PENALTY GUIDED GENETIC SEARCH FOR RELIABILITY DESIGN OPTIMIZATION

David W. Coit and Alice E. Smith[1]

Department of Industrial Engineering
University of Pittsburgh
1048 Benedum Hall
Pittsburgh, PA 15261
aesmith@engrng.pitt.edu

Accepted to *Computers and Industrial Engineering*, Special Issue on Genetic Algorithms,

Volume 30, Number 4, 1996

[1] Corresponding author.

Penalty Guided Genetic Search for Reliability Design Optimization

Abstract

Reliability optimization has been studied in the literature for decades, usually using a mathematical programming approach. Because of these solution methodologies, restrictions on the type of allowable design have been made, however heuristic optimization approaches are free of such binding restrictions. One difficulty in applying heuristic approaches to reliability design is the highly constrained nature of the problems, both in terms of number of constraints and the difficulty of satisfying constraints. This paper presents a penalty guided genetic algorithm which efficiently and effectively searches over promising feasible and infeasible regions to identify a final, feasible optimal, or near optimal, solution. The penalty function is adaptive and responds to the search history. Results obtained on 33 test problems from the literature dominate previous solution techniques.

1. Introduction to the Redundancy Allocation Problem of Reliability Design

Development of new system designs involves the selection of components and configurations to satisfy detailed functional and performance specifications. For systems designed using off-the-shelf components, with known cost, reliability, weight and other attributes, system design can be formulated as a combinatorial optimization problem. The best known reliability design problem of this type is the redundancy allocation problem, where either system reliability is maximized or system cost is minimized. Both formulations generally involve system level constraints on allowable weight, power, cost, space and/or minimum system reliability level.

The overall system is partitioned into a specific number of subsystems, *s*, based on the system's required functions. For each required function, there are different component types available with varying costs, reliabilities, weights and other characteristics. It is often advantageous or even required to use components in parallel for a particular subsystem because of the required subsystem function (i.e., a minimum number of working components, *k*, is needed for the subsystem to function) or the need to increase system reliability. The redundancy

allocation problem is to select the best combination of components and levels of redundancy to collectively meet reliability and other constraints at a minimum cost, or alternatively, to maximize reliability given cost and other constraints. The most studied design configuration of the redundancy allocation problem is the series-parallel system, such as depicted in Figure 1. This is because many systems can be appropriately represented as series-parallel and because a series-parallel design can often serve as a bound for other types of system configurations.

INSERT FIGURE 1 HERE

If k_i represents the minimum number of components required for subsystem i to function and is specified for each subsystem, and there are constraints for maximum system cost and weight, the maximizing system reliability problem is:

$$\max \quad \prod_{i=1}^{s} R_i\left(\mathbf{x}_i \middle| k_i\right)$$

$$\text{subject to} \quad \sum_{i=1}^{s} C_i(\mathbf{x}_i) \le C$$

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

$$k_i \le \sum_{j=1}^{m_i} x_{ij} \le n_{\max,i} \;\forall\; i = 1,2,\ldots,s$$

$$x_{ij} \in (0, 1, 2, \ldots)$$

where,

C = cost constraint
W = weight constraint
s = number of subsystems
$\mathbf{x}_i$ = $(x_{i1}, x_{i2}, \ldots, x_{i,mi})$
x_{ij} = number of the j^{th} component used in subsystem i
n_i = total number of components used in subsystem i
= $x_{i1} + x_{i2} + \ldots + x_{i,mi}$
$n_{max,i}$ = maximum number of components used in subsystem i (specified)
$R_i(\mathbf{x}_i|k_i)$ = reliability of subsystem i, given k_i
$C_i(\mathbf{x}_i)$ = total cost of subsystem i
$W_i(\mathbf{x}_i)$ = total weight of subsystem i

Since the 1960s the series-parallel redundancy allocation has been studied with a variety of approaches; first by the seminal work using dynamic programming of Fyffe et al [10]. The dynamic programming approach suffered from inefficiencies and non-convergence, and was improved upon using integer programming techniques by other researchers including Ghare and Taylor [12], Bulfin and Liu [4] and Misra and Sharma [19]. An improved dynamic programming technique was developed by Nakagawa and Miyazaki [20]. A similar problem was formulated as a nonlinear optimization problem with component reliability and/or other constraints/objectives as continuous variables (see for example, Tillman et al. [16, 31, 33]). The mathematical programming approaches, while assuring optimality if convergence occurs, can only solve the redundancy allocation problem by imposing restrictions on the allowable configurations. These restrictions are setting *k* equal to 1 and/or allowing only one component type to be selected per subsystem, although replicates of this component may be used in parallel.

More recently, there has been interest in applying the newer heuristic techniques such as genetic algorithms (GA) and simulated annealing to reliability design. The use of genetic algorithms to solve reliability design problems is described by Painton and Campbell [22, 23], Ida et al. [17] and Coit and Smith [5-7]. Painton and Campbell solved two small design problems while considering the probabilistic aspects of reliability (i.e., component failure rate was regarded as a random variable rather than a fixed value). Ida et al. solved a series-parallel problem with multiple failure modes and compared this to their earlier research which solved the same problem using mathematical programming. Coit and Smith have pursued work related to this paper, which has centered on removing the restrictions on design required by mathematical programming, and solving the redundancy allocation problem to optimality using a dynamic, but not adaptive, penalty function.

This paper centers on solving highly constrained reliability design problems with genetic search by focusing on a penalty function approach. All realistic formulations of the redundancy allocation problem are constrained and often highly constrained. There are certain aspects of genetic algorithms, which are discussed below, which make handling of constraints particularly difficult. After describing the difficulties of using GAs for constrained problems, previous penalty approaches are discussed. The adaptive penalty approach developed in this research is then presented and its effectiveness is demonstrated on 33 test problems from the literature.

2. Genetic Algorithms and Constrained Optimization

Genetic algorithms produce new (child) solutions by recombining the encoded solutions of existing solutions (parents) from a population, and by mutating the child solutions. The central idea is that superior parent solutions will tend to produce superior child solutions, so that eventually an optimal solution is obtained. Mutation is used to prevent convergence to local optima. Because the GA works by recombining and altering solutions, maintaining feasibility is difficult for many problems. Even with two feasible parents, crossover can yield infeasible children. This especially arises in combinatorial optimization where the encoding is often times something other than the traditional bit string.

One solution to preserving feasibility of children is to increase the complexity of the crossover and mutation operators, so that they are guaranteed to produce feasible encodings [27], so called “repair operators.” However, many optimization problems, including the redundancy allocation problem, involve constraints for which it is not useful to modify a nearly-feasible solution to make it feasible. Repair operations will either disrupt schema (building blocks of superior solutions) excessively or incur undue computational overhead, or both. In fact, in many cases the problem of finding any feasible solution is itself NP-hard [11] including the problem of minimizing cost given a minimum allowable reliability and a maximum weight constraint. The most generic solution approach to such situations is using exterior penalty methods.

In general, a penalty function approach is as follows. Given an optimization problem,

$$\begin{aligned} \min \quad & z(\mathbf{x}) && \text{(P)} \\ \text{s.t.} \quad & \mathbf{x} \in A \\ & \mathbf{x} \in B \end{aligned}$$

where $\mathbf{x}$ is a vector of decision variables, the constraints "$\mathbf{x} \in A$" are relatively easy to satisfy, and the constraints "$\mathbf{x} \in B$" are relatively hard to satisfy, the problem can be reformulated as

$$\begin{aligned} \min \quad & z(\mathbf{x}) + p(d(\mathbf{x}, B)) && \text{(R)} \\ \text{s.t.} \quad & \mathbf{x} \in A \end{aligned}$$

where $d(\mathbf{x}, B)$ is a metric function describing the distance of the vector $\mathbf{x}$ from the region B, and $p(\cdot)$ is a penalty function such that $p(0) = 0$. This is an exterior penalty function defined such that if the function $p(\cdot)$ grows quickly enough outside of B, the optimal solution of (P) will also be optimal for (R). Furthermore, any optimal solution of (R) will provide an upper bound on the optimum for (P), and this bound will in general be tighter than that obtained by simply optimizing $\psi(\mathbf{x})$ over A. In the area of combinatorial optimization, Lagrangian relaxation [1, 9, 25] is one such penalty method.

Many penalty functions have been implemented in GA optimization with several major approaches emerging. The most fundamental is eliminating any infeasible solution from consideration immediately, the so called "death penalty". Another approach is based on the number of constraints violated, so that the penalty for a specific number of constraint violations is constant regardless of the magnitude of the violations. A third, more effective, approach uses a distance metric of the infeasible solution from the feasible region [2, 13, 15, 21, 26].

A variation of distance based penalty functions also incorporates a dynamic aspect which increases the severity of the penalty for a given distance as the search progresses. This is most commonly done by increasing the penalty with each generation. This allows more infeasible solutions with attractive schema early in the search, while eventually moving the final solution to the feasible region. Uses of this approach include Joines and Houck [18], Petridis and Kazarlis

[24] and Coit and Smith [5]. While these penalty functions are dynamic, they do not respond to the search history except to monotonically increase with generation number.

A few researchers have proposed making use of search information in an adaptive way to formulate robust and effective penalty functions. Siedlecki and Sklansky's [27] method is restricted to binary-string encodings with a single constraint, and involves considerable computational overhead. Bean and Hadj-Alouane [3, 14] use penalty functions which are altered based on the feasibility or infeasibility of the best, penalized solution during recent generations. Their penalty function allows both increase and decrease of the penalty during the search and was demonstrated on multiple choice integer programming problems with one constraint. Smith and Tate [28, 30] used both the number of generations and the characteristics of the best found solution so far in their penalty function. Their work was confined to a single, discrete constraint, however it provided the basis for the approach used in this paper. More generic information about GAs and penalty functions, and a general form of an adaptive penalty approach can be found in Coit et al. [8].

3. Adaptive Penalty for Combinatorial Reliability Design

The penalty function used here introduces the notion of a "near-feasibility threshold" (*NFT*) for each constraint. Exterior penalty functions are usually characterized as being nondecreasing functions of the distance of a solution from the feasible region. Conceptually, the *NFT* is the threshold distance from the feasible region which is considered as being close to feasibility. Often this can be selected based on the practical implications of a constraint violation. The penalty function will encourage the GA to explore the feasible region and the *NFT*-infeasible region, and discourage, but permit, search further into the infeasible region.

The penalty function used to solve the redundancy allocation problem where reliability was maximized is presented as Equation 1. The penalty learns to adapt itself based on the severity of the constraints of a particular problem instance. An adaptive term, (V_{all} - V_{feas}) in Equation 1 calculates the difference between the non-penalized solution value of the best solution yet found (which will probably be infeasible) and the value of the best feasible solution yet found.

$$V_{ip} = V_i - \left(\left(\frac{\Delta w_i}{NFT_w} \right)^{\kappa} + \left(\frac{\Delta c_i}{NFT_c} \right)^{\kappa} \right) (V_{all} - V_{feas}) \quad (1)$$

V_{ip} is the penalized objective function value of solution i, V_i is the unpenalized objective function value for solution i, V_{all} denotes the unpenalized value of the best solution yet found, and V_{feas} denotes the value of the best feasible solution yet found. The exponent κ is a pre-set severity parameter. Previous work had demonstrated substantial robustness to values of κ [8, 28] and it was set to 2 for the work presented throughout this paper. NFT_c and NFT_w are the "Near-Feasible Thresholds" for the cost and weight constraints respectively, and Δw_i and Δc_i represent the magnitude of any constraint violations for the i^{th} solution vector.

There are two aspects of Equation 1 which need more explanation. First, if the objective function of the best feasible solution in the population is better than the unpenalized objective function of the best infeasible solution, then the (V_{all} - V_{feas}) term is zero. Any subsequent infeasible solution with an unpenalized objective function inferior to the best feasible solution will not have a penalty imposed. Conceptually, this would often imply the best unpenalized solution is likely to be in the interior of the feasible region (i.e., no tight constraints). However, in practice for coherent systems, this is an unlikely occurrence. When maximizing system reliability, when cost and/or weight constraints are violated by adding additional redundant components, it is unlikely that the system reliability will be inferior to a feasible solutions.

A second consideration is when, early in the search, a best infeasible solution is so dramatically infeasible that V_{all} is much different from V_{feas} and the penalty may then be too severe for the remainder of the search. In practice this has not been observed, primarily because of initial population selection strategies which largely preclude the possibility of selecting solution vectors sufficiently far from the feasible region for this to happen. If in fact, this becomes a problem for a specific problem instance, then it can be alleviated by determining new values for V_{all} and V_{feas} after an initial "burn-in" period.

Determination of suitable values for *NFT* requires further discussion. Given a specific problem and constraint set (*C*, *W*), the extent to which the infeasible region should be thoroughly explored may not be intuitive (although *NFT* can often be selected based on the practical implications of a constraint violation). Effective values of *NFT* could be found through iterative experimentation for any particular redundancy allocation problem, however, this is clearly undesirable. This paper fully investigates the sensitivity of the adaptive penalty approach to *NFT* by using different rules including a dynamic *NFT* which monotonically decreases during the search.

The dynamic *NFT* is defined as follows,

$$NFT = \frac{NFT_o}{1+\lambda g} \tag{2}$$

where NFT_o is an upper bound or starting point for *NFT*, g is the generation number, and λ is a constant which assures that the entire region between NFT_o and zero (strict feasibility) is searched. The only restrictions on selecting λ is that the *NFT* not approach zero either too quickly or too slowly.

4. Genetic Algorithm for Redundancy Allocation

The solution encoding, evolution parameters and genetic operators are as described in Coit and Smith [5], an earlier work with the redundancy allocation problem. However, this paper uses a more robust, adaptive penalty function which reduces the number of user set and problem specific parameters. The fundamental operations of the GA are summarized here. Each possible solution to the redundancy allocation problem is a collection of n_i components in parallel ($k_i \le n_i \le n_{max,i}$) for s different subsystems. The n_i components can be chosen in any combination from among m_i available components. The m_i components are indexed in descending order according to their reliability so that 1 represents the most reliable. The solution encoding is an integer vector representation with $\Sigma\, n_{max,i}$ positions. Each of the s subsystems are represented by $n_{max,i}$ positions with each component listed according to their reliability index. An index of $m_i + 1$ is assigned to a position where an additional component

was not used (i.e., $n_i < n_{max,i}$). The subsystem representations are then concatenated to complete the vector representation. As an example, consider a system with $s = 2$, $m_1 = 4$, $m_2 = 6$, and $n_{max,i}$ predetermined to be 6 for both i. The following example,

$$\mathbf{v} = (1\ 2\ 2\ 5\ 5\ 5 \mid 3\ 3\ 3\ 5\ 5\ 7)$$

represents a prospective solution with one of the most reliable components and two of the second most reliable component used in parallel for the first subsystem; and three of the third most reliable and two of the fifth most reliable component used in parallel for the second subsystem.

The initial population was determined by randomly selecting s integers between k_i and $n_{max,i}$ to represent the number of components in parallel (n_i) for each solution subsystem. Then, n_i components were randomly and uniformly selected from among the m_i available components, assuming an adequate supply available of each type. The components were then sequenced in accordance with their reliability. Previous experimentation [5] indicated that a population size of 40 was appropriate for the test problems studied.

The crossover operator used was uniform crossover where common components of the parents were retained in the child and non-common components were selected with equal probability from either parent. Parents were probabilistically selected based on a quadratic relationship to their ordinal ranking as in Tate and Smith [29, 30]. This method is computationally efficient because the ordinal rank is the only information used in parent selection. Also, it preserves selection pressure, which is a hallmark of GA, but avoids premature convergence to local optima when one solution is far better than any others, as might occur using a selection method based on objective function value rather than rank. For a solution vector chosen for mutation, each integer value is changed with probability equal to a pre-selected mutation rate. If selected to be mutated, it is changed to an index of $m_i + 1$ with 50 % probability (to no component) and to a randomly chosen component, from among the m_i choices, with 50 % probability. All mutated solutions are maintained within the population for at least one generation to assure that they have the opportunity to breed.

After crossover, the p best solutions from the current generation and the new child solutions were combined to form the next generation. Mutation was then performed after deleting inferior solutions from the population. The best solution within the population was never chosen for mutation to assure that it was never altered and to improve the rate of convergence (an elitist strategy).

5. Test Problems and Results

5.1. Test Problems

The test problems studied are the original problem posed by Fyffe, Hines and Lee [10] and the 33 problem variations from Nakagawa and Miyazaki [20]. The objective is to maximize reliability for a system with 14 subsystems and three or four component choices for each subsystem, where $k = 1$ for all subsystems. The size of the search space is larger than 7.6×10^{33} (with component mixing). For each component alternative, there is a specified reliability, cost and weight. In the original problem, there is a cost constraint of 130 and the weight constraint is 170. In the 33 problem variations, the cost constraint is maintained at 130 and the weight constraint decreases incrementally from 191 to 159, which increases the severity of the constraint.

Fyffe, Hines and Lee [10] used a dynamic programming approach with a Lagrangian multiplier to accommodate the weight constraint within the objective function. However, Nakagawa and Miyazaki [20] showed that Lagrangian multipliers are often inefficient and they used a surrogate constraint approach combining the cost and weight constraints into a single constraint, which must be iteratively altered with different surrogate multipliers. In both papers, the approaches required that only identical components could be placed in redundancy. Therefore, the optimal solution found pertains to a restricted search space, and a better solution might be identified by allowing different, yet functionally similar, component types to reside in parallel.

5.2. Computational Results

The 33 problems variations were solved with different *NFT* criteria including three levels of constant *NFT* (5%, 3% and 1% of each constraint), a dynamic *NFT* and a GA that allowed only feasible solutions (the "death penalty"). This allowed comparisons between a commonly used approach (discarding infeasible solutions) with several possible versions of the adaptive penalty function. The constant *NFT* versions had only an adaptive aspect where the dynamic *NFT* version used both an adaptive term and a dynamic term which increased the penalty with the number of generations, another common penalty formulation. For the decreasing *NFT* implementation in this paper, λ was set to 0.04 for both cost and weight, and NFT_{co} was set to 100 and NFT_{wo} was set to $W/1.3$. Ten different random number seeds were used for each of the 33 problem variations and *NFT* criteria, resulting in a total of 1650 GA runs.

Figure 2 shows the system reliability of the best feasible solution of the ten runs for each of the 33 problems as a ratio to the system reliability of the original solutions of Nakagawa and Miyazaki. The Nakagawa and Miyazaki solutions were used as a reference, however, it should be noted that (1) their algorithm did not converge to a feasible solution in several instances and the reference value used is not actually a viable solution, and (2) a strict comparison of their results and those presented here is unfair because different solution spaces were considered.

It can be seen in Figure 2 that as the problem grows more constrained the dynamic *NFT* becomes more dominant in finding good feasible solutions. Of the constant *NFT*s, the 1% is clearly inferior and produces feasible, but suboptimal, solutions. Apparently a small *NFT* promotes exploration of the feasible region but does not properly encourage exploration of the boundary region between feasibility and infeasibility. The strategy of allowing only feasible solutions is also poor by finding even more suboptimal, though feasible, solutions. For constrained problems, the optimum solution will lie on the boundary of at least one of the constraints. This serves as further empirical evidence that the preferred approach to find the optimal solution is to approach it through the infeasible region.

Although the dynamic *NFT* did not strictly dominate all other alternatives, comparative statistical analysis confirmed its general superiority. A paired t-test comparing the dynamic *NFT*

with each of the other strategies (constant *NFT* and death penalty) yielded t values ranging from 4.13 (5% of constraint) to 13.88 (death penalty), with corresponding p values of 0.0002 to <0.00005. It is also important to note that the superiority of the dynamic *NFT* increases as the severity of the constraints increases, i.e. from problem 1 to 33.

INSERT FIGURE 2 HERE

Since GA is a stochastic search algorithm, another aspect of comparing solutions is the sensitivity of the optimization procedure to the random number seed used. A robust optimization procedure will exhibit low sensitivity to random number seed as evidenced by the variance of the best solution found during the search. Figure 3 shows the standard deviation of the best feasible solution to each problem for each penalty strategy. It can be seen that as the problem grows more constrained, the difficulty in finding a good, feasible solution increases and the search becomes more sensitive to initial seed. The dynamic *NFT* has low standard deviation regardless of the extent of the constraints (i.e., size of the feasible region) where all the other strategies incur greater standard deviation as the problem becomes harder. Especially the strategies of the 1% static *NFT* and the death penalty are susceptible to greater variance in the final best feasible solution. The robustness of the dynamic *NFT* to degree of problem constraint and to random number seed is particularly encouraging.

INSERT FIGURE 3 HERE

Numerical summary results are presented in Table 1 and more detailed results are presented in the Appendix. For each table, the results are pooled from the 33 different problems, that is 330 different GA runs were made for each penalty strategy considered. The results show for each of the five *NFT* alternatives, the proportion of trials where the GA's best solution was feasible. Additionally, the table presents the system reliability of the best of the 10 runs and the mean of the 10 runs averaged over all 33 problems. The results from this problem indicate that use of the adaptive penalty function (with any of the *NFT* criteria) is preferable to considering only feasible solutions. Another important result is the comparisons of different *NFT* criteria. If the *NFT* was large (5%), the GA often found good feasible solutions but ultimately converged to

an infeasible solution in greater than 98% of the trials. Conversely, if the *NFT* was small (1%), the GA converged to a feasible solution in all cases, but the solution quality was poor. An intermediate *NFT* value (3%) was able to balance both the benefits and deficiencies but is still inferior to the dynamic *NFT* considering all aspects.

INSERT TABLE 1 HERE

The configuration of the best solution and its reliability, cost and weight for each of the 33 problems are shown in the Appendix. Note that most of the solutions involve mixing of components, which indicates that if mixing is an option, better solutions can be identified by not restricting the search space to a single component type per subsystem.

6. Conclusions

Adaptive penalty-guided genetic search is promising as an optimization method for solving reliability design. The investigations presented here for the redundancy allocation problem show that this approach can be powerful and robust for problems with large search spaces and difficult-to-satisfy constraints. For the dynamic *NFT* penalty function, the quality of the solutions found does not seem to be particularly sensitive to the precise penalty parameters used, the random number seed, the degree of constraint, or the particular problem instance, so that no extensive tuning of the penalty function is necessary, especially when using the dynamic *NFT* strategy.

The dynamic *NFT* was clearly superior to the strategy of allowing only feasible solutions in the population and to the static *NFT*s. Superiority was in terms of both final feasible solution quality, variance of final feasible solution quality and successful convergence to a final best solution that was feasible. The superiority of the dynamic *NFT* increased as the problem became harder (more constrained). Additionally, by having the *NFT* decrease throughout the infeasible region during the search, it is not necessary to pre-select a specific *NFT*.

References

[1] M. Avriel, 1976, *Nonlinear Programming: Analysis and Methods*, Prentice Hall, Englewood Cliffs, NJ.

[2] T. Baeck and S. Khuri, 1994, *An Evolutionary Heuristic for the Maximum Independent Set Problem*, **Proceedings of the First IEEE Conference on Evolutionary Computation**, 531-535.

[3] J. C. Bean and A. B. Hadj-Alouane, 1993, *A Dual Genetic Algorithm for Bounded Integer Programs*, University of Michigan Technical Report, to be published in **R.A.I.R.O.-R.O.**

[4] R. L. Bulfin and C. Y. Liu, 1985, *Optimal Allocation of Redundant Components for Large Systems*, **IEEE Transactions on Reliability R-34**, 241-247.

[5] D. W. Coit and A. E. Smith, 1996, *Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm*, **IEEE Transactions on Reliability 45 no. 2**, in print.

[6] D. W. Coit and A. E. Smith, 1994, *Use of a Genetic Algorithm to Optimize a Combinatorial Reliability Design Problem*, **Proceedings of the Third Industrial Engineering Research Conference**, 467-472.

[7] D. W. Coit and A. E. Smith, 1995, *Optimization Approaches to the Redundancy Allocation Problem for Series-Parallel Systems*, **Proceedings of the Fourth Industrial Engineering Research Conference**, 342-349.

[8] D. W. Coit, A. E. Smith and D. M. Tate, 1996, *Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problems*, **INFORMS Journal on Computing 8, no. 2**, in print.

[9] M. L. Fisher, 1981, *The Lagrangian Relaxation Method for Solving Integer Programming Problems*, **Management Science 27**, 1-18.

[10] D. E. Fyffe, W. W. Hines and N. K. Lee, 1968, *System Reliability Allocation and a Computational Algorithm*, **IEEE Transactions on Reliability R-17**, 64-69.

[11] M. R. Garey and D. S. Johnson, 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco.

[12] P. M. Ghare and R. E. Taylor, 1969, *Optimal Redundancy for Reliability in Series System*, **Operations Research 17**, 838-847.

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

[14] A. B. Hadj-Alouane and J. C. Bean, 1993, *A Genetic Algorithm for the Multiple-Choice Integer Program*, University of Michigan Technical Report 92-50, to be published in **Operations Research**.

[15] W-C. Huang, C-Y. Kao and J-T. Horng, 1994, *A Genetic Algorithm Approach for Set Covering Problem*, **Proceedings of the First IEEE Conference on Evolutionary Computation**, 569-573.

[16] C. L. Hwang, F. A. Tillman and W. Kuo, 1979, *Reliability Optimization by Generalized Lagrangian-Function and Reduced-Gradient Methods*, **IEEE Transactions on Reliability R-28**, 316-319.

[17] K. Ida, M. Gen and T. Yokota, 1994, *System Reliability Optimization with Several Failure Modes by Genetic Algorithm*, **Proceedings of 16th International Conference on Computers and Industrial Engineering**, 349-352.

[18] J. A. Joines and C. R. Houck, 1994, *On the Use of Non-stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with GA's*, **Proceedings of the First IEEE Conference on Evolutionary Computation**, 579-584.

[19] K. B. Misra and U. Sharma, 1991, *An Efficient Algorithm to Solve Integer Programming Problems Arising in System Reliability Design*, **IEEE Transactions on Reliability R-40**, 81-91.

[20] Y. Nakagawa and S. Miyazaki, 1981, *Surrogate Constraints Algorithm for Reliability Optimization Problems With Two Constraints*, **IEEE Transactions on Reliability R-30,** 175-180.

[21] A. L. Olsen, 1994, *Penalty Functions and the Knapsack Problem*, **Proceedings of the First IEEE Conference on Evolutionary Computation**, 554-558.

[22] L. Painton and J. Campbell, 1994, *Identification of Components to Optimize Improvements in System Reliability*, **Proceedings of the SRA PSAM-II Conference on System-based Methods for the Design and Operation of Technological Systems and Processes**, 10-15 - 10-20.

[23] L. Painton and J. Campbell, 1995, *Genetic Algorithms in Optimization of System Reliability*, **IEEE Transactions on Reliability 44**, 172-178.

[24] V. Petridis and S. Kazarlis, 1994, *Varying Quality Function in Genetic Algorithms and the Cutting Problem*, **Proceedings of the First IEEE Conference on Evolutionary Computation**, 166-169.

[25] C. R. Reeves, 1993, *Modern Heuristic Techniques for Combinatorial Problems*, John Wiley & Sons, New York, NY.

[26] J. T. Richardson, M. R. Palmer, G. Liepins and M. Hilliard, 1989, *Some Guidelines for Genetic Algorithms with Penalty Functions*, **Proceedings of the Third International Conference on Genetic Algorithms**, 191-197.

[27] W. Siedlecki and J. Sklansky, 1989, *Constrained Genetic Optimization via Dynamic Reward-Penalty Balancing and Its Use in Pattern Recognition*, **Proceedings of the Third International Conference on Genetic Algorithms**, 141-150.

[28] A. E. Smith and D. M. Tate, 1993, *Genetic Optimization Using a Penalty Function*, **Proceedings of the Fifth International Conference on Genetic Algorithms**, 499-505.

[29] D. M. Tate and A. E. Smith, 1995, *A Genetic Approach to the Quadratic Assignment Problem*, **Computers and Operations Research 22**, 73-83.

[30] D. M. Tate and A. E. Smith, 1995, *Unequal Area Facility Layout Using Genetic Search,* **IIE Transactions 27**, 465-472.

[31] F. A. Tillman, C. L. Hwang and W. Kuo, 1977, *Optimization Techniques for System Reliability with Redundancy - A Review*, **IEEE Transactions on Reliability R-26**, 148-155.

[32] F. A. Tillman, C. L. Hwang and W. Kuo, 1977, *Determining Component Reliability and Redundancy for Optimum System Reliability*, **IEEE Transactions on Reliability R-26**, 162-165.

Table 1. Feasibility and Performance Comparison for the 33 Reliability Optimization Problems Over 10 Runs of Each.

Comparison	Death Penalty	5% *NFT*	3% *NFT*	1% *NFT*	Dynamic *NFT*
% Feasible	100.00%	1.21%	80.00%	100.00%	100.00%
Best Solution	0.97096	0.97337	0.97302	0.97180	0.97366
Average Solution	0.96894	0.97239	0.97167	0.96956	0.97288
Coeff. of Variation (%)	0.14933	0.08218	0.11305	0.15847	0.06573

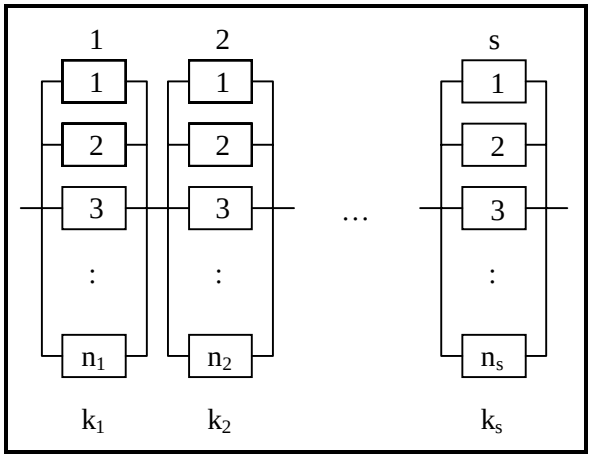

Figure 1. Series-Parallel System Configuration.

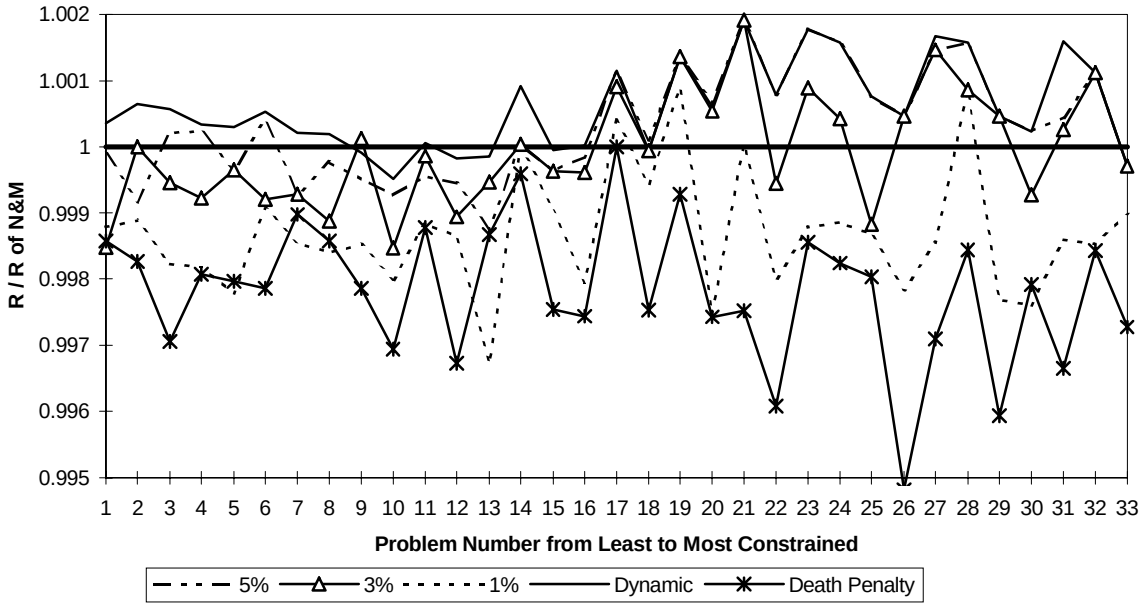

Figure 2. Reliability of Best Solution of Ten Runs for Each Problem Relative to N & M [20] Solution.

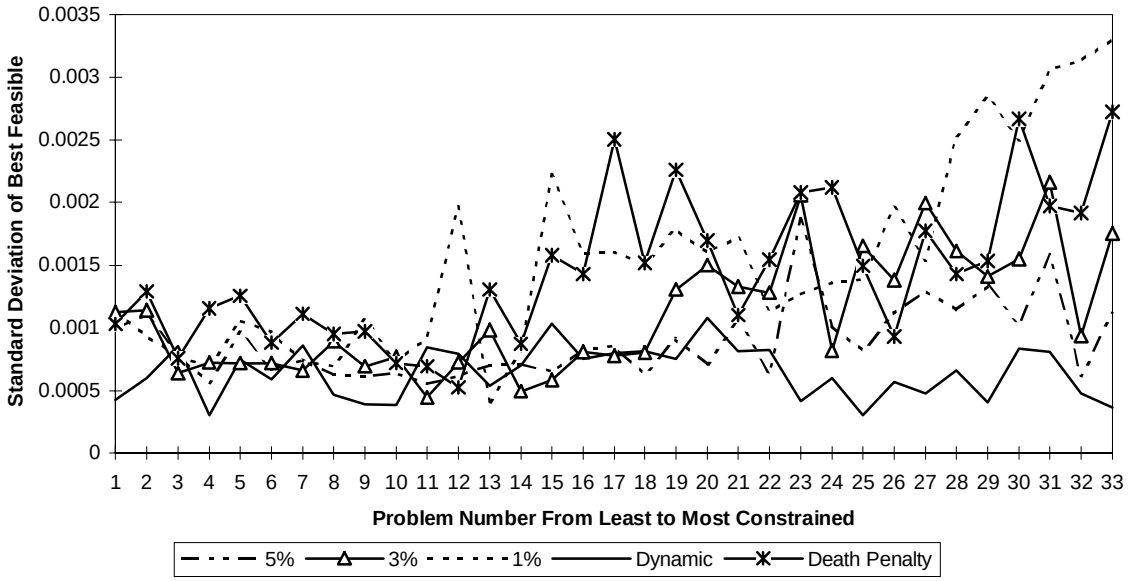

Figure 3. Standard Deviation of Best Feasible Over Ten Runs for Each Problem.

Appendix

Configuration and Reliability, Cost and Weight of the Best Solution to Each Problem

no.	W[1]	reliability	cost	weight	penalty[2]	solution vector[3]
1	191	.98675	130	191	D	333,11,111,2222,333,22,333,3333,23,122,333,4444,12,12
2	190	.98603	129	190	D	333,11,111,222,333,22,333,3333,13,111,333,4444,11,12
3	189	.98556	130	189	D	333,11,111,2222,333,22,333,3333,13,122,333,4444,12,22
4	188	.98503	130	188	D	333,11,111,2222,333,22,333,3333,13,112,33,4444,11,12
5	187	.98429	129	187	D	333,11,111,2222,333,22,333,3333,13,122,33,4444,11,12
6	186	.98362	128	186	D	333,11,111,222,333,22,333,3333,33,122,333,4444,11,12
7	185	.98311	130	185	D	333,11,111,2222,333,22,333,3333,33,122,33,4444,11,12
8	184	.98239	128	184	D	333,11,111,222,333,22,333,3333,33,112,333,4444,11,22
9	183	.9819	130	183	3	333,11,111,2222,333,22,333,3333,33,112,33,4444,11,22
10	182	.98102	126	182	D	333,11,111,222,333,22,333,3333,33,112,33,4444,11,12
11	181	.98006	128	181	D	333,11,111,222,333,22,333,113,33,122,33,4444,11,12
12	180	.97942	129	180	D	333,11,111,222,333,22,333,113,33,112,23,4444,11,22
13	179	.97906	125	179	D	333,11,111,222,333,22,333,3333,33,112,33,4444,11,22
14	178	.97810	127	178	D	333,11,111,222,333,22,333,113,33,122,33,4444,11,22
15	177	.97715	125	177	D	333,11,111,222,333,22,333,133,33,112,33,4444,11,22
16	176	.97642	124	176	D	333,11,111,222,333,22,333,133,33,122,33,4444,11,22
17	175	.97552	122	175	D,5	333,11,111,222,333,22,11,3333,33,122,33,4444,11,22
18	174	.97435	123	174	5	333,11,111,222,333,22,11,133,33,112,33,4444,11,22
19	173	.97362	122	173	D,3,5	333,11,111,222,333,22,11,133,33,122,33,4444,11,22
20	172	.97266	120	172	5	333,11,111,222,333,22,13,3333,33,222,33,4444,11,22
21	171	.97186	121	171	D,3,5	333,11,111,222,333,22,13,133,33,122,33,4444,11,22
22	170	.97076	120	170	D,5	333,11,111,222,333,22,13,133,33,222,33,4444,11,22
23	169	.96922	120	169	D,5	333,11,111,222,333,22,33,133,33,122,33,4444,11,22
24	168	.96813	119	168	D,5	333,11,111,222,333,22,33,133,33,222,33,4444,11,22
25	167	.96634	118	167	D,5	333,11,111,222,33,22,13,133,33,222,33,4444,11,22
26	166	.96504	116	166	D,3,5	333,11,11,222,333,22,13,133,33,222,33,4444,11,22
27	165	.96371	117	165	D	333,11,111,222,33,22,33,133,33,222,33,4444,11,22
28	164	.96242	115	164	D,5	333,11,11,222,333,22,33,133,33,222,33,4444,11,22
29	163	.96064	114	163	D,5	333,11,11,222,33,22,13,133,33,222,33,4444,11,22
30	162	.95912	114	162	D,5	333,11,11,222,33,22,33,133,33,122,33,4444,11,22
31	161	.95803	113	161	D	333,11,11,222,33,22,33,133,33,222,33,4444,11,22
32	160	.95567	114	160	D,5	333,11,11,222,33,22,33,133,33,222,33,144,11,22
33	159	.95432	110	159	D,5	333,11,11,222,333,22,33,133,1,222,33,4444,11,22

NOTES: (1) C = 130 for all problems

(2) D ≡ dynamic *NFT*, 3 ≡ *NFT* 3% of constraint value, 5 ≡ *NFT* 5% of constraint value

(3) Solution vector lists component choices sequentially for each of the 14 subsystems (separated by a comma). Components indexed from most to least reliable.