SOLVING THE REDUNDANCY ALLOCATION PROBLEM USING A COMBINED NEURAL NETWORK / GENETIC ALGORITHM APPROACH

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

Department of Industrial Engineering

1031 Benedum Hall

University of Pittsburgh

Pittsburgh, PA 15261

412-624-5045

412-624-9831 (fax)

aesmith@engrng.pitt.edu

Statement of Scope and Purpose

This paper presents a novel hybrid intelligent system which uses a genetic algorithm with a neural network to optimize a combinatorial engineering design problem considering a system reliability constraint. The genetic algorithm searches among candidate designs of the system configuration to identify optimal levels of component redundancy. The neural network provides the objective function value of each candidate design by estimating the system reliability. The resulting hybrid approach not only finds optimal solutions, but does so with fewer computational resources devoted to evaluation of the objective function.

---

[*] Corresponding author.

SOLVING THE REDUNDANCY ALLOCATION PROBLEM USING A COMBINED NEURAL NETWORK / GENETIC ALGORITHM APPROACH

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

Department of Industrial Engineering

1031 Benedum Hall

University of Pittsburgh

Pittsburgh, PA 15261

412-624-5045

412-624-9831 (fax)

aesmith@engrng.pitt.edu

Abstract

This paper optimizes a well known NP-hard combinatorial problem - redundancy allocation - using a combined neural network and genetic algorithm (GA) approach. The GA searches for the minimum cost solution by selecting the appropriate components for a series-parallel system, given a minimum system reliability constraint. A neural network is used to estimate the system reliability value during search. This approach is an example of a computationally efficient method to apply GA optimization to problems for which repeated calculation of the objective function is impractical.

---

[*] Corresponding author.

SOLVING THE REDUNDANCY ALLOCATION PROBLEM USING A COMBINED NEURAL NETWORK / GENETIC ALGORITHM APPROACH

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

Department of Industrial Engineering

1031 Benedum Hall

University of Pittsburgh

Pittsburgh, PA 15261

412-624-5045

412-624-9831 (fax)

aesmith@engrng.pitt.edu

1. Introduction

This paper presents an optimization approach using a genetic algorithm (GA) to identify the choice of components and design configuration in a series-parallel system which must meet a lower bound on system reliability. This redundancy allocation problem is a combinatorial optimization problem where the reliability goal is achieved by discrete choices made from available parts. Since GA (and other forms of evolutionary optimization) require numerous objective function evaluations to calculate fitness during evolution, a problem where the evaluation of the objective function is computationally time consuming may seem ill-fitted. Complicated reliability design problems are such cases where the determination of the reliability of a given solution (i.e., system configuration) can require considerable algorithmic calculations or even Monte Carlo simulations. The approach in this paper to this barrier to effective GA search is to develop a neural network approximation of system reliability.

Neural networks have rarely been used as a function evaluator to facilitate GA search. In fact, research by Pipe, Fogarty and Winfield [1] represents the only other known instance of neural networks being used for this purpose. The authors applied the temporal differences

---

[*] Corresponding author.

learning algorithm to a radial basis function neural network to learn movements through a simple maze. Learning takes place in a critic mode, and the resulting network outputs the value of each position in the maze relative to escaping the maze. This is used by their GA to select the next move through the maze. The approach describe below is considerably different both in motivation and in procedure.

1.1 The Reliability Design Problem

Design of a hardware system involves numerous discrete choices among available component types based on cost, reliability, performance, weight, etc. If the design objective is to minimize cost for a certain reliability requirement then a strategy is required to identify the optimal combination of components and/or design configuration. When there are many functionally similar components to choose from, it becomes difficult to identify the optimal solution, particularly when redundancy can be used to improve reliability. For the purposes of this paper, redundancy is defined as the use of functionally similar (but not necessarily identical) components together in a design such that if one component fails, the redundant part will be available to perform the required function without the system experiencing a failure.

Two particular reliability design problems where it is difficult to compute system reliability are network systems optimization and optimization of systems employing multiple k-out-of-n redundancies with dissimilar components in parallel. For all-terminal network reliability problems, determination of system reliability requires the consideration of all possible paths (which are not necessarily independent) which connect each pair of terminals (network nodes). For even small networks, this proves to be computationally impractical. For systems with multiple k-out-of-n redundancies, which is the case considered in this paper, computation of subsystem reliability also becomes a difficult combinatorial problem as k becomes large. Optimization of these systems has been proven to be NP-hard by Chern [2].

k-out-of-n redundancy is defined as a series of n parallel components where any k of the n are required to be operating (good) for the system to avoid a failure. The total number of

components in parallel, n_i, for each subsystem is a variable which can assume any integer value greater than or equal to k_i (where i is the subsystem number). For this paper, it is assumed that k_i has been specified while n_i remains a variable to be determined as part of the optimization process. Figure 1 shows a typical series-parallel system.

The problem for a series-parallel system can be stated formally as below:

$$\text{min} \qquad \sum_{i=1}^{s} C_i(\mathbf{x}_i)$$

$$\text{subject to} \qquad \prod_{i=1}^{s} R_i\left(\mathbf{x}_i|k_i\right) \geq R$$

$$\sum_{j=1}^{m_i} x_{ij} \geq k_i \ \forall \ i$$

where
C_i = cost of ith subsystem
R_i = reliability of ith subsystem
R = reliability constraint

$$\mathbf{x_i} = (x_{i1}, x_{i2}, ..., x_{i,m_i}), \ n_i = \sum_{j=1}^{m_i} x_{ij}$$

x_{ij} = number of the jth available component used in subsystem i
$x_{ij} \in (0, 1, 2, ...)$

If no limits or upper bounds are established for n, then the total number of possible design configurations is unbounded. However, it is practical to establish an upper bound on n (n_{max}). Once upper limits have been established, then it becomes possible to compute the total number of possibilities by treating the selection of parts for each function as an occupancy problem [3]. If the number of functionally similar components for each subsystem i is denoted m_i, the number of unique system representations is given by the following equation.

$$N = \prod_{i=1}^{s} \left[\binom{m_i + n_{max}}{m_i} - \binom{m_i + k_i - 1}{m_i} \right] \tag{1}$$

For a moderately sized problem with $s = 6$, $m_i = 10$ ($\forall \ i$) and $n_{max} = 8$, there are greater than 6.9 x 10^{27} possible design configurations. Clearly, a non-enumerative optimization strategy is required

to identify the optimal solution.

Many previous optimization studies concerned with improving the reliability of system designs, summarized in [4 and 5], use techniques such as nonlinear programming to identify optimal reliability levels at the component or subsystem level. These studies consider component reliability to be a continuous variable (implying an infinite number of component choices). However, there exists another class of problems where system reliability must meet a specified requirement, but component choices are limited to a finite number which have known reliabilities and costs. In these cases, the design problem becomes a combinatorial optimization problem. This approach has been accomplished in past research using integer or dynamic programming [6-12] which require some simplifying assumptions. When formulated as a dynamic programming or integer programming problem, it is necessary to restrict the search space to solutions of a particular form by either requiring $k=1$ and/or prohibiting mixing of different components within a given subsystem [13]. As a result, the true global optimum solution is often not found by these techniques.

GAs offer many advantages compared to alternative methods used to solve the redundancy allocation problem [13]. The mathematical programming approaches are only applicable to a limited problem domain and require simplifying assumptions which limits the search to an artificially restricted search space. Complete justification for using a GA for this problem is provided in Coit and Smith [14]. Other published uses of GA for reliability optimization are Painton and Campbell [15] and Ida, et al. [16]. These works treated the system reliability determination directly during GA search by either algorithms [14, 16] or by simulation [15].

1.2 Reliability Determination

There is a challenge which must be resolved to solve a complex and/or large reliability optimization problem. For each design configuration under consideration, an estimate of system reliability is required as a function of the component reliabilities and the particular series-parallel

configuration. For complex system designs, such as all-terminal networks or systems employing multiple *k*-out-of-*n* redundancies with non-identical components in parallel, determination of exact analytical solutions is computationally complex.

For series-parallel configurations, system reliability can be determined exactly using probability theory or estimated using Monte Carlo simulation. The below equation calculates system reliability for a series-parallel system where mixing of functionally similar components within the same subsystem is allowed.

$$R_s(\mathbf{x_1}, \mathbf{x_2}, \dots \mathbf{x_s} | \mathbf{k}) = \prod_{i=1}^{s} R_i(\mathbf{x_i} | k_i)$$

$$= \prod_{i=1}^{s} \left(1 - \sum_{l=0}^{k_i-1} \sum_{S_l} \prod_{j=1}^{m_i} \binom{x_{ij}}{t_j} r_{ij}^{t_j} \left(1 - r_{ij}\right)^{x_{ij}-t_j} \right)$$

(2)

where, $\qquad S_l = \left\{ \mathbf{t} \in \boldsymbol{I}_+^{m_i} \middle| \sum_{j=1}^{m_i} t_j = l \right\}$

$\qquad \mathbf{t} \;=\; (t_1, t_2, \dots, t_{m_i})$

$\qquad t_j \;=\;$ quantity of the jth component functioning properly (unfailed) $(0 \le t_j \le x_{ij})$

$\qquad r_{ij} \;=\;$ reliability of the jth component available for the ith subsystem

Once a design configuration has been established, Monte Carlo simulation is often used in lieu of exact analytical solutions to provide an accurate method to compute the system reliability of very complex system designs. While simulation is an excellent tool to estimate reliability of a static design configuration, it is not efficient for use in a heuristic optimization search because of the need to recompute system reliability for each solution encountered during the search. (Each recomputation would require multiple simulation runs.)

For large problems where determination of exact analytical solutions is difficult and simulation is computationally inefficient as part of the search strategy, an alternative is the use of neural networks. Neural networks are a nonlinear robust modeling technique which are developed, or trained, based on either analytical or simulated results of a subset of possible

solutions. The resulting model is then used to estimate system reliability as a function of the component reliabilities and the design configuration. In this way, multiple estimates of system reliability are available without solving a new probability model for each new candidate solution, or performing the multiple iterations required with simulation. A disadvantage of using neural networks as a reliability evaluator is that the reliability prediction is only an estimate which may be subject to bias and/or variance depending on the adequacy of the neural network. Figure 2 shows how the neural network works with GA search.

INSERT FIGURE 2 HERE

2. Formulation of Genetic Algorithm

A genetic algorithm is a stochastic heuristic optimization search technique patterned after the natural selection process taking place during biological evolution. The basic GA methodology consists of (a) encoding of solutions as a vector string, (b) random selection of an initial population of solutions, (c) calculation of fitness (objective function value) for each solution, (d) selection of parents (e) breeding of parents (recombination) to create new solutions (children), (f) mutation of existing solutions to create new solutions (mutants), and (g) culling of inferior solutions. This proceeds in an iterative manner, called generations, until the search converges or some other termination criterion is met. GAs have proven to be effective for problems characterized by non-convex search areas.

2.1 Encoding and Schema

For the redundancy allocation problem, a particular solution is encoded as a vector of dimension $s \times n_{max}$. The chromosome is partitioned into different genetic pieces corresponding to each subsystem. For each subsystem, the total number of selected components (n_i) are ordered and indexed from most reliable to least reliable and followed by ($n_{max} - n_i$) of the $m_i + 1$ index corresponding to no further use of redundancy, i.e., a blank gene. For example, consider a system with s = 3, $m_i = 6$ ($\forall i$) and $n_{max} = 5$ for each subsystem. The following vector,

8

$$\mathbf{v} = (1\ 1\ 6\ 7\ 7\ 3\ 7\ 7\ 7\ 7\ 2\ 2\ 7\ 7\ 7)$$

$$\underbrace{\qquad}\quad|\quad\underbrace{\qquad}\quad|\quad\underbrace{\qquad}$$
$$i = 1 \qquad i = 2 \qquad i = 3$$

represents a possible solution for a system with three subsystems. For this system, two of the most reliable and one of the 6th most reliable parts are in parallel in the first subsystem, the 3rd most reliable part (for that particular function) provides the second subsystem and two of the second most reliable part provides the third subsystem. Of course, n_{max} can be individualized for each subsystem creating $n_{max,i}$.

A "blank" gene is used as part of a chromosome whenever there are less than n_{max} components within a particular subsystem. The optimization process involves the selection of n_i components for each of s subsystems where $k_i \le n_i \le n_{max}$. The chromosome is partitioned into s genetic pieces, each corresponding to a particular subsystem. A chromosome is arranged first with the n_1 components for the first subsystem ordered according to their reliability followed by n_{max} - n_1 blank genes with an index of m_1+1, and then followed by the analogous genetic piece corresponding to the second subsystem and so on until all s subsystems are encoded. Using this encoding assures that a child vector will have a quantity of components (n_i^c) bounded by those of the parents, i.e., $n_i^{p1} \le n_i^c \le n_i^{p2}$, for each subsystem.

It is important to design a GA where the chromosomes pass meaningful information to the child solutions. The encoding used was specifically designed to produce meaningful schema. This was accomplished by the pre-processing step which arranges the genes (which each correspond to a particular component) in accordance with their reliability value for each subsystem. This arrangement yields chromosomes where each position has a unique and meaningful interpretation which is then passed to subsequent generations via the associated schema. For example, a particular schema for a single subsystem, (4 ∗ ∗ ∗), corresponds to a subsystem design where at least one of the fourth most reliable component is used, but the three most reliable components are not used anywhere within the subsystem. Alternatively, a schema,

9

(∗ ∗ m_i+1 ∗ ∗), where $m_i + 1$ is the index for a blank gene, represents a subsystem where no more than two components, of any type, are used.

2.2 Evolution Parameters and Operators

The GA search is started by randomly selecting an initial population. For each of the subsystems, the required number of parts (k_i) for the system to operate has been specified. For each subsystem a random number, say n', between k_i and n_{max} is selected, and then n' of the m_i different alternatives (with replacement) are randomly selected. This is repeated for each of the s subsystems, and then for each member of the population. Finally all parts within any subsystem are ordered from most to least reliable and the indices are added for blanks.

Solutions were selected to breed in the manner of [17] by taking a uniform random number between 1 and the square root of the population size, and then selecting the parent with an objective function rank closest to the square of the selected random number. There were different crossover operators considered for the GA. Two possible methods including (1) the transfer of individual components from parent to child solution, and (2) the transfer of whole subsystems, together with the associated components from parent to child. However, transfer of entire subsystems would not introduce sufficient diversity to the population. A preferred crossover operator involves the transfer of individual components, which produces children which share many characteristics with the two parents but still allows for new, although similar and potentially better, subsystem designs.

The breeding operation was uniform crossover: retaining identical portions of the parents, and then randomly selecting, with equal probability, from the two parents for those parts which differ. Because the components were ordered from most to least reliable for each subsystem, matches were fairly common. The child is then ordered from most to least reliable within each subsystem. For example, consider the two following parent vectors ($\mathbf{v_1}$, $\mathbf{v_2}$) and resulting child vector ($\mathbf{v_c}$),

$\mathbf{v_1}$ = (1 1 6 7 7 3 7 7 7 7)
$\mathbf{v_2}$ = (2 3 6 6 7 2 2 7 7 7)
_____
$\mathbf{v_c}$ = (X X 6 X 7 X X 7 7 7) –   identical indices
$\mathbf{v_c}$ = (2 1 6 7 7 3 2 7 7 7) –   random selection of non-matching indices
$\mathbf{v_c}$ = (1 2 6 7 7 2 3 7 7 7) –   final arrangement

The mutation operation took place after breeding and culling so that each mutated solution remained in the population for at least one generation. This allowed a mutant with even an inferior fitness to have the possibility of breeding at least once. The population members to be mutated were uniformly selected and controlled via a preselected number of mutants per generation and a mutation rate. The mutated member replaced the original member in the population, except if the pre-mutated solution was the best solution in the population. The mutation rate was set at 0.10, so that with a 10% probability, each element in a selected solution vector was exposed to mutation. If selected to be mutated, there was a 50% chance that the element is assigned an index of $m_i + 1$, corresponding to a blank, and a 50% chance that a random component index from the m_i alternatives was selected. Within each subsystem of each mutant the genes were then reordered from most to least reliable before insertion to the population.

GA parameter selection was based on brief exploration. Mutation selection probabilities were varied and different population sizes were attempted to test the robustness of the GA and to identify "good" parameter setting values to use on subsequent problems of a similar nature. However, GAs have repeatedly been demonstrated to be robust regarding parameter settings and detailed exploratory investigations were not warranted to identify the "ideal" parameter set.

2.3 Objective Function

The problem formulation involves a constraint on system reliability. However, the crossover operator does not necessarily produce feasible child solutions even if both parent solutions are feasible. Additionally, the mutation operator may alter a feasible solution and result in an infeasible solution. Therefore, different strategies were considered so that the GA leads to a final feasible solution. The two primary approaches are the use of repair mechanisms and penalty

functions.

In general, the use of penalty functions in lieu of repair mechanisms should be avoided because the penalty disturbs the process of identifying promising building blocks. However, for this particular problem, penalty functions were selected for two fundamental reasons. The first reason is that all conceivable repair mechanisms introduce some form of systematic, and undesirable, bias. For example, if random devices are chosen and "upgraded" to the next most reliable alternative until the constraint is satisfied, this would bias the search to solutions with fewer, but more reliable components. Alternatively, a repair mechanism which continually adds the cheapest component until the constraint is satisfied will bias the search to solutions with more, but less reliable components. The second reason is that the ultimate intention is to extend these techniques to actual design problems where there are multiple, and sometimes conflicting, constraints. For example, it would be common to have weight and volume constraints, in addition to reliability. For these problems, there would generally be no conceivable repair mechanism.

The objective function is therefore the sum of the total cost for the selected parts plus a quadratic penalty function. The penalty function is applied when the reliability estimation does not meet the reliability requirement, and thus the design does not represent a feasible solution. The penalty function (P) is similar to that devised in [18] and is given by

$$P = \delta \, (500 \, (R - R_{est}))^2 \tag{3}$$

$$\begin{aligned}
\text{where} \quad \delta \quad &= 0, \ R_{est} \geq R \\
\delta \quad &= 1, \ R_{est} < R \\
R_{est} \quad &= \text{estimated reliability} \\
R \quad &= \text{system reliability requirement} \\
500 \quad &= \text{empirically determined constant.}
\end{aligned}$$

After new solutions were generated, the associated reliability was estimated using the neural network model as described in the next section and the objective function value was determined. Only the best among the parents and children were kept for the next generation, i.e., inferior solutions were culled to maintain constant population size from generation to generation. The GA was terminated after 500 generations for populations of 100 and after 1000 generations

for populations of 50, resulting in 5×10^4 (non-unique) solutions generated.

3. Subsystem Reliability Estimation Using Neural Network

For this research, a straightforward backpropagation neural network was developed to estimate the reliability of single k-out-of-n subsystems based on k, n and n independent part reliability values. The system reliability was then computed as the product of the subsystem reliability estimates. Alternatively, a neural network could have been created to estimate total system reliability directly, but it would have been applicable only for the specified series-parallel configuration. Instead, the developed neural network is capable to predicting subsystem reliability for *any* problem which involves k-out-of-n subsystems.

Neural networks represent only one possible function approximator. From a theoretical perspective, the neural network offers distinct advantages over techniques such as regression models for this particular problem. Neural networks are particularly effective in modeling relationships with nonlinearities and significant interactions. The estimation of subsystem reliability includes both nonlinear aspects and complex interaction effects. While the nonlinear aspects could possibly be accommodated by intelligent data transformations and a regression model, the component reliability interactions could not be handled by any regression model unless a prohibitively large number of interaction terms were explicitly added. For these reasons, the neural network approach was pursued.

3.1 Training Sets and Strategies

The data set used to train and test the neural network was chosen using a full-factorial design of the critical parameters k, n and three underlying distributions (uniform, quadratic skewed-left and quadratic skewed-right) in equal proportions for component reliabilities, where reliability ranged from 0 to 1. The skewed distributions were necessary to expose the neural network to test cases with relatively high and low subsystem reliability for design configurations where these outcomes are rare yet it is critical that an accurate prediction can be made. Initial

attempts to train a neural network using only a uniformly generated training set resulted in poor estimations for very high and very low values of subsystem reliability.

Using 8 as an upper bound for both k and n, there are 192 different combinations (8×8×3). Analytical calculations of k-out-of-n reliability (using Equation 2) were made for 50 randomly chosen instances for each cell in the factorial design, resulting in a total of 9,600 data vectors. 90% of these data vectors were used for neural network training while the remainder were used for neural network validation, forming the test set.

Three separate classes of network architectures were evaluated for possible implementation as the subsystem reliability function evaluator. The three classes were (1) a series of independent networks, each pertaining to a specific value of k, with n_{max} part reliability inputs and a single output for subsystem reliability, (2) a single network using k as an input in addition to the n_{max} part reliabilities and outputting a single subsystem reliability, and (3) a single network using n_{max} part reliability inputs and outputting k_{max} subsystem reliabilities, each pertaining to a predetermined specific value of k. The first two strategies had a single network output while the last had multiple outputs, one for each possible value of k, up to a predetermined k_{max}, which was set equal to n_{max}.

When considering performance and efficiency within a GA search, the best model was found to be of formulation 2 above, and consisted of inputs for k and each individual part reliability (up to n_{max}), one hidden layer with 15 hidden neurons and a single output. Network formulation 1 provided the most accurate reliability predictions, but was inefficient because k neural networks had to be trained, validated and interfaced with the GA optimization. Network formulation 3 was efficient, but was less accurate due to the difficulty of learning multiple (k_{max}) estimations for a single given input vector. The resulting trained neural network from formulation 2 had a mean absolute error of 0.00484 and a root mean square error of 0.00816 for estimating reliability over the test set. Considering that reliability ranges from 0 to 1, this was a good fit to the analytical data. However bias was exhibited by overestimation of the highest reliabilities and underestimation of the lowest reliabilities. This is an atypical neural network problem and is

called overshoot, where the neural approximation exceeds the extremes of the relationship modeled. The authors believe that the skewed data sets overcompensated for the poor performance of the uniform training set, causing overshoot at the extremes of the reliability range. Tempering the mix of skewed and uniform distributions for training should correct this phenomenon.

3.2 Using a Psuedo Constraint

Although the neural network provided a good fit to the test set, there is still always some error associated with even an unbiased prediction and some conditions should be introduced on the use of the neural network as a function evaluator. This approach can best be applied if the constraint is somewhat "soft" and small deviations can be allowed. Alternatively, a pseudo-constraint (R_{psuedo}) can be determined by making a small, conservative adjustment to the constraint to assure that the original constraint is met even if there is a small violation of the pseudo-constraint. For reliability, a minimum must be met, so the pseudo-constraint should be set higher than the actual constraint as shown below:

$$R_{psuedo} = R + \gamma \tag{4}$$

$$\text{where } \gamma \sim f(s)$$

Another consideration to be made when considering the pseudo-constraint is necessitated by the structure of the neural network model. The neural network was used to estimate the reliability of a single subsystem, therefore errors were multiplied across the series system, compounding the underestimation problem. As the number of subsystems, s, in series increases, the compounding of errors geometrically increases. Therefore the conservatism of the pseudo-constraint depends on the number of subsystems in series, where fewer subsystems requires a smaller γ and more subsystems requires a larger γ.

4. Examples And Results

A large system with $s = 6$ and a reliability requirement set at 0.80 was first considered. Recall that this approach works best with a slightly inflated reliability requirement so that

estimation errors from the neural network will not adversely affect the final system design. For each subsystem there were $m = 10$ different part choices and $n_{max} = 8$. There are therefore greater than 6.9×10^{27} different possible design configurations, while the GA search examined about 5×10^4 (non-unique) solutions. Table 1 presents the reliability values and costs associated with each component alternative. The system has a significant need for k-out-of-n redundancies, as indicated by the values of k in the table.

INSERT TABLE 1 HERE

By varying the size of the population and the relative percentage of breeding and mutation, different results (Table 2) were obtained. For each algorithm, 8 different random number seeds were used, and the minimum cost, mean cost and standard deviation are shown. It is evident that the GA was robust across the parameter alterations tested. Because of the size of the problem, the optimum solution could not be enumerated, however when generating 1 million random solutions, the best solution of 1308 found by the GA was 4.84 standard deviations lower than the mean feasible random solution of 2582.77 and 1.72 standard deviations lower than the best feasible random solution of 1762. The percent feasible for randomly generated solutions was 5%, indicating a fairly constrained problem. The solution to the large problem with cost = 1308 is shown in Figure 3.

INSERT TABLE 2 AND FIGURE 3 HERE

To more stringently evaluate the results of the GA approach, the best combination of evolution parameters (population size of 50 and 50%/50% split between breeding and mutation) was selected and a smaller test problem which was a subset of the larger problem was developed. The larger problem was pared down to the first two subsystems listed in Table 1. All other problem specifics remained the same. The search space of this smaller problem was 1.9×10^9 and the optimal solution was found by enumeration, as shown in Figure 4, which had a reliability predicted by the neural network of 0.8022 and a cost of 475. The GA was run with 10 different seeds, and six converged to the optimal design configuration within 1000 generations. Of the

other four solutions found, three were slightly infeasible as shown in Table 3. The costs shown are the truncated penalized cost (where penalty = 0 for feasible solutions). However, the results indicate that as few as two runs will be likely to find the optimal solution for this problem.

INSERT FIGURE 4 AND TABLE 3 HERE

To examine the effects of neural network estimation on the reliability calculated for the final solution of each test problem, Table 4 shows the reliability of each subsystem calculated by analytic means versus the reliability of that same subsystem as predicted by the neural network. The error in the neural network predictions compound as the subsystems are multiplied to find the final system reliability. Errors largely in one direction, for example biased upwards as is found here, provide the worse case in error compounding. Future research should address improvement of neural network estimation for system reliability by either removing all bias or forcing uni-directional bias. For these problems, bias downwards (i.e., underestimation of subsystem reliability) would always ensure a feasible solution because the configuration chosen would have an actual reliability higher than that predicted by the neural model. This may lead to conservative or overly safe designs which are feasible, but not necessarily optimal.

INSERT TABLE 4 HERE

One interesting way to evaluate the efficiency of the combined neural network / GA approach is to consider the number of function evaluations required to train the neural network compared to the number of function evaluations saved by use of the neural network. For the first problem, 9600 function evaluations were required to train and validate the neural network, which then saved approximately 50,000 function evaluations during one GA run. If a single run for a single problem was the extent of the search, then the implicit savings of approximately 5 times fewer function evaluations is intriguing but hardly dramatic considering that the saving was obtained at the expense of precision in the reliability prediction.

However, as the second test problem is considered and additional problems are

contemplated, the benefits of this approach become increasingly clear. The second problem used the same neural network so no additional function evaluations were required yet 50,000 more function evaluations are saved for each single run. Collectively, the neural network approach now leads to approximately an order of magnitude fewer function evaluations. The benefit continues to increase as more problems are analyzed (because the neural network was designed to be a universal approximator). Ultimately, the "cost" of the initial training set becomes inconsequential if the neural network is used repeatedly to analyze additional problems.

5. Conclusions

A unique approach to the optimization of a general form of the redundancy allocation problem using a GA and a neural network approach was formulated and tested. The results are very promising with proven optimal convergence on a small, but complex, test problem and probable near-optimal convergence on a very large problem. The method seems robust to parameter settings and random number seed. The neural network estimator was constructed so that it will serve any series-parallel configuration, and thus could be reused for many design problems. Although the estimation of subsystem reliability was very computationally efficient and quite precise, one has to adapt the constraints to mitigate the possibility of incorrect estimates. Imprecision is compounded as the number of subsystems in series grows due to the multiplicative calculation of system reliability. Clearly more research is needed on the effectiveness and relative efficiency of this combined neural network/GA approach. Specifically, the use of a local optimization post-processor to improve efficiency and a more thorough comparison of the GA with the neural network and a GA with the original objective function would be particularly enlightening.

A hybrid objective function evaluation approach, where the neural network estimation is used early in the search, and a more exact method (either algorithms or simulation) is used directly in later phases of the search to fine tune the final solution identified as optimal by the GA, might be just as effective and more computationally efficient. This is a practical approach, because for real design problems the exact reliability (only estimated by the neural network)

would always be verified with algorithms or simulations. Other simpler surrogates to accomplish this verification would be to verify the reliability of the top few solutions identified from GA optimization, or utilize the pseudo-constraint method discussed earlier.

This approach will be especially applicable to problems where an exact calculation of objective function is not possible and must be estimated through Monte Carlo simulation. This situation is found in reliability design in complex and/or large systems and in network systems. The computational expense of multiple simulation replications for each solution encountered during GA search makes a computationally efficient estimator imperative.

References

[1] A. G. Pipe, T. C. Fogarty, A. Winfield, Balancing exploration with exploitation - solving mazes with real numbered search spaces, *Proceedings of the First IEEE Conference on Evolutionary Computation*, 485-489 (1994).

[2] M. S. Chern, On the computational complexity of reliability redundancy allocation in a series system, *Operations Research Letters*, **11**, 309-315 (1992).

[3] W. Feller, *An Introduction to Probability Theory*, John Wiley & Sons (1968).

[4] F. A. Tillman, C. L. Hwang, W. Kuo, *Optimization of System Reliability*, Marcel Dekker (1980).

[5] F. A. Tillman, C. L. Hwang, W. Kuo, Optimization techniques for system reliability with redundancy - a review, *IEEE Transactions on Reliability*, **R-26**, 148-155 (1977).

[6] D. E. Fyffe, W. W. Hines, N. K. Lee, System reliability allocation and a computational algorithm, *IEEE Transactions on Reliability*, **R-17**, 64-69 (1968).

[7] Y. Nakagawa, S. Miyazaki, Surrogate constraints algorithm for reliability optimization problems with two constraints, *IEEE Transactions on Reliability*, **R-30**, 175-180 (1981).

[8] P. M. Ghare, R. E. Taylor, Optimal redundancy for reliability in series system, *Operations Research*, **17**, 838-847 (1969).

[9] R. L. Bulfin, C. Y. Liu, Optimal allocation of redundant components for large systems, *IEEE Transactions on Reliability*, **R-34**, 241-247 (1985).

[10] K. B. Misra, U. Sharma, An efficient algorithm to solve integer programming problems arising in system reliability design, *IEEE Transactions on Reliability*, **40**, 81-91 (1991).

[11] M. Gen, K. Ida, Y. Tsujimura, C. E. Kim, Large-scale 0-1 fuzzy goal programming and its application to reliability optimization problem, *Computers and Industrial Engineering*, **24**, 539-549 (1993).

[12] M. Gen, K. Ida, J. U. Lee, A computational algorithm for solving 0-1 goal programming with GUB structures and its application for optimization problems in system reliability, *Electronics and Communications in Japan, Part 3*, **73**, 88-96 (1990).

[13] D. W. Coit, A. E. Smith, Optimization approaches to the redundancy allocation problem for series-parallel systems, *Proceedings of the 4th Industrial Engineering Research Conference*,

in print (1995).

[14] D. W. Coit, A. E. Smith, Reliability optimization of series-parallel systems using a genetic algorithm, *IEEE Transactions on Reliability*, in print.

[15] L. Painton, J. Campbell, 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 (1994).

[16] K. Ida, M. Gen, T. Yokota, System reliability optimization with several failure modes by genetic algorithm, *Proceedings of 16th International Conference on Computers and Industrial Engineering*, 349-352 (1994).

[17] D. M. Tate and A. E. Smith, A genetic approach to the quadratic assignment problem, *Computers and Operations Research*, **22**, 73-83 (1995).

[18] A. E. Smith, D. M. Tate, Genetic optimization using a penalty function, *Proceedings of the 5th International Conference on Genetic Algorithms*, 499-505 (1993).

Table 1. Reliability Values and Costs for Each Component for Test Problems.

		Part Alternatives - Unit Reliability									
subsystem	k	1	2	3	4	5	6	7	8	9	10
1	4	0.98	0.93	0.73	0.72	0.71	0.70	0.66	0.62	0.60	0.35
2	2	0.93	0.92	0.89	0.86	0.84	0.81	0.61	0.43	0.39	0.34
3	1	0.94	0.88	0.85	0.76	0.73	0.62	0.60	0.59	0.34	0.31
4	1	0.93	0.67	0.63	0.62	0.62	0.48	0.41	0.41	0.39	0.32
5	2	0.95	0.95	0.90	0.86	0.67	0.66	0.64	0.54	0.38	0.38
6	3	0.96	0.85	0.84	0.76	0.75	0.66	0.65	0.61	0.50	0.48
		Part Alternatives - Unit Cost									
subsystem	k	1	2	3	4	5	6	7	8	9	10
1	4	95	86	80	75	61	45	40	36	31	26
2	2	137	132	127	122	100	59	54	41	36	30
3	1	118	113	108	59	54	49	45	35	30	25
4	1	149	84	74	69	64	58	38	31	26	21
5	2	131	120	103	93	60	43	36	31	26	21
6	3	149	104	96	79	45	40	35	30	25	20

Table 2. Summary of Results to the Larger Test Problem.

Generations	Population	Breed %/Mut. %	Min. Cost	Avg. Cost	Std. Dev.	Coef. Variation
500	100	50/50	1318	1348.63	25.46	0.0189
1000	50	50/50	1308	1337.50	21.76	0.0163
1000	50	75/25	1308	1375.75	40.72	0.0296
1000	50	25/75	1318	1374.34	32.22	0.0234

Table 3. Summary of Non-Optimal Results for Smaller Test Problem.

Non-Optimal Solution	Reliability ($R = 0.80$)	Cost
1	0.7986*	478
2	0.7953*	480
3	0.8005	480
4	0.7995*	479

* Infeasible Solution

Table 4. Comparison of Predicted versus Actual Reliability by Subsystem for Each Problem.

Problem	Source	$s_i = 1$	$s_i = 2$	$s_i = 3$	$s_i = 4$	$s_i = 5$	$s_i = 6$
Large	Calculated	0.92	0.98	0.96	0.96	0.94	0.92
	Predicted	0.96	0.97	0.96	0.97	0.96	0.96
Small	Calculated	0.81	0.94	-	-	-	-
	Predicted	0.87	0.92	-	-	-	-
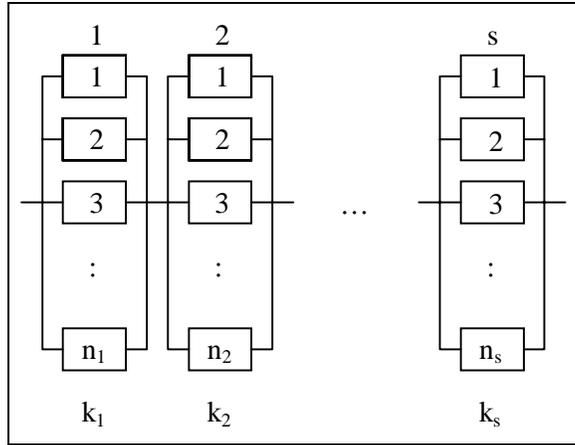
Figure 1. Typical Series-Parallel System.
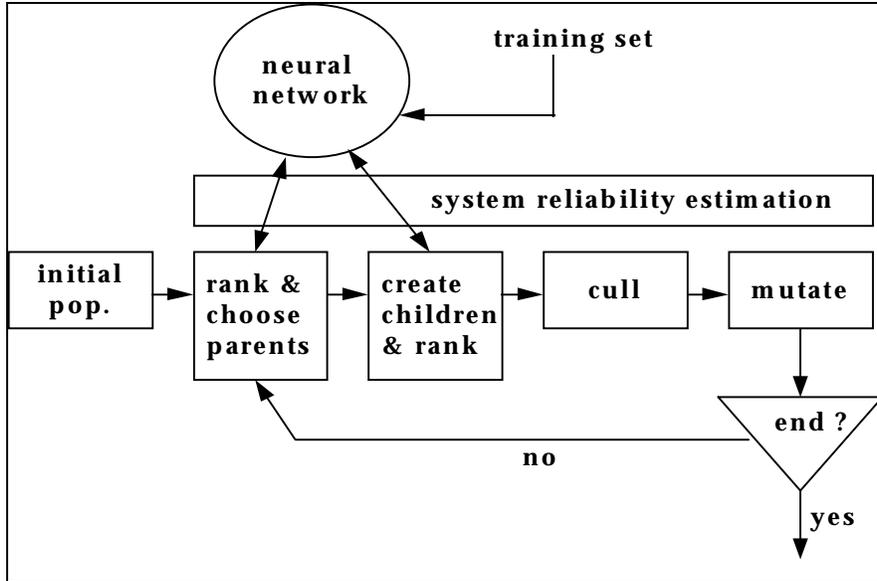
Figure 2. Schematic of Combined Neural Network / GA Approach for Redundancy Allocation.
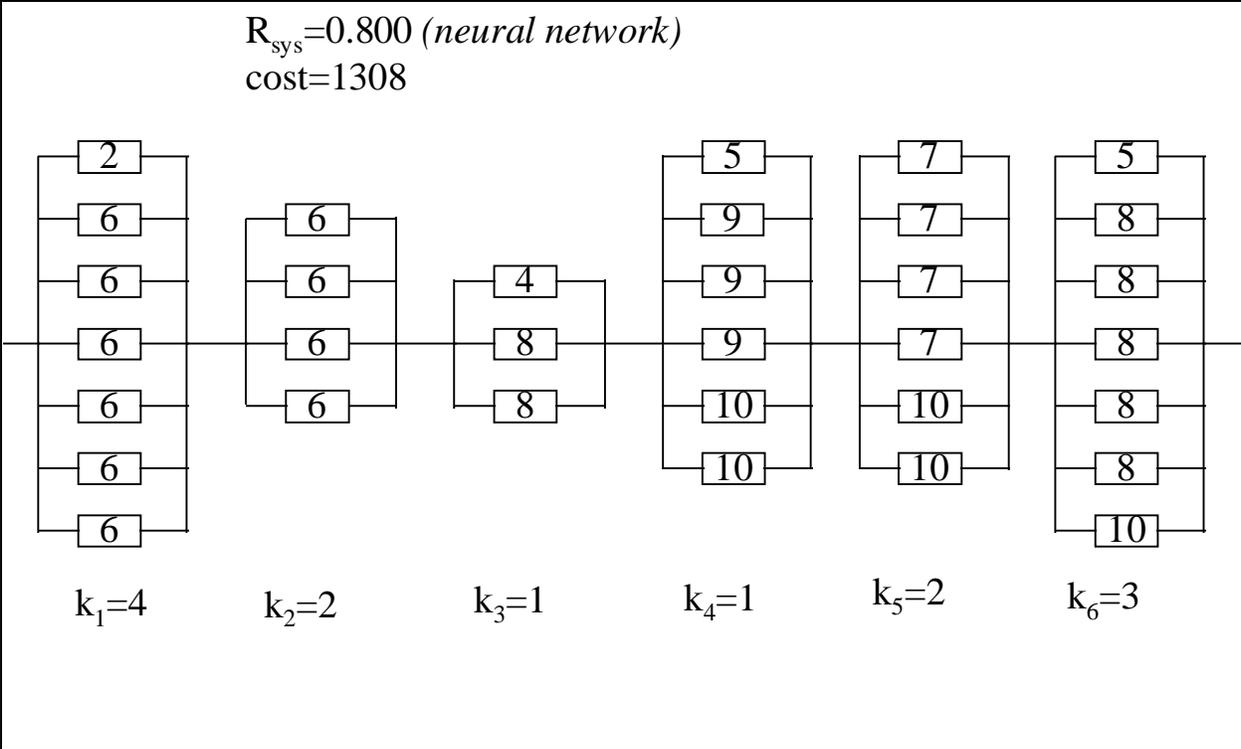
Figure 3. Layout of Best Solution Found for Large Test Problem.
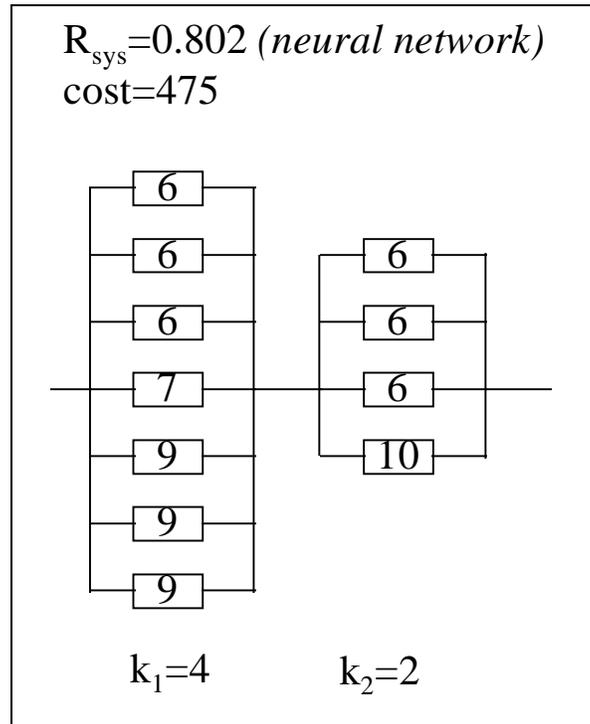
Figure 4. Optimal Solution to Smaller Test Problem.

AUTHORS

David W. Coit; 1178C Benedum Hall; Department of Industrial Engineering; University of Pittsburgh; Pittsburgh, PA 15261 USA.

David W. Coit received a BS degree in mechanical engineering from Cornell University in 1980, an MBA degree from Rensselaer Polytechnic Institute in 1988 and an MS degree in industrial engineering from the University of Pittsburgh in 1993. He is currently a PhD candidate at the University of Pittsburgh. From 1980 to 1992, he was a reliability engineer and project manager at IIT Research Institute, Rome NY where he established reliability programs, analyzed the reliability of engineering designs and developed statistical models to predict reliability of electronic components for client companies. His current research involves reliability optimization, stochastic optimization techniques and industrial applications for artificial neural networks. Mr. Coit is a student member of IEEE, IIE and INFORMS.

Alice E. Smith; 1031 Benedum Hall; Department of Industrial Engineering; University of Pittsburgh, Pittsburgh, PA 15261 USA.

Alice E. Smith is Assistant Professor - Industrial Engineering. After ten years of industrial experience with Southwestern Bell Corporation, she joined the faculty of the University of Pittsburgh in 1991. Her research interests are in modeling and optimization of complex systems using computational intelligence techniques, and her research has been sponsored by Lockheed Martin Corporation, the Ben Franklin Technology Center of Western Pennsylvania, and the National Science Foundation, from which she was awarded a CAREER grant in 1995. She is an Associate Editor of *ORSA Journal on Computing* and *Engineering Design and Automation*, and a registered Professional Engineer in the state of Pennsylvania. Dr. Smith is a member of IEEE, ASEE and INFORMS, and a senior member of IIE and SWE.

LIST OF TABLE CAPTIONS

Table 1. Reliability Values and Costs for Each Component for Test Problems.

Table 2. Summary of Results to the Larger Test Problem.

Table 3. Summary of Non-Optimal Results for Smaller Test Problem.

Table 4. Comparison of Predicted versus Actual Reliability by Subsystem for Each Problem.

LIST OF FIGURE CAPTIONS

Figure 1. Typical Series-Parallel System.

Figure 2. Schematic of Combined Neural Network / GA Approach for Redundancy Allocation.

Figure 3. Layout of Best Solution Found for Large Test Problem.

Figure 4. Optimal Solution to Smaller Test Problem.