the form of an expected value of service level or a probability that a service is reachable. Therefore, the evaluation of the objective function requires techniques from network reliability analysis to compute the probability that a network maintains the desired connectivity. However, evaluating this type of a stochastic function is a daunting task because it requires considering all possible failure scenarios. In fact, most network reliability problems are *NP*-hard ([9]). Because of its difficulty, in the literature the reliable *p*-median problem has usually been studied either for special cases or with assumptions to make the evaluation of the objective function computationally feasible.

One of the earlier work related to the reliable *p*-median problem is given by Nel and Colbourn [4]. The authors formulate a problem to identify a single node on a network such that the expected number of nodes connected to that node is maximized in the presence of edge failures. Because computing this expected value is intractable, an efficient upper bound on the two-terminal reliability is utilized to identify promising nodes. Melachrinoudis and Helander [5] study a similar problem on tree networks with unreliable edges. Note that, unlike general networks as considered in this paper, on a tree network, it is computationally feasible to compute this objective function. Berman and Drezner [10] also study the stochastic one-median problem on general networks where the objective is to maximize the probability of reaching all nodes from a service center within a time threshold.

As mentioned earlier, Eiselt et al. [8] study the reliable *p*-median problem on general networks. However, the authors consider a special case of the problem where only a single edge

failure is considered at a time, and they propose an algorithm to transform a network into a tree network while preserving failure probabilities. Then, the problem is solved on the transformed tree network. This transformation is only possible because of the assumption that only a single node failure could occur at a time. Eiselt et al. [11] extend this approach to networks with unreliable nodes.

Berman et al. [12] define a reliable *p*-median on distribution networks to minimize the expected amount of unsatisfied demand. They assume that the probability of providing a satisfactory service to a node by another node is a monotonically decreasing function of the shortest distance between these two nodes. Nakaniwa et al. [13] consider the optimal mirror Web server assignment problem considering reliability. In this problem, edges are perfectly reliable and nodes are subject to failure. For a given server to node assignment, it is assumed that users are served by the closest Web server. Therefore, the probability that a user can access to a particular server is computed as a function of the failure rates of the nodes on the shortest path from the user to the server. An integer programming formulation is developed to maximize the overall system reliability under the cost, delay and capacity constraints. Snyder and Daskin [7] formulate a reliable *p*-median problem where nodes and edges of the network are perfectly reliable, but service facilities fail with known probabilities. A Lagrangian relaxation algorithm is used to determine a primary facility and a set of backup facilities for each customer. The primary facility provides customers with the service as long as it is operational, and the backup servers take turns when the primary facility or other backup facilities fail.

III. Methodology

The contributions of this paper to the body of work summarized in the previous section are three-fold: (*i*) the RSA problem is formulated on a general unreliable network by relaxing the

restrictive assumptions on the network topology and the number of servers; (*ii*) a new objective function is proposed to handle simultaneous edge/node failures; (*iii*) a novel simulation optimization approach is developed based on a Monte Carlo (MC) simulation and embedded into three nature inspired metaheuristics, namely Ant Colony Optimization (ACO), Clonal Selection Algorithm (CSA), and Particle Swarm Optimization (PSO). Biology and nature have been a continuous source of inspiration for scholars doing cutting edge research in optimization ([14]). The exact computation of the objective function is intractable for the size of problems studied in this paper. Therefore, MC simulation is used to evaluate the objective function of solutions created by the metaheuristic approaches. The metaheuristics, the MC simulation and hashing are integrated in a novel way to minimize the computational effort to efficiently evaluate candidate solutions and reduce the effect of the noise in the objective function evaluation due to simulation.

A. Ant Colony Optimization

ACO is developed by Dorigo et al. [15], [16] as a new approach to combinatorial optimization problems, particularly Traveling Salesperson Problem (TSP), based on the behavior of real ants. While searching for food, ants initially wander randomly. When a food source is found, ants lay down a chemical called pheromone on the way from the food source to their nest so that other ants are invited to follow this chemical trail instead of wandering randomly. ACO is a population based meta-heuristic approach. In ACO, the population mimics a real ant colony, and each artificial ant in the population represents a solution. Unlike other metaheuristics such as Simulated Annealing (SA), Tabu Search (TS), and Genetic Algorithms (GA) where new solutions are generated from existing ones using local and global search operators, ACO is a construction based approach. In each iteration of an ACO algorithm, new solutions (the entire

population) are randomly constructed from scratch. Surveys of recent research efforts and developments in ACO are given in [17], [18], and [19].

An ACO Algorithm for the RSA Problem

The ACO algorithm developed in this paper operates on the decision variables of the RSA problem defined in Section I. We define artificial pheromone trail $\tau(i)$ as the favorability of assigning a server to node i. In each iteration of the ACO, μ feasible solutions are constructed by randomly assigning servers to nodes, one node at a time. In other words, each ant constructs a solution by randomly selecting nodes to deploy servers based on the pheromone trails. While constructing a solution, each ant j calculates the probability of assigning a server to node i as follows:

$$p_j(i) = \frac{\tau(i)^{\beta}\eta(i)^{1-\beta}}{\sum_{k\in A}\tau(k)^{\beta}\eta(k)^{1-\beta}} \tag{2}$$

where A is the set of admissible nodes and $\eta(i)$ is the problem specific heuristic information for nodes. Node i is considered admissible for server assignment if the node is available (i.e., $s_{ji} = 0$) and there is an adequate budget to assign a server to the node. The ratio of the node reliability to the node cost (r_i/c_i) is used as the problem specific heuristic information after being normalized in [1, n] as follows:

$$\eta(i) = \frac{(r_i / c_i) - \min_k\{r_k / c_k\}}{\max_k\{r_k / c_k\} - \min_k\{r_k / c_k\}} \times (n-1) + 1 \tag{3}$$

Pheromone $\tau(i)$ values are also normalized between 1 and n at the end of each iteration. Therefore, only one parameter, $0 \le \beta \le 1$, is used to set the relative importance of the pheromone

trail information and the problem specific heuristic information. The procedure for the solution creation is given as follows:

Procedure Create_Solution (*j*) {

Set A=*V*, $C' \leftarrow C$, s_{ji} =0 for each node $i \in V$
While (A≠∅) {
Calculate $p_j(i)$ for each node $i \in$ A
Randomly select node *i* from A with probability $p_j(i)$
Set s_{ji} =1 and $C' \leftarrow C' - c_i$
Update set A
}
Return solution $\mathbf{S}_j$
}

Hashing to Check Solutions

In ACO as well as in other metaheuristics, as the search converges, same solutions will appear in the population with increasing frequencies. Therefore, computationally expensive simulation would be used to evaluate same solutions again and again in the case of the RSA problem. To prevent this, a hash table (HT) is used to rapidly detect whether a solution has been previously investigated or not. A HT maps data from a large domain to associated indexes (or hash values) from a much smaller domain. The hash function is used to generate hash values from data. HTs have been previously proposed in [20-21] to record the solutions encountered during recent iterations in TS. All solutions investigated during a search are stored in a list, called solution list (SL). A HT is used as a pointer to quickly access the solutions stored in SL. The hash value of a solution **S** is calculated as follows:

$$d(\mathbf{S}) = \mathrm{mod}(\prod_{i=1}^{n} (e_i)^{s_i}, H)$$

where H is the hash size and e_i is a prime number corresponding to decision variable s_i. Because the products of prime numbers are unique due to the Fundamental Theorem of Arithmetic, prime numbers are used in the calculation of hash value $d(\mathbf{S})$ to minimize probability of hash collisions, which is the case in which two different solutions have the same hash value.

HT is an integer array such that HT[$d(\mathbf{S})$]=0 if a solution with a hash value of $d(\mathbf{S})$ has not been searched yet, or HT[$d(\mathbf{S})$]=t if the first solution with a hash value of $d(\mathbf{S})$ is the t^{th} solution in solution list SL. In other words, SL[t] stores the t^{th} evaluated solution without any hash collision. After calculating $d(\mathbf{S})$ for a solution **S**, three cases are possible.

Case 1: If HT[$d(\mathbf{S})$]=0, then **S** has not been investigated before.

Case 2: If HT[$d(\mathbf{S})$]=t and **S**=SL[t], then **S** has been investigated before.

Case 3: If HT[$d(\mathbf{S})$]=t and **S**≠SL[t], then a hash collision occurs (i.e., two different solutions have the same hash value).

A second list, called collision list (CL), is used to store solutions with a hash collision. In this case, **S** is compared with all solutions in the CL. If **S** is not in the CL, then it has not been searched before and added to the CL. As it is demonstrated in the computational experiments, Case 3 occurs very rarely. In most cases, it is possible to check whether a solution has been previously searched or not in $O(1)$ comparisons after calculating $d(\mathbf{S})$. The procedure of checking a solution using hashing is given as follows:

Procedure Check_Solution (**S**) {

```
Calculate d(S)
If (HT[d(S)]=0) {
        t=t+1
        HT[d(S)]=t and SL[t]=S
        Return TRUE
}
```

```
        If (HT[d(S)]≠0 and SL[ HT[d(S)] ]= S) Return FALSE
        If (HT[d(S)]≠0 and SL[ HT[d(S)] ] ≠S) {
                If S∉CL {
                        cl=cl+1
                        CL[cl]=S
                        Return TRUE
                }
                        Else Return FALSE
        }
}
```

Solution Evaluation

As mentioned earlier, the objective function of the problem, CSR, is estimated using a crude MC simulation [22]. In the crude MC simulation, a state vector **X** is sampled by individually simulating the state of each edge $(i, j) \in E$ and node $i \in N$ using Bernoulli random variables with means r_{ij} and r_i, respectively. To estimate CSR, K state vectors are sampled in this way and the accessibility of servers by each operational node is checked for each state sampled. In turn, the ratio of the number of network states where critical service level α is achieved to sample size K yields an unbiased estimator of CSR.

Evaluating candidate solutions using simulation within an optimization algorithm has two important drawbacks. First, simulation output includes a statistical error which might affect the performance of the optimization algorithm. Second, simulation is computationally expensive, especially if a small margin of estimation error is required. To remedy these problems, a hierarchical approach to solution evaluation is used in the ACO as follows. First, a solution is evaluated using a low number of simulation replications (K_1), and if the solution seems to be promising after this first evaluation, then it is rigorously evaluated using a higher number of replications (K_2). The best b solutions found so far during the search are maintained in a sorted

list called *elitist list* (EL) such that CSR(EL[1])> CSR(EL[2])>…> CSR(EL[*b*]). After the first evaluation, a solution is identified as promising if the solution has a better estimated objective function value than the worst elitist solution. The statistical error due to simulation can be controlled by increasing the size of the EL. After rigorously evaluating a promising solution, the solution is compared with each elitist solution, and the EL is updated if necessary. After the search is terminated, all elitist solutions are evaluated one more time using a very high number of simulation replications (K_3). The proposed simulation approach in this paper has similarities with the drop-by-drop simulation approach given in [23-25].

Procedure Evaluate_Solution (**S**) {

```
    If Check_Solution(S)=TRUE {
        Evaluate S using K1 simulation replications
        If (CSR(S)> the worst CSR in EL) {
            Evaluate S using K2 replications
            Update EL by including S if necessary
        }
    }
}
```

Pheromone Trail Update and Overall Algorithm

Initially, $\tau(i)$ is set to the number of nodes, n, for each node i, and it is normalized over [1, n] after being updated and each elitist solution deposits pheromone proportionate to its rank in the EL at the end of each iteration. Therefore, $\tau(i)$ is updated as follows:

$$\tau(i) = \rho \times \tau(i) + \sum_{j=1..|\mathrm{EL}|} \frac{s_{ji}}{j} \tag{4}$$

where s_{ji} is decision variable s_i of the j^{th} elitist solution in the EL. After updating pheromone trails, they are normalized in [1, n] as follows:

$$\tau(i) = \frac{\tau(i) - \min_k\{\tau(k)\}}{\max_k\{\tau(k)\} - \min_k\{\tau(k)\}} \times (n-1) + 1 \quad (5)$$

Normalizing pheromone trails in [1, n] has two benefits. Firstly, η(i) and $\tau(i)$ are brought to the same numerical range, eliminating the need for one additional parameter to set the relative importance of η(i) and $\tau(i)$. Secondly, since $\tau(i)$ never becomes less than one, even unpromising nodes can be selected, although with a small probability, throughout the search. Therefore, a premature convergence of the algorithm might be avoided. The overall procedure of the ACO is given as follows:

Procedure ACO {
    Set SL=∅, CL=∅, EL=∅,
    Set t=0, cl=0, $\tau(i)=n$ for i=1,…,n
    While (stopping criterion is not satisfied) {
        For j=1...μ do {
            $\mathbf{S}_j$=Create_Solution(j)
            Evaluate_Solution($\mathbf{S}_j$)
        }
        Update $\tau(i)$ according to (4)
        Normalize $\tau(i)$ according to (5)
    }
    Evaluate each elitist solution using K_3 replications
    Return the best elitist solution
}

B. Particle Swarm Optimization

PSO was developed by Eberhart and Kennedy [26] based on the social behavior of species that live in the form of swarms in nature. These species are capable of exchanging valuable information such as food locations in the habitat by means of simple interactions among swarm members. As a result of these simple interactions among the members of a swarm, a global swarm behavior such as flocking may emerge. In PSO, the objective of each particle is to discover the optimal point in a continuous search space as it moves from one point to another

during the search. For a problem with n decision variables, the current position of particle j in the solution space is represented by a vector $\mathbf{S}_j=\{s_{j1},\ldots, s_{jn}\}$. In each iteration, each particle j moves to a new position in the solution space according to its velocity vector $\mathbf{V}_j=\{v_{j1},\ldots, v_{jn}\}$ as follows:

$$\mathbf{S}_j = \mathbf{S}_j + \mathbf{V}_j \tag{6}$$

As particles swarm over the solution space, they communicate with each other, broadcasting about the fitness of their best positions and learning about the best positions of others. Based on these interactions as well as their past experiences, particles adjust their velocities in each iteration. Let $\mathbf{B}_j=\{b_{j1},\ldots, b_{jn}\}$ represent the position where particle j has had its best fitness so far and $\mathbf{G}=\{g_1,\ldots, g_n\}$ is the position of the global best fitness so far by any particle in the population. The velocity vector of each particle j is updated as follows:

$$\mathbf{V}_j = \omega\mathbf{V}_j + \varphi_1\mathbf{U}_1 \cdot (\mathbf{B}_j - \mathbf{S}_j) + \varphi_2\mathbf{U}_2 \cdot (\mathbf{G} - \mathbf{S}_j) \tag{7}$$

In (7), ω is called the inertia coefficient that is used to balance exploration versus exploitation of the solution space. Values of ω close to 1 usually encourage exploration of new areas in the solution space, and for small values of ω such as $\omega < 0.4$, the search shifts to the exploitation mode. Parameters φ_1 and φ_2 are called cognition and social coefficients, respectively, and they are very important for facilitating convergence in PSO. Based on an empirical study [27], as well as theoretical results [28], it is recommended to set φ_1 and φ_2 such that $\varphi_1+\varphi_2 <4$. $\mathbf{U}_1$ and $\mathbf{U}_2$ are two vectors of n uniform random numbers between 0 and 1. Note that operator $(\cdot)$ in (7) indicates element-by-element vector multiplication.

As briefly described above, PSO is originally designed for the problems with real-valued decisions variables. Kennedy and Eberhart [29] introduce the binary PSO for problems with

binary decision variables. In the binary PSO, the values of decision variables are randomly determined using particle velocities and a logistic function as follows:

$$s_{ji} = \begin{cases} 1 & \text{if } U(0,1) < (1+\exp(-v_{ji}))^{-1}, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

where $U(0,1)$ is a Uniform number between 0 and 1. In (8), the logistic function is used to calculate the probability that the value of a binary decision is one. The rest of the algorithm is the same with the real-value PSO algorithm.

A PSO Algorithm for the RSA Problem

The PSO algorithm proposed in this paper depends on the binary PSO introduced in [29]. However, the RSA problem is a constrained problem and using (8) to determine the values of decision variables would result in many infeasible solutions. To address this problem, Procedure Create_Solution(), which is the solution construction heuristic of the ACO, is used to construct feasible particle positions with the following probabilities:

$$p_j(i) = \frac{(1+\exp(-v_{ji}))^{-1}}{\sum_{k \in A} (1+\exp(-v_{jk}))^{-1}}. \tag{9}$$

Similar to the ACO, in (9), A is the set of admissible nodes for server assignment and $p_j(i)$ represents the probability of assigning a server to node i while determining the new position of particle j based on $\mathbf{V}_j$ (i.e., $p_j(i)=\Pr\{s_{ji}=1\}$). Notice that in (9), the relative weights of the probabilities in (8) are preserved, and the procedure Create_Solution() randomly selects nodes for server assignment based on their selection probabilities.

The hierarchical solution evaluation approach (procedure Evaluate_Solution()) with hashing, which is described in the ACO algorithm, is also used within the PSO algorithm. The

best elitist solution (EL[1]) is used as the global best position so far **(G)**. The procedure of the PSO algorithm is given below.

Procedure PSO {

Set SL=∅, CL=∅, EL=∅
Set t=0, cl=0
For j=1...μ do {
Set v_{ji}=0 for i=1,…, n
$\mathbf{S}_j$=Create_Solution(j)
Evaluate_Solution($\mathbf{S}_j$)
Set $\mathbf{B}_j$=$\mathbf{S}_j$
}
Set **G**=EL[1]
While (stopping criterion is not satisfied) {
For j=1...μ do {
Set v_{ji}= v_{ji}+$U(0,\varphi_1)\times(b_{ji}-s_{ji})$+$U(0,\varphi_2)\times(g_i-s_{ji})$ for i=1,…, n
$\mathbf{S}_j$=Create_Solution(j)
Evaluate_Solution($\mathbf{S}_j$)
If (CSR($\mathbf{S}_j$)> $\mathbf{B}_j$) set $\mathbf{B}_j$=$\mathbf{S}_j$
}
Set **G**=EL[1]
}
Evaluate each elitist solution using K_3 replications
Return the best elitist solution
}

C. Artificial Immune Systems and Clonal Selection Algorithm

Artificial Immune Systems (AIS) are relatively new bio-inspired computational algorithms. The early theoretical immunology work [30-32] has prepared the origins for AIS. There has been an increasing interest in applying AIS in many different areas such as scheduling, image processing, bio informatics etc. after its applications on machine learning [33-35].

One of the population based AIS algorithms, namely CSA, is applied to solve the RSA problem in this paper. CSA is developed based on the foundational clonal selection theory of Burnet [36]. The basic immunological components are: maintaining a specific memory set, selecting and cloning most stimulated antibodies, removing poorly stimulated or non-stimulated antibodies, hypermutating (i.e., affinity maturation) activated immune cells and generating and maintaining a diverse set of antibodies [33]. The principle of the theory is mainly as follows: The antigen (i.e., the foreign molecule that the immune system is defending against) selects the lymphocytes (i.e., B-cells or white blood cells that detect and stop antigens) with receptors which are capable of reacting with a part of the antigen. The rapid proliferation of the selected cell occurs during the selection to combat the invasion (i.e., clonal expansion and production of antibodies). While duplicating the cells, coping errors occur (i.e., somatic hypermutation) resulting in an improved affinity of the progeny cells receptors for the triggering antigen. CSA has been applied to many engineering and optimization problems. A basic CSA has been proposed by de Castro and von Zuben [37] and later renamed as CLONALG by de Castro and von Zuben [38], and they also provide the extensive analysis on its algorithmic computational complexity. In the general CLONALG, first, antibodies (i.e., candidate solutions) are selected based on affinity either by matching against an antigen pattern or via evaluation of a pattern by an objective function (affinity means objective function value in optimization problems). Then,

selected antibodies are cloned proportional to their affinity, and each clone is subject to a hypermutation inversely proportional to its affinity. Finally, the resultant clonal-set competes with the antibody population to survive for membership in the next generation, and low-affinity population members are replaced with randomly generated antibodies [39]. The main steps of CLONALG are as follows:

```
Initialization
While (stopping criterion is not satisfied) {
        Antigen Selection
        Exposure and Affinity Evaluation
        Clonal Selection
        Affinity Maturation (Hypermutation)
        Metadynamics (Replacement)
}
Termination
```

A CSA for the RSA Problem

In this section, a CSA is proposed to the RSA problem. We are adopting CLONALG, and hereafter, the algorithm used in this paper will be called as the CSA. In the CSA, each solution is cloned and mutated according to its rank in the population such that a higher-ranked solution produces a higher number of clones than a lower-ranked solution, and it is mutated at a lesser degree. To achieve this, solutions in the population are sorted and ranked into three equal sized tiers as top 1/3, middle 1/3, and bottom 1/3 based on their objective functions. Each solution at the top, middle, and bottom tiers respectively produces $\mu/2$, $\mu/3$, and $\mu/4$ clones which are mutated by bitwise invert operators based on its tier. This ensures that solutions with better objective function values will produce a higher number of clones. On the other hand, each clone is mutated at a rate inversely proportional to its tier. Therefore, three different invert mutation operators are applied to the clones of the three tiers as follows. Top tier clones are perturbed by

inverting one or two decision variables (Mutation1), middle tier clones by three decision variables (Mutation2), and bottom tier clones by four decision variables (Mutation3). In other words, new solutions created from top tier solutions are similar to their parents and solutions created from bottom tier solutions are much more different than their parents. Top tier solutions are promising; therefore, a local search around these solutions is justified. On the other hand, bottom tier solutions are probably in non-promising regions of the search space. Therefore, moving away from those regions by large perturbations is desired. The mutation operators of the CSA are as follows:

Procedure Mutation1 (**S**$_j$) {
    Randomly select two nodes i_1 and i_2 such that $s_{ji1}+s_{ji2} \leq 1$
    If $s_{ji1}+s_{ji2}=1$, then flip the values of s_{ji1} and s_{ji2} to obtain new solution **S**
    If $s_{ji1}+s_{ji2}=0$, then flip the value of s_{ji1} to obtain new solution **S**
    Return **S**
}

Procedure Mutation2 (**S**$_j$) {
    Randomly select three nodes i_1, i_2, and i_3 such that $1 \leq s_{ji1}+s_{ji2}+s_{ji3} \leq 2$
    Flip the values of s_{ji1}, s_{ji2}, and s_{ji3} to obtain new solution **S**
    Return **S**
}

Procedure Mutation3 (**S**$_j$) {
    Randomly select four nodes i_1,i_2, i_3, and i_4 such that $s_{ji1}+s_{ji2}+s_{ji3}+s_{ji4}=2$
    Flip the values of s_{ji1}, s_{ji2}, s_{ji3}, s_{ji4} to obtain new solution **S**
    Return **S**
}

The replacement strategy of the CSA is to replace the worst γ% of the population with randomly generated new solutions, and it is applied once in every five iterations. The procedure of the CSA is given as follows:

Procedure CSA {

    Randomly generate μ solutions

```
    While (stopping criterion is not satisfied) {
        Sort and rank the population
        If (replacement=TRUE) {
            Replace worst γ% of the population with randomly generated solutions
        }
        For each top tier solution S_j do {
            Generate μ/2 mutated clones of S_j using Mutation1(S_j)
            Use Evaluate_Solution() procedure to evaluate clones
            Replace S_j if a better new solution is found
        }
        For each middle tier solution S_j do {
            Generate μ/3 mutated clones of S_j using Mutation2(S_j)
            Use Evaluate_Solution() procedure to evaluate clones
            Replace S_j if a better new solution is found
        }
        For each bottom tier solution S_j do {
            Generate μ/4 mutated clones of S_j using Mutation3(S_j)
            Use Evaluate_Solution() procedure to evaluate clones
            Replace S_j if a better new solution is found
    }
}
```

IV. Computational Study and Discussions

A. Study of CSR and the RSA problem

The first set of computational experiments focuses on understanding the structure of good solutions to the RSA problem based on the CSR measure in this paper. For this purpose, a test problem with 11 nodes and 14 edges is used as shown in Figure 1. This network features nodes with various node degrees and connectivity. The problem is simplified as to determine k nodes of the network to assign k servers in order to maximize CSR for various values of k and α. To make the solution to the problem independent from the cost and reliability data, they are set as follows: c_i=1 and r_i=0.95 for each node i, and $r_{(i,\ j)}$=0.90 for each edge (i, j). The optimal solution was found by enumerating all possible solutions and exactly evaluating CSR using a backtracking algorithm. The problem was solved for different problem input parameters such as C={1, 2, 3} and α={0.50, 0.60, 0.70, 0.80, 0.90, 1}. In addition, two versions of the problem,

one with reliable nodes (r_i=1) and the other with unreliable nodes (r_i =0.95), were considered to understand the effect of node failures on the optimal solution. The optimal solutions are given in Table 1.

It is clear that the optimal solution to the problem significantly depends on the critical service level α. For example, when only one server is available (i.e., C=1) and all nodes are assumed to be perfectly reliable, the best node to locate the server changes depending on α as follows: node 6 for α=0.50,…, 0.80, either of nodes 2, 5, or 7 for α=0.90, and either of the nodes for α=1.0. From a practical point of view, it makes sense to assign the single server to a highly-connected, central node in a network, which is node 6 in this case (assuming all nodes are perfectly reliable). Node 6 is also the optimal solution for the deterministic p-median problem defined by Hakimi [2]. However, as α approaches 1.0, the optimal assignment policy seems shifting towards assigning the single server to a sparsely connected, perimeter node of a network from a highly connected, central node of the network. This pattern is also observed with multiple servers. For example, if three servers are available (i.e., C=3), the optimal solution includes perimeter nodes 1, 8, and 11 for α =1.0. Although this result is unexpected, it can be intuitively explained as follows. As α increases, CSR requires that more operative nodes must be connected to a server in order to satisfy the service requirement (e.g., all operative nodes must be connected to at least one server for α=1.0). Assume that only edge (1, 2) fails in a network state and consider the following two assignments for C=1 and α=1.0: (a) the server is located at perimeter node 1 and (b) the server is located at central node 6. In case (a), the other nodes cannot access the server. On the other hand, in case (b), node 1 cannot access the server. In either case, at least one node does not have access to the server, and the targeted service level is

not achieved. In other words, these two assignments are not different in terms of providing the expected high service level. In fact as seen in Table 1, for C=1 and α=1 with perfectly reliable nodes, all solutions to the problem will have the same level of CSR, which is equal to the all-terminal reliability of the network. Because the central nodes are strongly connected to one another, if one of them has access to a server, the others will have access to that server with a high probability as well. Therefore, servers are assigned to sparsely connected perimeter nodes in the optimal solutions given in Table 1 for high values of α and perfectly reliable nodes. The similar pattern is also observed in the cases with unreliable nodes. Servers tend to be assigned to perimeter nodes as α increases. However, the results for the cases with perfectly reliable and unreliable nodes can be very different. For example, for C=1, α=0.90, and assuming perfectly reliable nodes, assigning the server to either of nodes 2, 5, or 7 yields the same level of CSR. However, when node failures are considered, the optimal solution is assigning the server to node 2. Again, not the central node 6, but the perimeter node 2 is the optimal assignment in this case. For α=0.90, all network states in which two or more nodes do not have access to a server are considered as failed states. Note that node 6 is three hops away from nodes 1, 8, and 11, which are the weakest connected nodes in the network, and node 2 is also three hops away from nodes 8 and 11, but only one hop away from node 1. Therefore, node 2 is in fact more central than node 6 with respect to the weakly connected nodes of the network. Because these weakly connected nodes become significantly critical as α approaches 1.0, the server is assigned to node 2. In summary, if network designers are interested in the median performance of a network, an α value close to 0.5 should be selected. If servicing each node in the network is mission critical as in the all-terminal reliability measure, α should be set close to 1.

------- Insert Figure 1 Here -----

------- Insert Table 1 Here -----

B. Comparisons of Heuristics

First, the ACO, PSO, and CSA metaheuristics in this paper are compared in a rigorous experimental study to investigate their performances under different scenarios, including various sizes of problem, search space and stopping criterion. The solutions found by the heuristics are compared using a Multivariate Analysis of Variance (MANOVA) to test whether there are significant differences among their performances under different scenarios. The experimental design and the results of the MANOVA are provided in Table 2. In all runs, the simulation related parameters are $K_1=10^3$, $K_2=8\text{x}10^3$, $K_3=10^5$, $H=99001$, and $|EL|=20$, and all cases are solved for critical service level of $\alpha=0.95$. The algorithm specific parameters of the three metaheuristics were determined after initial runs. The ACO parameters, β and ρ, are randomly selected from uniform distributions [0.25, 0.75] and [0.93, 0.97], respectively, in each iteration of the ACO. The performance of the ACO has been very robust in these ranges of the parameters in initial experiments. The PSO parameters φ_1 and φ_2 are 2 as generally recommended in the PSO literature. In the CSA, the worst 20% (γ%) of the population are replaced with randomly generated solutions once in every five iterations based on the results in initial experimentation. For all heuristics, the population size is 50, and the number of solutions searched so far (*ns*) is used as the stopping criterion. To investigate the effect of problem size, problems ranging from 30 to 100 nodes are used. For each problem, ten random instances are created by randomly assigning node and edge reliabilities from [0.90, 0.95] and edge costs from [1, 2]. Each random problem instance is solved for two levels of budget constraints (C=5 and

C=8) and two levels of stopping criteria (*ns*=1000 and *ns*=8000) using ten random replications. Therefore, each scenario in Table 1 includes 300 runs (ten random problem instances, ten replications for each instance, and the three heuristics). Using multiple random test problems, multiple replications and various problem input parameters, it is aimed to minimize the possible effect of metaheuristic specific parameter settings. A MANOVA model is used to test the hypothesis whether the performances of the heuristics are different or not for each scenario. The MANOVA models include the problem instances and the heuristics as the fixed factors and the replications as a random factor. In Table 2, the *p*-value and R^2 of the models are provided, and the three heuristics are sorted based on their average performance in each cell. A *p*-value that is less than 0.05 indicates that the results of the heuristics are statistically different with 95% confidence (these cells are set as bold in the table). In addition, a high R^2 indicates a small variation within ten replications. Analyzing the results in Table 2, the following conclusions can be drawn.

- The ACO performs better than the PSO and the CSA in the earlier stages of the search by quickly discovering good solutions. When the search is stopped after the first 1000 solutions discovered, the solutions found by the ACO are statistically better than the solutions found by the other two heuristics in the majority of the cases for both *C*=5 and *C*=8 (with the *p*-values of almost zero). Superior performance of the ACO in the initial stages of the search can be explained by its use of the problem specific heuristic information. Note that the ACO uses the heuristic information about the ratio of the node reliability to the cost in the solution construction procedure while the PSO operates with randomly initialized particle velocities.

- For C=5 and ns=8000, there is no statistically significant difference among the three heuristics. Note that for C=5 and ns =1000, the ACO provides the best results. Based on this observation, therefore, it can be concluded that the ACO converges faster than the PSO and the CSA, but they catch up with the ACO as the search progresses for C=5.
- For C=8 and ns=8000, there is no statistically significant difference among the three heuristics in the problem groups of n= 30, 40, and 50. However, the PSO yields the best results for the problem groups of n= 60, 70, 80, and 100 (the differences are statistically significant). Based on this observation and the fact that increasing the budget constraint from C=5 to C=8 exponentially increases the number of feasible solutions in each problem instance, it can be suggested that the PSO is more effective in exploring the solution space, particularly in large sized problems.

------- Insert Table 2 Here -----

To further understand the results presented in Table 2, the best elitist solutions of the three heuristics are compared at different stages during the search. In Figure 2, the 95% confidence intervals and the average CSR of the best elitist solutions found so far in ten random replications of a problem instance of two cases (n=70 with C=5 and n=100 with C=8, respectively) are plotted against the number of the solutions searched so far (ns). Figure 2 represents a case where the PSO and the CSA perform better than the ACO. The convergence pattern observed in this figure is also representative for the other cases. The ACO clearly dominates the PSO in the beginning of the search. The improvement of the ACO is very rapid in the first a few hundred solutions searched. As the search progresses, first the CSA and then the PSO catch up with the ACO and outperform it for the problems with n=100 and C=8. The fast convergence of the ACO may become a drawback in large size problems because the search may stagnate around a few

good solutions discovered early in the search. In the literature, randomly resetting pheromone trails is recommended to remedy this drawback of ACO [19]. In this paper, the main difference between the ACO and the PSO is in the calculation of the probabilities of assigning servers to nodes during the solution construction process. In the ACO, all ants use the same pheromone trails to calculate those probabilities while in the PSO, each particle maintains its individual velocity vector which may aim a different direction in the search space. This causes a slow convergence of the PSO, but also reduces the likelihood of being trapped in local optima and encourages exploration of different regions in the search space. In the literature, a few papers [40-42] that study comparative performances of ACO and PSO also report the similar findings with ours (i.e., ACO converges faster, but PSO performs better in longer runs).

---- Insert Figure 2 Here -----

Tables 3 and 4 present computational results (averaged over ten random problems and ten random replications) for *ns*=8000 with *C*=5 and *C*=8, respectively. In the tables, (Elitist Range)/σ is the ratio of the CSR range of the elitist list (i.e., the difference between the best CSR and the worst CSR) to the estimation of the standard deviation in simulation. (Elitist Range)/σ is an indicator for the quality of the best solution with respect to the elitist list. In each case, the gap between the best and worst elitist solutions is much larger than the estimated standard deviation. Therefore, it can be assured with a high confidence level that the final elitist list will include the true best solutions found during the search. Collision% is the ratio of |CL| to |CL|+|SL| at the termination (i.e., percent of hash table collision). A small value of Collision% indicates how efficiently hashing can detect whether a solution has been previously investigated or not. This ratio is very low for all cases. For example, 4% collision rate means that the solution check can be performed in $O(1)$ time (i.e., checking the HT only after calculating the

hash value) in 96% of the search. Finally, average CPU seconds (with the 3.0 GHz Intel Xeon E5450 Quad-Core Processors and 32 GB memory) are given for all three heuristics. Clearly, there is not any significance difference among the CPU times of the three heuristics because the simulation is computationally the most expensive operation.

---- Insert Table 3 here. ----

---- Insert Table 4 here. ----

In Figure 3, the best elitist solution for one of the random problem instances with n=100 and C=8 is presented. Note that this solution is found for α=0.95. Therefore, the servers are assigned to the perimeter nodes as the optimal solutions given in Table 1 for high values of α.

---- Insert Figure 3 Here ----

Among the three metaheuristics approaches studied in this section, the main advantage of PSO is its ease of implementation. PSO is a very simple algorithm to implement, but it has been proven to be effective to solve a wide variety of problems. PSO has a limited number of parameters to tune, and the convergence properties of PSO with respect to various parameter settings are well studied in the literature. PSO does not rely on special operators unlike such as, crossover in GA or move operator in SA and TS. PSO can be directly used in wide variety of problems where a solution is represented as a vector. The main disadvantage of PSO is that it is originally intended to solve problems with continuous decision variables. Therefore, adapting PSO to solve combinatorial optimization problems requires additional transformation from continuous space to discrete space. The PSO in this paper utilizes a simple construction heuristic to achieve this. The main advantage of the ACO in this paper is its fast convergence due to incorporating problem specific heuristic information into the search. However, the ACO does not perform well in the large-size problems. In addition, the performance of the ACO highly

depends on both ACO parameters and selected heuristic specific information, and determining their best combination could be difficult. Compared to PSO and ACO, CSA is a new nature inspired metaheuristic with relatively limited applications in the literature. The main advantage of the CSA in this paper is its ability to discover a diverse set of solutions.

V. Conclusions

In this paper, the reliable server assignment problem in networks is studied. To ensure an uninterrupted level of service between the clients and the server(s), first, a new network reliability measure, CSR, is defined, and then three heuristic approaches, ACO, PSO, and CSA are applied to the problem. A rigorous experimental study is conducted to investigate the performances of the three heuristics under different scenarios. The experimental study show that the ACO is able to quickly discover good solutions to the problem and the PSO converges slowly but it eventually outperforms the ACO particularly for large size problems. The performance of the CSA is par with the PSO. Different convergence properties of the three heuristics suggest that hybridizing may be beneficial. For example, solutions found by the ACO in the early stages of the search can be improved by the PSO or CSA.

References

[1] M. Ball and J. Provan, "Calculating bounds on reachability and connectedness in stochastic networks," *Networks,* vol. 13, pp. 253-278, 2006.

[2] S. L. Hakimi, "Optimum locations of switching centers and absolute centers and medians of graph," *Operations Research,* vol. 12, pp. 450-459, 1964.

[3] Z. Drezner, "Heuristic solution methods for two location problems with unreliable facilities," *Journal of the Operational Research Society,* vol. 38, pp. 509-514, 1987.

[4] L. D. Nel and C. J. Colbourn, "Locating a broadcast facility in an unreliable network," *INFOR,* vol. 28, pp. 363-379, 1990.

[5] E. Melachrinoudis and M. E. Helander, "A single facility location problem on a tree with unreliable edges," *Networks,* vol. 27, pp. 219-237, 1996.

[6] A. Nakaniwa, *et al.*, "Reliability-based optimal allocation of mirror servers for Internet," in *IEEE Global Telecommunications Conference*, San Francisco, CA, USA, 2000, pp. 1571-1577.

[7] L. V. Snyder and M. S. Daskin, "Reliability models for facility location: the expected failure cost case," *Transportation Science,* vol. 39, pp. 400-416, 2005.

[8] H. A. Eiselt, *et al.*, "Location of facilities on a network subject to a single-edge failure," *Networks,* vol. 22, pp. 231-246, 1992.

[9] M. Ball, "Complexity of Network Reliability Calculation," *Networks,* vol. 10, pp. 153-165, 1980.

[10] O. Berman and Z. Drezner, "A probabilistic one-centre location problem on a network," *Journal of the Operational Research Society,* vol. 54, pp. 871-877, 2003.

[11] H. A. Eiselt, *et al.*, "Optimal location of facilities on a network with an unreliable node or link," *Information Processing Letters,* vol. 58, pp. 71-74, 1996.

[12] O. Berman, *et al.*, "Locating service facilities whose reliability is distance dependent," *Computers & Operations Research,* vol. 30, pp. 1683-1695, 2003.

[13] A. Nakaniwa, *et al.*, "Reliability-based mirroring of servers in distributed networks," *IEICE Transactions on Communications,* vol. E85-B, pp. 540-549, 2002.

[14] N. Krasnogor, *et al.*, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)* vol. 129: Springer Berlin / Heidelberg, 2008.

[15] M. Dorigo, *et al.*, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics),* vol. 26, pp. 29-41, 1996.

[16] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation,* vol. 1, pp. 53-66, 1997.

[17] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theoretical Computer Science,* vol. 344, pp. 243-278, 2005.

[18] M. Zlochin, *et al.*, "Model-based search for combinatorial optimization: a critical survey," *Annals of Operations Research,* vol. 131, pp. 373-395, 2004.

[19] S. D. Shtovba, "Ant algorithms: theory and applications," *Programming and Computer Software,* vol. 31, pp. 167-178, 2005.

[20] D. L. Woodruff and E. Zemel, "Hashing vectors for tabu search," *Annals of Operations Research,* vol. 41, pp. 123-137, 1993.

[21] R. Battiti and G. Tecchiolli, "The reactive tabu search," *ORSA Journal on Computing*, vol. 6, pp. 126-140, 1994.
[22] A. Konak, "Efficient event-driven simulation approaches to analysis of network reliability and performability," *International Journal of Modelling and Simulation*, vol. 29, pp. 156-168, 2009.
[23] M. Marseguerra and E. Zio, "Optimizing maintenance and repair policies via a combination of genetic algorithms and Monte Carlo simulation," *Reliability Engineering & System Safety*, vol. 68, pp. 69-83, 2000.
[24] M. Marseguerra, *et al.*, "Condition-based maintenance optimization by means of genetic algorithms and Monte Carlo simulation," *Reliability Engineering and System Safety*, vol. 77, pp. 151-165, 2002.
[25] M. Marseguerra, *et al.*, "Multiobjective spare part allocation by means of genetic algorithms and Monte Carlo simulation," *Reliability Engineering and System Safety*, vol. 87, pp. 325-335, 2005.
[26] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Sixth International Symposium on Micro Machine and Human Science*, Nagoya, 1995, pp. 39–43.
[27] E. Ozcan and C. K. Mohan, "Particle swarm optimization: surfing the waves," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*, Piscataway, NJ, USA, 1999, pp. 1939-1944.
[28] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 58-73, 2002.
[29] J. Kennedy and R. C. Eberhart, "Discrete binary version of the particle swarm algorithm," in *the IEEE International Conference on Systems, Man and Cybernetics*, 1997, pp. 4104–4108.
[30] J. D. Farmer, *et al.*, "The immune system, adaptation, and machine learning," in *Physica D (Netherlands)*, Netherlands, 1986, pp. 187-204.
[31] A. S. Perelson, "Immune network theory," *Immunological Reviews*, vol. 110, p. 5, 1989.
[32] F. J. Varela, *et al.*, "Cognitive networks: Immune, neural and otherwise," *Theoretical Immunology* vol. 2, pp. 359-375, 1988.
[33] D. Dasgupta, "An overview of Artificial Immune Systems and their applications," in *Artificial immune systems and their applications*, ed New York: Springer-Verlag 1998, pp. 3-18.
[34] D. Dasgupta and L. P. Nino, *Immunological Computation: Theory and Applications*: CRC Press, 2009.
[35] E. Hart and J. Timmis, "Application areas of AIS: the past, the present and the future," *Applied Soft Computing Journal*, vol. 8, pp. 191-201, 2009.
[36] F. M. Burnet, *The Clonal Selection Theory of Acquired Immunity*: Cambridge University Press, 1959.
[37] L. N. De Castro and F. J. Von Zuben, "The clonal selection algorithm with engineering applications," in *Proceedings of GECCO'00*, Las Vegas, USA, 2000, pp. 36-37.
[38] L. N. de Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 239-251, 2002.

[39] J. Brownlee, "Clonal Selection Algorithms," Complex Intelligent Systems Laboratory (CIS), Swinburne University of Technology, Victoria, Australia Report ID: 070209A, 2007.
[40] E. Elbeltagi, *et al.*, "Comparison among five evolutionary-based optimization algorithms," *Advanced Engineering Informatics*, vol. 19, pp. 43-53, 2005.
[41] F. Moussouni, *et al.*, "Comparison of Two Multi-Agent Algorithms: ACO and PSO for the Optimization of a Brushless DC Wheel Motor," in *Intelligent Computer Techniques in Applied Electromagnetics*, ed, 2008, pp. 3-10.
[42] R. Saravanan, *et al.*, "Optimization of cuffing conditions during continuous finished profile machining using non-traditional techniques," *International Journal of Advanced Manufacturing Technology*, vol. 26, pp. 30-40, 2005.

Figures

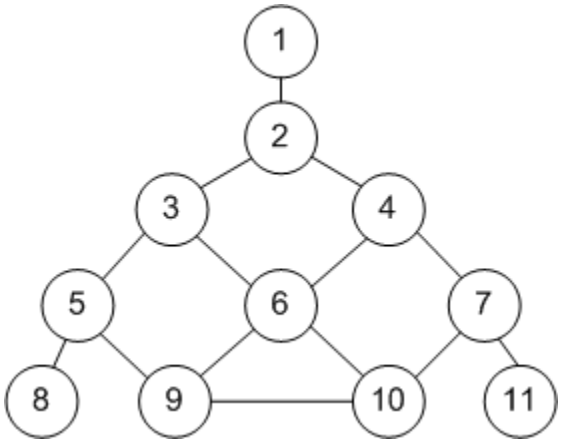

Figure 1 The test network used to study the structures of optimal solutions to the problem.

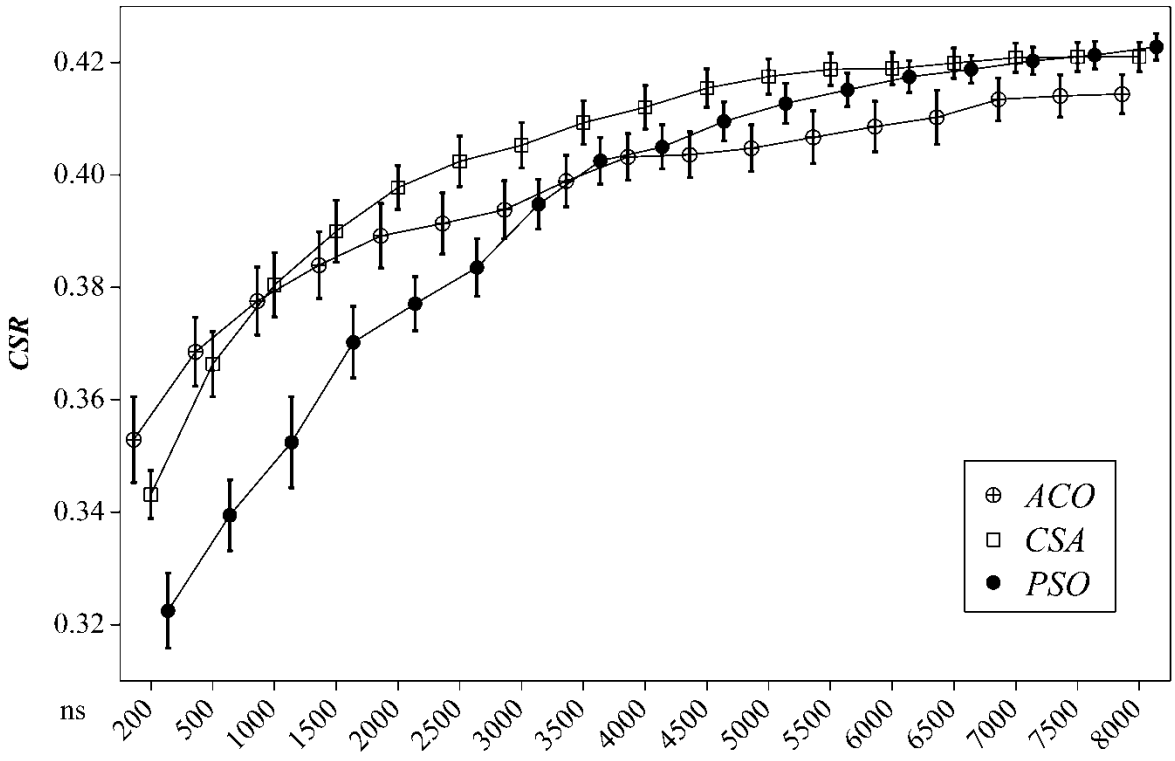

Figure 2 For problem n=100 and C=8, 95% confidence intervals (ten random replications of a single problem instance) of the best solutions so far are plotted against the number of the solutions searched (*ns*).

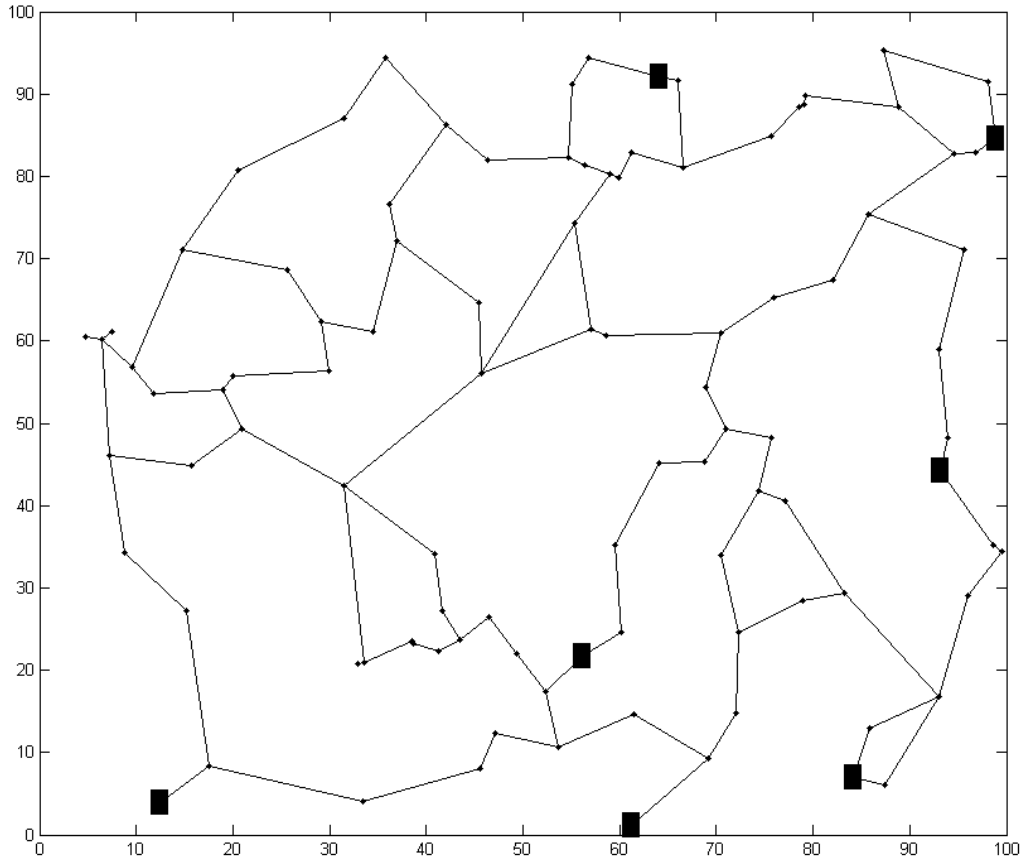

Figure 3 The best solutions for a random problem with n=100, C=8, α=0.95, and ns=8000. This solution has similar features with the optimal solutions given in Table 1 for high values of α.

Tables

Table 1 The optimal solutions for the test network given in Figure 1. CSR is exactly calculated using a backtracking algorithm and the optimal solutions are investigated using explicit enumeration.

		With Reliable Nodes		With Unreliable Nodes	
C	α	Server Nodes	CSR	Server Nodes	CSR
1	0.50	{6}	0.999024	{6}	0.946783
2	0.50	{2,9},{2,10}	0.999952	{4,9},{3,10}	0.995303
3	0.50	{2,5,10}	0.999999	{2,5,10},{2,7,9}	0.999390
1	0.6	{6}	0.997072	{6}	0.941139
2	0.6	{3,7},{4,5}	0.999790	{3,7},{4,5}	0.992498
3	0.6	{2,8,10},{2,9,11}	0.999992	{2,5,7}	0.998689
1	0.7	{6}	0.995235	{6}	0.931498
2	0.7	{5,7}	0.999116	{5,7}	0.986367
3	0.7	{2,5,7}	0.999943	{1,5,7}	0.996930
1	0.8	{6}	0.984725	{6}	0.900560
2	0.8	{5,11},{7,8}	0.995071	{5,7}	0.965170
3	0.8	{1,5,7}	0.999727	{1,5,11},{1,7,8}	0.991649
1	0.9	{2},{5},{7}	0.935116	{2}	0.813771
2	0.9	{5,11},{7,8}	0.975395	{5,11}, {7,8}	0.906875
3	0.9	{1,5,11},{1,7,8}	0.998443	{1,8,11}	0.978258
1	1.0	{1},{2},…,{11}	0.699518	{2}	0.587003
2	1.0	{8,11}	0.872656	{8,11}	0.790276
3	1.0	{1,8,11}	0.990659	{1,8,11}	0.967072

Table 2 Experimental design and results (p-value, R^2 (adjusted)). The metaheuristics approaches are ordered based on the average performance. Statically significant results with a confidence level 95% are set as bold.

ns=	1000	1000	8000	8000
(*n*, *m*)\ *C*=	5	8	5	8
(30,36)	**ACO**>PSO≈CSA 0.00, 0.95	**CSA**>ACO≈PSO 0.047, 0.78	CSA>PSO≈ACO 0.238, 0.99	CSA≈PSO>ACO 0.82, 0.98
(40,53)	ACO≈PSO≈CSA 0.099, 0.87	**ACO≈PSO**>CSA 0.00, 0.73	**CSA**>PSO≈ACO 0.00, 0.93	PSO>PSO≈ACO 0.00, 0.96
(50,98)	**ACO≈PSO**>CSA 0.00, 0.79	**ACO**>PSO>CSA 0.00, 0.44	ACO>CSA>PSO 0.088, 0.99	ACO>CSA≈PSO 0.19, 0.78
(60,118)	**ACO**>CSA>PSO 0.00, 0.66	**ACO**>PSO>CSA 0.00, 0.54	ACO≈CSA>PSO 0.023, 0.96	**PSO**>ACO>CSA 0.00, 0.85
(70,138)	**ACO**>CSA>PSO 0.00, 0.62	**ACO**>CSA>PSO 0.00, 0.61	ACO>PSO>CSA 0.221, 0.97	**PSO**>ACO>CSA 0.00, 0.86
(80,158)	**ACO≈CSA**>PSO 0.00, 0.70	**ACO**>CSA>PSO 0.00, 0.55	PSO>CSA>ACO 0.908, 0.97	**PSO**>ACO>CSA 0.00, 0.79
(100,115)	**ACO≈CSA**>PSO 0.00, 0.74	**ACO**>CSA>PSO 0.00, 0.54	ACO>CSA>PSO 0.256, 0.98	**PSO**>ACO≈CSA 0.00, 0.71

Table 3 Computational results for *C*=5 and *ns*=8000.

	ACO			PSO			CSA		
(*n*, *m*)	(Elitist Range)/σ	% Collision	CPU Sec.	(Elitist Range)/σ	Collision %	CPU Sec.	(Elitist Range)/σ	Collision %	CPU Sec.
(30, 36)	20.1	3.7	141	20.9	1.6	91	19.7	2.8	133
(40, 53)	17.1	3.9	268	15.6	1.9	180	15.7	3.2	254
(50, 98)	15.5	4.1	428	12.9	2.8	344	13.4	3.4	397
(60, 118)	15.9	4.3	664	14.1	3.1	568	15.0	3.4	589
(70, 138)	17.2	4.4	837	13.8	3.5	771	15.6	3.6	775
(80, 158)	18.0	4.4	1129	13.4	3.9	1098	15.4	3.7	1035
(100, 115)	25.0	4.4	1554	17.9	4.2	1599	19.2	3.9	1525

Table 4 Computational results for *C*=8 and *ns*=8000.

	ACO			PSO			CSA		
(*n*, *m*)	(Elitist Range)/σ	% Collision	CPU Sec.	(Elitist Range)/σ	Collision %	CPU Sec.	(Elitist Range)/σ	Collision %	CPU Sec.
(30, 36)	23.7	0.7	75	24.0	0.4	65	23.9	0.6	81
(40, 53)	33.5	1.2	134	36.8	0.7	111	31.0	1.4	164
(50, 98)	18.8	2.3	310	19.8	1.6	261	18.9	2.0	295
(60, 118)	17.5	2.9	517	17.3	1.9	429	18.0	2.3	474
(70, 138)	17.8	3.4	689	17.8	2.5	589	17.6	2.6	633
(80, 158)	11.3	3.6	969	11.6	3.0	881	11.3	2.9	886
(100, 115)	14.0	4.0	1477	13.2	3.6	1446	13.1	3.3	1380