An Ant Colony Approach to Redundancy Allocation

Yun-Chia Liang and Alice E. Smith, Senior Member, IEEE
Department of Industrial and Systems Engineering
Auburn University
207 Dunstan Hall
Auburn, AL 36849 USA

Submitted to *IEEE Transactions on Reliability*

October 2001

An Ant Colony Approach to Redundancy Allocation

Keywords: redundancy allocation, series-parallel system, ant colony optimization, genetic algorithm, combinatorial optimization

I. Summary & Conclusions

This paper uses an ant colony meta-heuristic optimization method to solve the redundancy allocation problem (RAP). The RAP is a well known NP-hard problem that has been the subject of much prior work, generally in a restricted form where each subsystem must consist of identical components in parallel to make computations tractable. The newer meta-heuristic methods overcome this limitation and offer a practical way to solve large instances of the relaxed RAP where different components can be placed in parallel. The ant colony method has not yet been used in reliability design, yet it is a method that is expressly designed for combinatorial problems with a good neighborhood structure, as in the case of the RAP. An effective ant colony optimization algorithm for the RAP is devised and tested on the most well known suite of problems from the literature. It is shown that the ant colony method has excellent performance with very little variability over problem instance or random number seed. It is easily competitive with the best-known heuristics for redundancy allocation.

II. Introduction

The most studied design configuration of the redundancy allocation problem (RAP) is a series system of s independent k-out-of-n:G subsystems (Figure 1). A subsystem is functioning properly if at least k_i of its p_i components are operational. If k_i is one for all subsystems, then it is a series-parallel system. The RAP is NP-hard [6] and has been studied in many different forms as summarized in Tillman, et al. [31], and most recently by Kuo & Prasad [23].

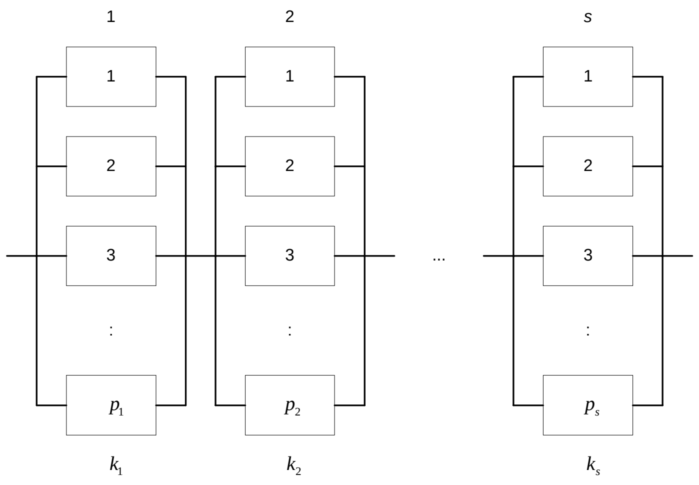


Figure 1. Series-parallel system configuration.

The RAP can be formulated to maximize system reliability given restrictions on system cost of *C* and system weight of *W*. It is assumed that system weight and system cost are linear combinations of component weight and cost, although this is a restriction which can readily be relaxed using a meta-heuristic approach.

$$\max \quad R = \prod_{i=1}^{s} R_i(\mathbf{y}_i \mid k_i) \tag{1}$$

Subject to the constraints

$$\sum_{i=1}^{s} C_i(\mathbf{y}_i) \le C,$$

$$\sum_{i=1}^{s} W_i(\mathbf{y}_i) \le W,$$

If there is a pre-selected maximum number of components in parallel, the following constraint is added:

$$k_i \le \sum_{j=1}^{a_i} y_{ij} \le p_{\max} \quad \forall i = 1,2,\dots,s$$

Typical assumptions are:

- The states of components and the system have only two options - good or failed.

- Failed components do not damage the system, and are not repaired.
- The failure rates of components when not in use are the same as when in use (i.e., active redundancy).
- Component attributes including reliabilities are known and deterministic.
- The supply of components is unlimited (i.e., off-the-shelf).

The series-parallel RAP has been widely studied. Traditional approaches include dynamic programming (e.g., [2, 19, 27]), integer programming (e.g., [3, 21, 22, 26]), and mixed-integer and nonlinear programming (e.g., [32]). More recently, genetic algorithm (GA) meta-heuristic formulations for the RAP have been proposed by Painton & Campbell [28], Levitin, et al. [24], and Coit & Smith [7, 8]. For a fixed design configuration and known incremental decreases in component failure rates and their associated costs, [28] uses a GA to find a maximum reliability solution to satisfy specific cost constraints. An algorithm to optimize the 5th percentile of the mean-time-between-failure distribution is devised. Levitin, et al. [24] generalize a redundancy allocation problem to multi-state systems, where the system and its components have a range of performance levels - from perfect functioning to complete failure. A universal moment generating function is used to estimate system performance (capacity or operation time), and a GA is employed as the optimization technique. Coit & Smith [7] use a penalty guided GA which searches over feasible and infeasible regions to identify a final, feasible optimal, or near optimal, solution to a relaxed version of the RAP. Coit & Smith [8] also applied GA to problems with stochastic system reliability and mean-time-to-failure.

Because of the search space size of the RAP and lack of dominant heuristic, it is a good candidate for other meta-heuristic approaches including the focus of this paper, the ant colony optimization (ACO).

Notation

Ant Colony Optimization (ACO)

τ_{ij}	pheromone trail of combination (i, j)
P_{ij}	transition probability of combination (i, j)
η_{ij}	problem-specific heuristic of combination (i, j)
α	relative importance of the pheromone trail
β	relative importance of the problem-specific heuristic
ρ	$\in [0,1]$, trail persistence
q	$\in [0,1]$, a uniformly generated random number
q_0	$\in [0,1]$, a parameter which determines the relative importance of exploitation versus exploration
$\Delta\tau_{ij}^k$	quantity of pheromone trail added to τ_{ij} by the k^{th} ant
γ	amplification parameter in the penalty function

Redundancy Allocation Problem (RAP)

R	overall reliability of the series-parallel system
C	cost constraint
W	weight constraint
s	number of subsystems
a_i	number of available component choices for subsystem i
r_{ij}	reliability of component j available for subsystem i
c_{ij}	cost of component j available for subsystem i

w_{ij}	weight of component j available for subsystem i
y_{ij}	quantity of component j used in subsystem i
$\mathbf{y}_i$	$(y_{i1},...,y_{ia_i})$
p_i	$=\sum_{j=1}^{a_i} y_{ij}$, total number of components used in subsystem i
$p_{\max}$	maximum number of components in parallel (user assigned)
k_i	minimum number of components in parallel required for subsystem i to function
AC	set of available component choices

III. The Ant Colony Optimization Approach

Ant Colony Optimization (ACO) is one of the adaptive meta-heuristic optimization methods inspired by nature which include simulated annealing, genetic algorithms and tabu search. ACO is distinctly different from these methods in that it is a constructive, rather than an improvement, algorithm. ACO was inspired by the behavior of real ants. Ethologists have studied how blind animals, such as ants, could establish shortest paths from their nest to food sources. The medium that is used to communicate information among individual ants regarding paths is pheromone trails. A moving ant lays some pheromone on the ground, thus marking the path. The pheromone, while gradually dissipating over time, is reinforced as other ants use the same trail. Therefore, efficient trails increase their pheromone level over time while poor ones reduce to nill. Inspired by this behavior of real ants, Marco Dorigo first introduced the ant colony optimization approach in his Ph.D. thesis in 1992 [13] and expanded it in his further work as summarized in [14, 15, 18]. The characteristics of an artificial ant colony include a method to construct solutions that balances pheromone trails and a problem-specific heuristic, a method to both reinforce and evaporate pheromone, and local search to improve the constructed solutions.

ACO methods have been successfully applied to diverse combinatorial optimization problems including traveling salesman [16, 17], quadratic assignment [25, 30], vehicle routing [4, 5, 20], telecommunication networks [12], graph coloring [10], constraint satisfaction [29], Hamiltonian graphs [33] and scheduling [1, 9, 11].

A. Overview of the Method and Encoding

The overview of the method is as follows.

```
Set all parameters and initialize the pheromone trails
Loop
    Sub-Loop
        Generate an ant based on the state transition rule
        Apply the online pheromone update rule
    Continue until all ants in the colony have been generated
    Apply local search to each ant
    Evaluate all ants in the colony, rank them and record the best feasible one
    Apply the offline pheromone update rule
Continue until a stopping criterion is reached
```

Each ant represents one design of the entire system, a collection of p_i parts in parallel $(k_i \le p_i \le p_{\max})$ for s different subsystems. The p_i parts can be chosen in any combination from the a_i available type of components. The a_i components are indexed in descending order in accordance with their reliability; i.e., 1 represents the most reliable component, etc. An index of $a_i + 1$ is assigned to a position where an additional component was not used (left blank), i.e., with attributes of zero. Each of the s subsystems is represented by $p_{\max}$ positions with each component listed according to its reliability index, as in [7].

B. Initial Solution Construction

In the ACO-RAP algorithm, ants use problem-specific heuristic information, denoted by η_{ij}, as well as pheromone trails, denoted by τ_{ij}, to choose p_i components ($k_i + 1 \leq p_i \leq p_{\max} - 4$) in each subsystem as follows where component *v* is selected for subsystem *i*:

$$v = \begin{cases} \arg \max\limits_{l \in AC} [(\tau_{il})^{\alpha} (\eta_{il})^{\beta}] & q \leq q_0 \\ V & q > q_0 \end{cases} \tag{2}$$

and *V* is selected according to the transition probability given by

$$P_{iv} = \begin{cases} \dfrac{(\tau_{iv})^{\alpha} (\eta_{iv})^{\beta}}{\sum\limits_{l \in AC} (\tau_{il})^{\alpha} (\eta_{il})^{\beta}} & v \in AC \\ 0 & Otherwise \end{cases} \tag{3}$$

where α and β are parameters that control the relative weight of the pheromone and the local heuristic, respectively, *AC* is the set of available component choices for subsystem *i*, q is a random number uniformly generated between 0 and 1, and q_0 is a parameter which determines the relative importance of exploitation versus exploration. When $q \leq q_0$ exploitation of the knowledge available about the problem in the form of problem-specific heuristic and pheromone trails are used, whereas $q > q_0$ favors more (random) exploration. There is no pheromone value for the "blank."

The problem specific heuristic used is $\eta_{ij} = \frac{r_{ij}}{c_{ij} + w_{ij}}$ where r_{ij}, c_{ij}, and w_{ij} represent the associated reliability, cost and weight of component *j* for subsystem *i*. Components with higher reliability and smaller cost and weight have greater probability to be selected.

C. Objective Function

After all components of an ant *k* are selected, the unpenalized reliability R_k is calculated using (1). The penalized reliability R_{kp} for systems that exceed *C* and / or *W* is calculated:

$$R_{kp} = R_k \cdot \left(\frac{W}{W_k}\right)^{\gamma} \cdot \left(\frac{C}{C_k}\right)^{\gamma} \qquad (4)$$

where the exponent γ is a preset amplification parameter. This penalty function encourages the ACO-RAP algorithm to explore the feasible region and infeasible region that is near the border of feasible area, and discourages, but permits, search further into the infeasible region.

D. Improving Constructed Solutions Through Local Search

After each colony is generated, each ants is improved using local search. Starting with the first subsystem, a chosen component type is deleted and another component type is added. All possibilities are enumerated. For example, if a subsystem has one of component 1, two of component 2 and one of component 3, then one alternative is to delete a component 1 and to add a component 2. Another possibility is to delete a component 3 and to add a component 1. Whenever an improvement of the objective function is detected, the new solution replaces the old one and the process continues until all subsystems have been searched. This local search method does not require recalculating the entire system reliability each time, only the reliability of the subsystem under consideration needs to be recalculated and system reliability is updated accordingly.

E. Pheromone Trail Update

The pheromone update consists of two phases – online (ant-by-ant) updating and offline (colony) updating. Online updating is done after each ant is constructed and its purpose is to decay the pheromone intensity of the components that ant selected to encourage exploration of other component choices in the subsequent ants to be constructed. Online updating is by

$$\tau_{ij}^{new} = \rho \cdot \tau_{ij}^{old} + (1-\rho) \cdot \tau_{\text{io}} \tag{5}$$

where initial trail intensities (τ_{io}) are set to $\frac{1}{a_i}$. After all ants have constructed a complete system and have applied local search, pheromone trails are updated offline. Offline updating is to reflect the discoveries of this colony, that is, this iteration. The offline trail update is:

$$\tau_{ij}^{new} = \rho \cdot \tau_{ij}^{old} + (1-\rho) \cdot \sum_{m=1}^{E} (E-m+1) \cdot R_m \tag{6}$$

where m = 1 is the best feasible ant yet found (which may or may not be in the current colony) and the remaining ants are the top ones in the current colony.

IV. Computational Experience

The ACO is coded in Borland C++ and run using an Intel Pentium III 800 MHz PC with 256 MB RAM. All computations use real float point precision without rounding or truncating values. The system reliability of the final solution is rounded to four digits behind the decimal point in order to compare with results in literature.

The parameters of the ACO algorithm are set to the following values: $\alpha = 1$, $\beta = 0.5$, $q_0 = 0.9$, $\rho = 0.9$ and E = 5. This gives relatively more weight to the pheromone trail than the problem-specific heuristic and greater emphasis on exploitation rather than exploration. The ACO is not very sensitive to changes in these values and tested well for quite a range of them.

For the penalty function, $\gamma = 0.1$ except when the previous iteration has 90% or more infeasible ants, then $\gamma = 0.3$. This increases the penalty temporarily to move the search back into the feasible region if all or nearly all solutions in a colony are not feasible. It was found that this dual level penalty improved performance on the most constrained instances of the test problems. Because of changes in the magnitudes of *R*, *C* and *W*, all η_{ij} and τ_{ij} are normalized between (0,1) before the state transition rules apply. 100 ants are used in each iteration. The stopping criterion is when the number of iterations reaches 1000 or the best feasible ant has not changed for 500 consecutive iterations. This results in a maximum of 100,000 ants.

The 33 variations of the Fyffe, et al. problem [19], which were devised by Nakagawa & Miyazaki [27] were used to test the performance of ACO. In this set $C = 130$ and W is increased incrementally from 159 to 191. In [19] and [27], the optimization approaches required that only identical components be placed in redundancy, however for the ACO approach, as in Coit & Smith [7], different types are allowed to reside in parallel assuming that a value of $p_{\max}$ = 8 for all subsystems. Allowing component mixing, the search space size is larger than 7.6×10^{33}. Since the heuristic benchmark for the RAP where component mixing is allowed is the GA of [7], it is chosen for comparison. Ten runs of each algorithm were made using different random number seeds for each problem instance.

The results are summarized in Table 1 where the shaded box shows the best solution. ACO is even or superior to the GA in all categories and all problem instances. When the problem instances are less constrained (the first 18), the ACO performs much better than the GA. When the problems become more constrained (the last 15), ACO is equal to GA for 12 instances and better than GA for 3 instances in terms of the Max R measure (best over 10 runs). However, for Min R (worst over 10 runs) and Mean R (of 10 runs), ACO dominates GA. Thus, the ACO tends to find better solutions than the GA, is significantly less sensitive to random number seed, and for the 12 most constrained instances, finds the best solution each and every run.

The configuration of the best solution and its system reliability, cost and weight for each of the 33 instances are shown in Table 2. For instances 6 and 11, two alternatives with different system cost but the same reliability and weight are found. All but instance 33 involve mixing of components which is an indication that better designs can be identified by not restricting the search space to a single component type per subsystem.

Table 1. Comparison of GA and ACO over 10 random number seeds each.

			C&S GA - 10 runs			**ACO-RAP - 10 runs**		
No	**C**	**W**	**Max R**	**Mean R**	**Min R**	**Max R**	**Mean R**	**Min R**
1	130	191	0.9867	0.9862	0.9854	0.9868	0.9862	0.9860
2	130	190	0.9857	0.9855	0.9852	0.9859	0.9858	0.9857
3	130	189	0.9856	0.9850	0.9838	0.9858	0.9853	0.9852
4	130	188	0.9850	0.9848	0.9842	0.9853	0.9849	0.9848
5	130	187	0.9844	0.9841	0.9835	0.9847	0.9841	0.9837
6	130	186	0.9836	0.9833	0.9827	0.9838	0.9836	0.9835
7	130	185	0.9831	0.9826	0.9822	0.9835	0.9830	0.9828
8	130	184	0.9823	0.9819	0.9812	0.9830	0.9824	0.9820
9	130	183	0.9819	0.9814	0.9812	0.9822	0.9818	0.9817
10	130	182	0.9811	0.9806	0.9803	0.9815	0.9812	0.9806
11	130	181	0.9802	0.9801	0.9800	0.9807	0.9806	0.9804
12	130	180	0.9797	0.9793	0.9782	0.9803	0.9798	0.9796
13	130	179	0.9791	0.9786	0.9780	0.9795	0.9795	0.9795
14	130	178	0.9783	0.9780	0.9764	0.9784	0.9784	0.9783
15	130	177	0.9772	0.9771	0.9770	0.9776	0.9776	0.9776
16	130	176	0.9764	0.9760	0.9751	0.9765	0.9765	0.9765
17	130	175	0.9753	0.9753	0.9753	0.9757	0.9754	0.9753
18	130	174	0.9744	0.9732	0.9716	0.9749	0.9747	0.9741
19	130	173	0.9738	0.9732	0.9719	0.9738	0.9735	0.9731
20	130	172	0.9727	0.9725	0.9712	0.9730	0.9726	0.9714
21	130	171	0.9719	0.9712	0.9701	0.9719	0.9717	0.9710
22	130	170	0.9708	0.9705	0.9695	0.9708	0.9708	0.9708
23	130	169	0.9692	0.9689	0.9684	0.9693	0.9693	0.9693
24	130	168	0.9681	0.9674	0.9662	0.9681	0.9681	0.9681
25	130	167	0.9663	0.9661	0.9657	0.9663	0.9663	0.9663
26	130	166	0.9650	0.9647	0.9636	0.9650	0.9650	0.9650
27	130	165	0.9637	0.9632	0.9627	0.9637	0.9637	0.9637
28	130	164	0.9624	0.9620	0.9609	0.9624	0.9624	0.9624
29	130	163	0.9606	0.9602	0.9592	0.9606	0.9606	0.9606
30	130	162	0.9591	0.9587	0.9579	0.9592	0.9592	0.9592
31	130	161	0.9580	0.9572	0.9561	0.9580	0.9580	0.9580
32	130	160	0.9557	0.9556	0.9554	0.9557	0.9557	0.9557
33	130	159	0.9546	0.9538	0.9531	0.9546	0.9546	0.9546

Table 2. Configuration, reliability, cost and weight of the best solution to each problem.

No.	W	R	Cost	Weight	Solution
1	191	0.9868	130	191	333,11,111,2222,333,22,333,3333,23,122,333,4444,12,12
2	190	0.9859	129	190	333,11,111,2222,333,22,333,3333,22,112,333,4444,11,22
3	189	0.9858	130	189	333,11,111,2222,333,22,333,3333,22,122,11,4444,11,12
4	188	0.9853	130	188	333,11,111,2222,333,22,333,3333,23,112,13,4444,12,12
5	187	0.9847	130	187	333,11,111,2222,333,22,333,3333,23,122,13,4444,11,12
6	186	0.9838	129	186	333,11,111,2222,333,22,333,3333,22,122,11,4444,11,22
			130	186	333,11,111,2222,333,24,333,3333,33,122,13,4444,12,12
7	185	0.9835	130	185	333,11,111,2222,333,22,333,3333,13,122,13,4444,11,22
8	184	0.9830	130	184	333,11,111,222,333,22,333,3333,33,112,11,4444,11,12
9	183	0.9822	128	183	333,11,111,222,333,22,333,3333,33,112,13,4444,11,12
10	182	0.9815	127	182	333,11,111,222,333,22,333,3333,33,122,13,4444,11,12
11	181	0.9807	125	181	333,11,111,222,333,22,333,3333,13,122,13,4444,11,22
			126	181	333,11,111,222,333,22,333,3333,23,122,11,4444,11,22
12	180	0.9803	128	180	333,11,111,222,333,22,333,3333,33,122,11,4444,11,22
13	179	0.9795	126	179	333,11,111,222,333,22,333,3333,33,122,13,4444,11,22
14	178	0.9784	125	178	333,11,111,222,333,22,333,3333,33,222,13,4444,11,22
15	177	0.9776	126	177	333,11,111,222,333,22,333,133,33,122,13,4444,11,22
16	176	0.9765	125	176	333,11,111,222,333,22,333,133,33,222,13,4444,11,22
17	175	0.9757	125	175	333,11,111,222,333,22,13,3333,33,122,11,4444,11,22
18	174	0.9749	123	174	333,11,111,222,333,22,13,3333,33,122,13,4444,11,22
19	173	0.9738	122	173	333,11,111,222,333,22,13,3333,33,222,13,4444,11,22
20	172	0.9730	123	172	333,11,111,222,333,22,13,133,33,122,13,4444,11,22
21	171	0.9719	122	171	333,11,111,222,333,22,13,133,33,222,13,4444,11,22
22	170	0.9708	120	170	333,11,111,222,333,22,13,133,33,222,33,4444,11,22
23	169	0.9693	121	169	333,11,111,222,333,22,33,133,33,222,13,4444,11,22
24	168	0.9681	119	168	333,11,111,222,333,22,33,133,33,222,33,4444,11,22
25	167	0.9663	118	167	333,11,111,222,33,22,13,133,33,222,33,4444,11,22
26	166	0.9650	116	166	333,11,11,222,333,22,13,133,33,222,33,4444,11,22
27	165	0.9637	117	165	333,11,111,222,33,22,33,133,33,222,33,4444,11,22
28	164	0.9624	115	164	333,11,11,222,333,22,33,133,33,222,33,4444,11,22
29	163	0.9606	114	163	333,11,11,222,33,22,13,133,33,222,33,4444,11,22
30	162	0.9592	115	162	333,11,11,222,33,22,33,133,33,222,13,4444,11,22
31	161	0.9580	113	161	333,11,11,222,33,22,33,133,33,222,33,4444,11,22
32	160	0.9557	112	160	333,11,11,222,33,22,33,333,33,222,13,4444,11,22
33	159	0.9546	110	159	333,11,11,222,33,22,33,333,33,222,33,4444,11,22

It is difficult to make a precise computational comparison. CPU seconds vary according to hardware, software and coding. Both the ACO and the GA are population-based and iterative, therefore the computational effort changes in direct proportion to these factors. The number of solutions constructed in [7] (a population size of 40 with 1200 iterations) is about half of the ACO algorithm (a colony size of 100 with up to 1000 iterations). However, given the improved performance per seed of the ACO, a direct comparison per run is not meaningful. If the average solution of the ACO is compared to the best performance of GA, in 13 instances ACO is better, in 9 instances GA is better and in the remaining instances (11) they are equal, as shown in Figure 2. Since this is a comparison of average performance (ACO) versus best of ten performance

(GA), the additional computational effort of the ACO is more than compensated for. In summary, an average run of ACO is apt to be as good or better than the best of 10 runs of GA. The difference in variability over all 33 test problems is clearly shown in Figure 3.

Given the well-structured neighborhood of the RAP, a meta-heuristic that exploits it is likely to be more effective and more efficient than one that does not. While the GA certainly performs well relative to previous approaches, the mostly random mechanisms of crossover and mutation result in greater run to run variability than the ACO. Since the ACO shares the GA's attributes of flexibility, robustness and implementation ease and improves on its random behavior, it seems a very promising general method for other NP-hard reliability design problems such as those found in networks and other complex structures.

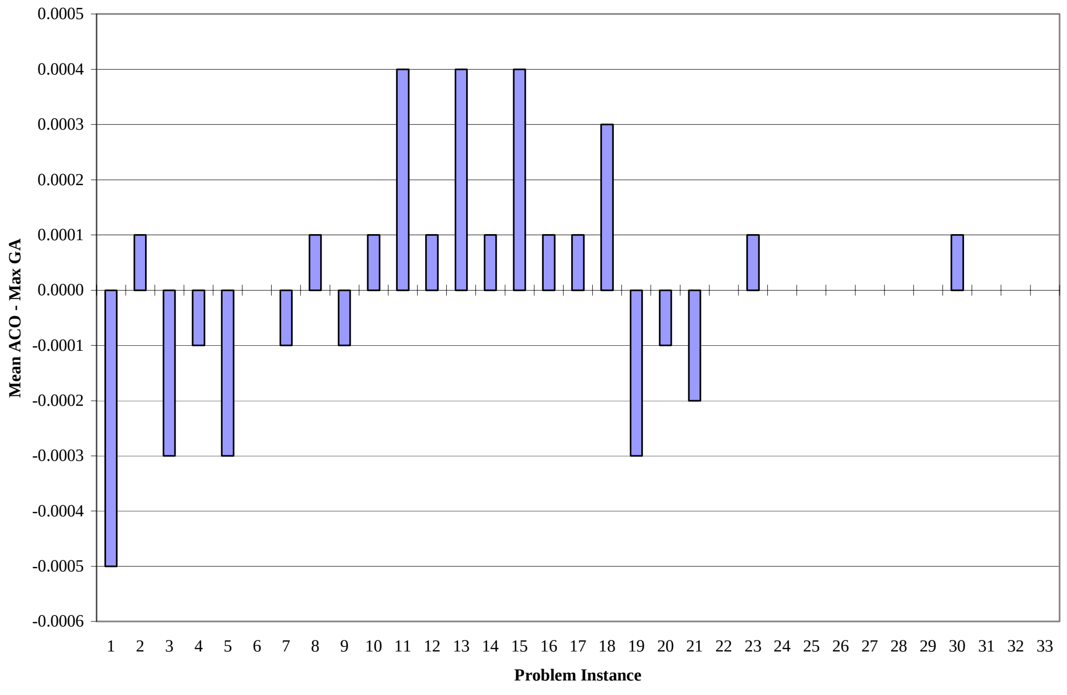
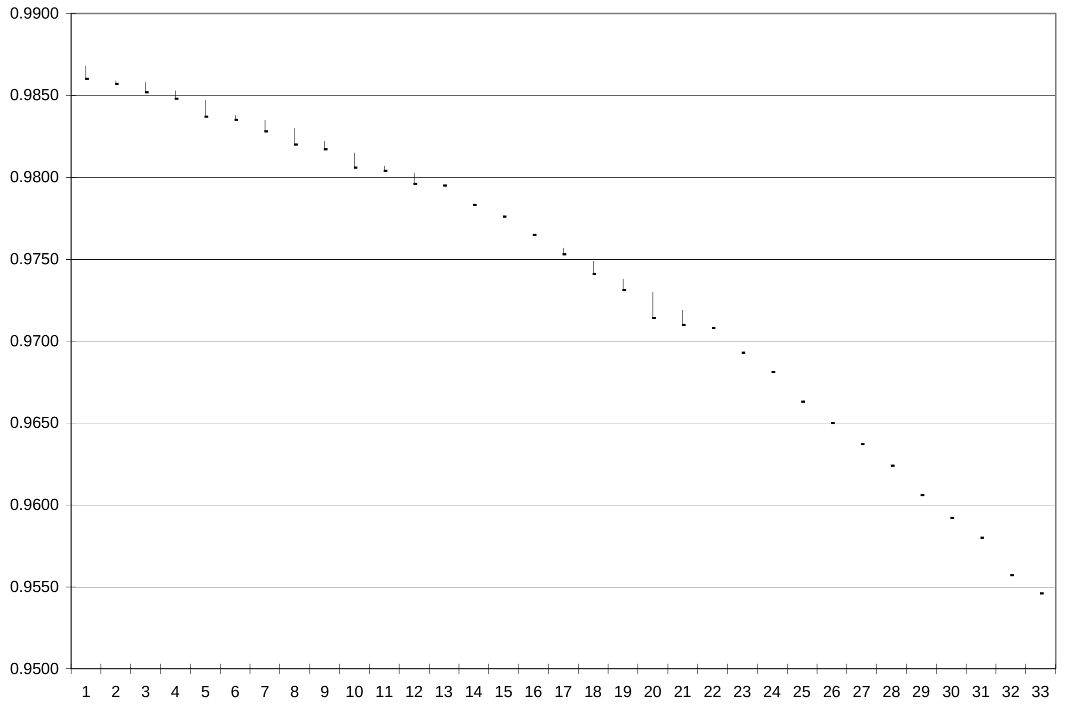
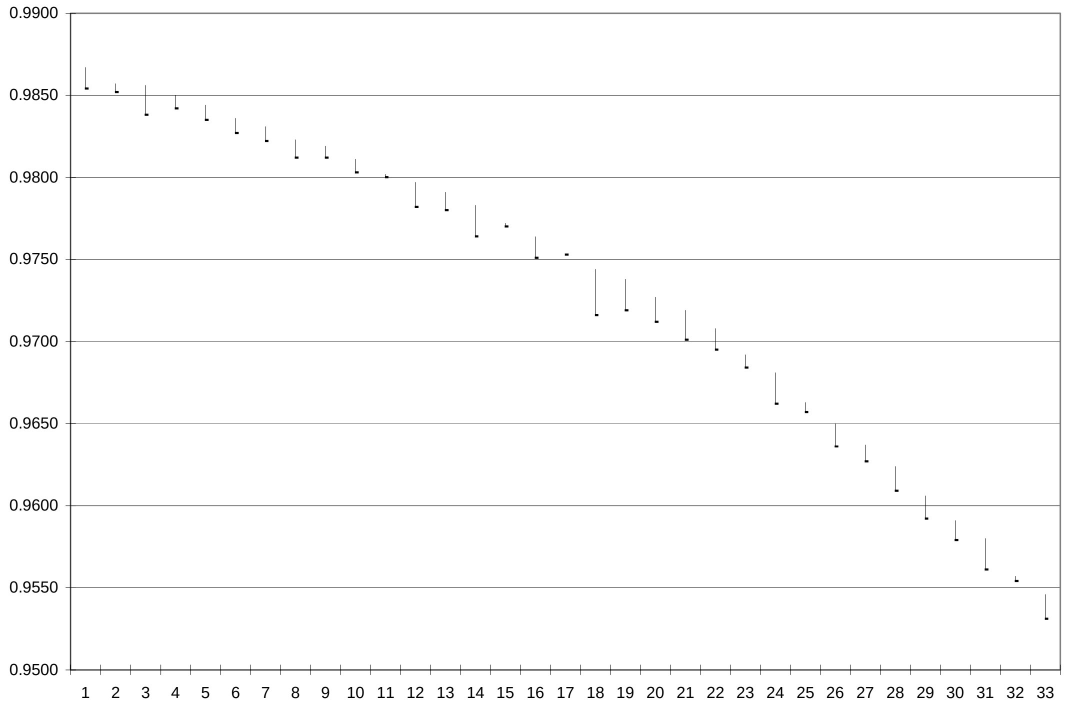


Figure 2. Comparison of Mean ACO with Best GA Performance.



(a) ACO



(b) GA

Figure 3. Range of Performance over 10 Seeds with Mean Shown as Horizontal Dash.

References

1. Bauer, A., B. Bullnheimer, R. F. Hartl, and C. Strauss, "Minimizing Total Tardiness on a Single Machine Using Ant Colony Optimization," *Central European Journal of Operations Research*, vol. 8, no. 2, 2000, 125-141.
2. Bellman, R. and S. Dreyfus, "Dynamic Programming and the Reliability of Multicomponent Devices," *Operations Research*, vol. 6, 1958, 200-206.
3. Bulfin, R. L. and C. Y. Liu, "Optimal Allocation of Redundant Components for Large Systems," *IEEE Transactions on Reliability*, vol. R-34, no. 3, 1985, 241-247.
4. Bullnheimer, B., R. F. Hartl, and C. Strauss, "Applying the Ant System to the Vehicle Routing Problem," in S. Voss, S. Martello, I. H. Osman, and C. Roucairol (eds.), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, 1999, Kluwer, 285-296.
5. Bullnheimer, B., R. F. Hartl, and C. Strauss, "An Improved Ant System Algorithm for the Vehicle Routing Problem," *Annals of Operations Research*, vol. 89, 1999, 319-328.
6. Chern, M. S., "On the Computational Complexity of Reliability Redundancy Allocation in a Series System," Operations Research Letters, vol. 11, 1992, 309-315.
7. Coit, D. W. and A. E. Smith, "Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm," *IEEE Transactions on Reliability*, vol. 45, no. 2, 1996, 254-260.
8. Coit, D. W. and A. E. Smith, "Considering Risk Profiles in Design Optimization for Series-Parallel Systems," *Proceedings of the 1997 Annual Reliability and Maintainability Symposium*, Philadelphia, January 1997, 271-277.
9. Colorni, A., M. Dorigo, V. Maniezzo, and M. Trubian, "Ant System for Job-Shop Scheduling," *Belgian Journal of Operations Research, Statistics and Computer Science* (*JORBEL*), vol. 34, no. 1, 1994, 39-53.
10. Costa, D. and A. Hertz, "Ants Can Colour Graphs," *Journal of the Operational Research Society*, vol. 48, 1997, 295-305.
11. den Besteb, M., T. Stützle, and M. Dorigo, "Ant Colony Optimization for the Total Weighted Tardiness Problem," *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature* (*PPSN* VI), LNCS 1917, Berlin, 2000, 611-620.
12. Di Caro, G. and M. Dorigo, "Ant Colonies for Adaptive Routing in Packet-Switched Communication Networks," *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature* (*PPSN* V), Amsterdam, The Netherlands, September 1998, 673-682.
13. Dorigo, M., *Optimization, Learning and Natural Algorithms*, Ph.D. Thesis, 1992, Politecnico di Milano, Italy.
14. Dorigo, M. and G. Di Caro, "The Ant Colony Optimization Meta-Heuristic," in D. Corne, M. Dorigo and F. Glover (eds.), *New Ideas in Optimization*, 1999, McGraw-Hill, 11-32.
15. Dorigo, M., G. Di Caro, and L. M. Gambardella, "Ant Algorithms for Discrete Optimization," *Artificial Life*, vol. 5, no. 2, 1999, 137-172.

16. Dorigo, M. and L. M. Gambardella, “Ant Colonies for the Travelling Salesman Problem,” *BioSystems*, vol. 43, 1997, 73-81.

17. Dorigo, M. and L. M. Gambardella, “Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997, 53-66.

18. Dorigo, M., V. Maniezzo, and A. Colorni, “Ant System: Optimization by a Colony of Cooperating Agents,” *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, vol. 26, no. 1, 1996, 29-41.

19. Fyffe, D. E., W. W. Hines, and N. K. Lee, “System Reliability Allocation And a Computational Algorithm,” *IEEE Transactions on Reliability*, vol. R-17, no. 2, 1968, 64-69.

20. Gambardella, L. M., E. Taillard, and G. Agazzi, “MACS-VRPTW A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows,” in D. Corne, M. Dorigo and F. Glover (eds.), *New Ideas in Optimization*, 1999, McGraw-Hill, 63-76.

21. Gen, M., K. Ida, Y. Tsujimura, and C. E. Kim, “Large-Scale 0-1 Fuzzy Goal Programming and Its Application to Reliability Optimization Problem,” *Computers and Industrial Engineering*, vol. 24, no. 4, 1993, 539-549.

22. Ghare, P. M. and R. E. Taylor, “Optimal Redundancy for Reliability in Series Systems,” *Operations Research*, vol. 17, 1969, 838-847.

23. Kuo, W. and V. R. Prasad, “An Annotated Overview of System-reliability Optimization,” *IEEE Transactions on Reliability*, vol. 49, no. 2, 2000, 176-187.

24. Levitin, G., A. Lisnianski, H. Ben-Haim, and D. Elmakis, “Redundancy Optimization for Series-Parallel Multi-State Systems,” *IEEE Transactions on Reliability*, vol. 47, no. 2, 1998, 165-172.

25. Maniezzo, V. and A. Colorni, “The Ant System Applied to the Quadratic Assignment Problem,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 5, 1999, 769-778.

26. Misra, K. B. and U. Sharma, “An Efficient Algorithm to Solve Integer-Programming Problems Arising in System-Reliability Design,” *IEEE Transactions on Reliability*, vol. 40, no. 1, 1991, 81-91.

27. Nakagawa, Y. and S. Miyazaki, “Surrogate Constraints Algorithm for Reliability Optimization Problems with Two Constraints,” *IEEE Transactions on Reliability*, vol. R-30, no. 2, 1981, 175-180.

28. Painton, L. and J. Campbell, “Genetic Algorithms in Optimization of System Reliability,” *IEEE Transactions on Reliability*, vol. 44, no. 2, 1995, 172-178.

29. Schoofs, L. and B. Naudts, “Ant Colonies are Good at Solving Constraint Satisfaction Problems,” *Proceedings of the 2000 Congress on Evolutionary Computation*, San Diego, CA, July 2000, 1190-1195.

30. Stuetzle, T. and M. Dorigo, “ACO Algorithms for the Quadratic Assignment Problem,” in D. Corne, M. Dorigo and F. Glover (eds.), *New Ideas in Optimization*, 1999, McGraw-Hill.

31. Tillman, F. A., C. L. Hwang, and W. Kuo, “Optimization Techniques for System Reliability

with Redundancy - A Review," *IEEE Transactions on Reliability*, vol. R-26, no. 3, 1977, 148-155.

32. Tillman, F. A., C. L. Hwang, and W. Kuo, "Determining Component Reliability and Redundancy for Optimum System Reliability," *IEEE Transactions on Reliability*, vol. R-26, no. 3, 1977, 162-165.

33. Wagner, I. A. and A. M. Bruckstein, "Hamiltonian(t)-An Ant Inspired Heuristic for Recognizing Hamiltonian Graphs," *Proceedings of the 1999 Congress on Evolutionary Computation*, Washington, D.C., July 1999, 1465-1469.