Comparison of genetic algorithms and simulated annealing for cost minimization in a multisensor system

Richard R. Brooks, MEMBER SPIE
California State University Monterey Bay
Institute of Communications Science and Technology
Seaside, California 93955-8001

S. S. Iyengar
Louisiana State University
Department of Computer Science
Baton Rouge, Louisiana 70803-4020
E-mail: iyengar@bit.csc.lsu.edu

Suresh Rai
Louisiana State University
Department of Electrical and Computer Engineering
Baton Rouge, Louisiana 70803-4020

Abstract. Many sensor fusion systems combine redundant inputs to increase information reliability. In spite of this, few studies show how to choose redundant sensors for these systems. We find sensor configurations that minimize system cost while ensuring system dependability. Dependability is the generic term for system reliability and availability. Given many types of sensors, all fulfilling system operational requirements, but with different dependability and per item cost, heuristic search methods are used to find minimum cost configurations. Our main contributions are deriving the optimization problem, showing the search can be limited to a multidimensional surface, deriving a fitness function, and providing an efficient algorithm for computing dependability bounds. Two heuristics, genetic algorithms and simulated annealing, are proposed as methods. Experimental results show cost savings of up to 20% compared to systems with only one component type. [S0091-3286(98)00902-7]

Subject terms: sensor fusion; dependability; genetic algorithms; simulated annealing.

Paper FUS-09 received June 2, 1997; revised manuscript received Aug. 20, 1997; accepted for publication Sep. 15, 1997.

1 Introduction

Sensor integration increases the ability of systems to interact with their environment by combining readings from independent physical sensors.[1,2] Highly redundant sensing systems are a major area of sensor integration research. Applications for highly redundant sensors include autonomous land vehicles,[1] manufacturing systems,[2] missile defense systems,[3] nuclear power plant regulators,[2] global positioning systems,[3] and remote sensing applications.[1,2] Existing battlefield analysis systems handle up to 100,000 reports a minute from 156 separate sensor platforms covering over 800 km^2.

Highly redundant sensor systems have these advantages: reduced cost since multiple inaccurate sensors can cost less than a few accurate sensors,[5] increased sensor reliability,[5,6] and increased sensor efficiency.[6] Several methods have been attempted for designing systems that satisfy the need for reliable systems that interact with the real world. Methods not based on redundancy are particularly sensitive to sensor noise.[7] Success in designing these systems depends on both the correct fusion of sensor readings and making the "choice of best possible trade-off at least cost."[3] This paper discusses a strategy for making these trade-offs that minimizes system cost.

The methodology derived here finds combinations of commercial off-the-shelf (COTS) components that fulfill dependability constraints while minimizing system cost. The same methodology can minimize system weight for aerospace applications. Alternatively, the dual problem of finding the most reliable system within weight or cost constraints can be solved.

The approach we propose requires standardized sensors that enable redundancy using heterogeneous components. Good engineering practice dictates the use of off-the-shelf components to reduce system complexity and cost,[8] especially for redundant sensor systems.[9] The National Institute of Standards and Technology has produced a draft standard for sensor interfaces[10] enabling heterogeneous components to be used redundantly without impacting software complexity. For avionics and sensors applications the use of heterogeneous components is innately beneficial to system dependability.[11]

2 Problem Formulation and Layout

The N-modular redundancy (NMR) uses N independent components in parallel.[12] The results from the N components can be compared to ensure correctness.[13] Distributed systems must infer correct information in the presence of failures in a relatively large number of modules, this is known as the Byzantine generals problem.[14] Algorithms exist that make a unanimous decision in the presence of arbitrary errors as long as fewer than one third of the processors are faulty. For a full discussion refer to Ref. 15.

As noted in Section 1, the same problem exists for multisensor fusion applications that glean the best interpretation from noisy sensor data of limited resolution. Figure 1 illustrates a 1-D case for which these algorithms tolerate failures in up to one half of the sensors. Marzullo and Chew[16] expanded this approach to cover more than one dimension and found that the number of faults that can be tolerated depends on the number of dimensions used.[16–18] The relationship between sensor fusion and the Byzantine

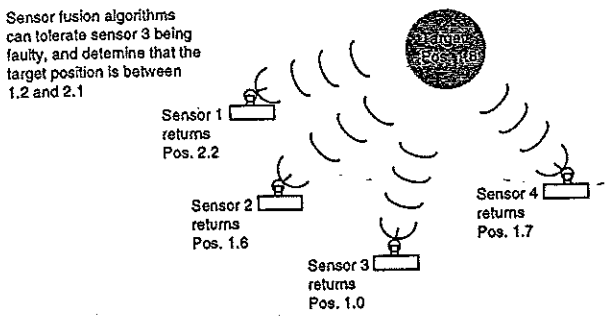

Fig. 1 These sensors function correctly if the data are within 0.5 of the correct values.

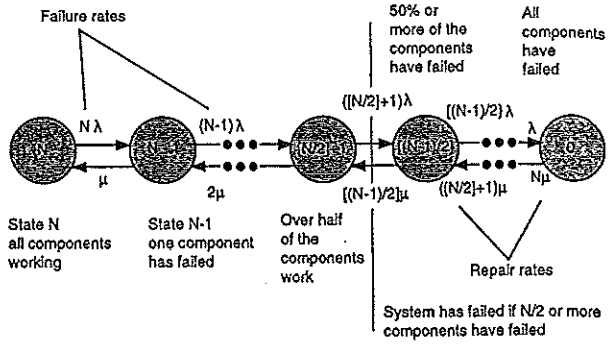

Fig. 2 Markov chain for a system that can tolerate failure of up to half of its components.

generals problems is explained by Brooks and Iyengar.[19,20] The sensor fusion and Byzantine agreement extensions to NMR aid in the design of reliable systems and guarantee that a system will function correctly as long as more than $N/2D$ (in the case of D-dimensional sensor fusion) of the N components function correctly.

Sensor selection to simultaneously improve reliability and resolve resource allocation conflicts is an open question.[6] An approach using system cost and sensor accuracy was given in Ref. 5. This paper presents a more general methodology based on system dependability (reliability or availability) instead of sensor accuracy. We assume component failure is statistically independent. The dependability statistics for components can be based on several statistical models.

For sensor systems, dependability models must reflect the exact problem being considered; if the design is concerned primarily with mechanical failure then mean time to failure and mean time to repair are adequate. Since sensor information must be timely, mechanical degradation, which affects performance, is equivalent to failure. To consider transient and intermittent errors the distributions of fault arrival and duration must be known. Our examples use exponential distributions for component failure and repair for tractability and consistency with reliability literature. Equations are derived for systems where over half of the components must be functional, this can be changed by replacing $N/2$ as necessary.

Dependability is a generic term for both reliability and availability. When a system with strict dependability requirements is being designed, a choice must be made between different component types that could be used redundantly to mask errors. Given a choice between various modules, with different dependability parameters and costs, we search for the configuration that meets dependability requirements with the lowest system cost. Cost may be currency, weight, power consumption, bandwidth requirements, etc., as appropriate for the system being designed.

The paper is organized as follows. Section 3 explains the computations necessary for determining system dependability measurements and presents the necessary equations. It also provides a framework for formalizing the optimization problem in Section 4. Section 5 illustrates the problem geometrically and presents a methodology that obtains the configuration. Results, generated on problems of less than five dimensions, are presented in Section 6. Section 7 considers these results to justify utilizing genetic algorithms and simulated annealing to obtain minimal cost configurations for problems of more dimensions. It also presents results from all three approaches on examples of more than five dimensions. Section 8 discusses the results of this paper and proposes future extensions to our research.

3 Dependability Measures

Dependability is the standard term used in a generic sense to address either system reliability or availability. Reliability refers to the probability that a system is functional at a given point in time, called the mission time T. It is used for single mission systems where repair is unfeasible. Availability is the percentage of time a system will be functional when the system reaches a steady state. Availability is used for systems that can be maintained and repaired.

To compute system reliability, we either consider a Markov model or perform a combinatorial analysis. Figure 2 shows a Markov chain used to model system dependability. For reliability evaluation, the repair rate μ is zero. The system contains N identical components that fail independently. Availability provides a dependability measure for systems where maintenance is possible. For availability, the repair rate μ is nonzero. The formulas derived here for system reliability are also valid for system availability, except that component reliability $r(t)$ must be replaced by component availability $a(t)$ (Ref. 21).

The first approach to solving this problem uses the Markov model directly, assumes all failure rates are constant, derives a set of differential equations directly from the diagram in Figure 2, and solves the equations using Laplace transforms or numerically. Another approach performs a combinatorial analysis and assumes each component has an identical probability of success $r(t)$. If all components have the constant failure rate λ, $r(t)=e^{-\lambda t}$. The probability of failure, $q(t)=1-r(t)$. The assumption of statistical independence uses Bernoulli's law, which gives the probability of i out of N components working at time t:

$$\binom{N}{i}[r(t)]^i[q(t)]^{N-i}. \tag{1}$$

The reliability for the system is the summation of the terms with i varying from N to $\lfloor N/2 \rfloor + 1$. Both approaches derive the same answer.

The combinatorial approach is independent of the distribution defined by the reliability function $r(t)$ and easier to apply when more than one type of component is used (a situation encountered in sensor fusion). When a system has two different types of sensors and $N_1(N_2)$ is the number of components of type 1(2), the total number of sensors is $N = N_1+N_2$. The Markov chain model would require of the order of $3(N-2)$ states, and be computationally expensive as the number of component types increases. The derivation using Bernoulli's law can be cover the new configuration with little additional computation.

Consider the cases where no components of type 1 have failed, one component of type 1 has failed, etc., up to the case where all N_1 components of type 1 have failed. These cases are disjoint and the sum of their probabilities is one.[22] It partitions the sample space giving an expression for system reliability at time t in terms of $r_1(t)$ the reliability of component type 1, and $r_2(t)$ the reliability of component type 2:

$$R(t)=\sum_{k=0}^{N_1}\left(\binom{N_1}{k}[r_1(t)]^k[1-r_1(t)]^{N_1-k} \times \sum_{m=\lfloor N/2\rfloor+1-k}^{N_2}\left\{\binom{N_2}{m}[r_2(t)]^m[1-r_2(t)]^{N_2-m}\right\}\right). \quad (2)$$

This approach can be extended to more than two component types. Evaluating a combination of J different types of components requires J levels of summations in the format of Eq. (2). The following algorithm[23,24] is equivalent to Eq. (2) and reduces the number of multiplications drastically. For sensor fusion applications the constant m is equal to $\lfloor (N-1)/2 \rfloor$.

Procedure: Rel_eval
Input: component failure probabilities $q_1, q_2, \ldots, q_N$
Output: reliability of system composed of the N components
Pr (f)
Procedure:
Step 1a. Let $s[1,1]=q_1$

b. For $k=(2,\ldots,N)$ and $m=(2,\ldots,k-1)$, obtain
$s[1,k]=s[1,k-1]+q_k$
$s[k,k]=s[k-1,k-1]\star q_k$
$s[m,k]=s[m,k-1]+s[m-1,k-1]\star q_k$

Step 2. Solve:

$$\Pr(f)=\sum_{k=m}^{N}(-1)^{k-m}\binom{k-1}{m-1}s[k,N].$$

Rel_eval calculates the nested summations efficiently by creating a tableau of intermediate terms as shown in Figure 3. Should the number of components of a single type be large, say N_1, the algorithm may be made more efficient by making the first N_1, elements of the tableau all elements of type T_1. Column N_1 can be approximated using Stirling's approximation or calculated directly as:

$$s(i,N_1)=\binom{N_1}{i}q_1^i. \quad (3)$$

Figure 3 shows the elements of the tableau that must be computed in this case. If Stirling's approximation is used approximately $N_1+8(N-m)$ multiplications and $3(N-m)$ logs must be calculated to compute all elements $s(i,N_1)$. If $N_1/2<m$, as is the case for the sensor fusion problem, in the worst case calculating $s(i,N_1)$ directly requires approximately $N_1+m(4N_1-m)/2+N-m$ multiplications. The original algorithm takes $(N_1)(N-m)/2$ multiplications to compute the column; $O[(N-N_1)(N-m)]$ multiplications are required to compute the rest of the tableau.

4 Optimization Model

Section 3 provides an efficient algorithm to determine the dependability of a given system configuration. In this section, we derive an optimization model for finding the configuration that fulfills the dependability criteria with minimum cost. Components have either a known component reliability function or a known component availability. Each component type is characterized by its dependability statistics and positive per item cost.

Given J different component types that meet a system's requirements. Finding the combination of components that meets dependability requirements with the lowest total cost, requires considering each combination of the J component

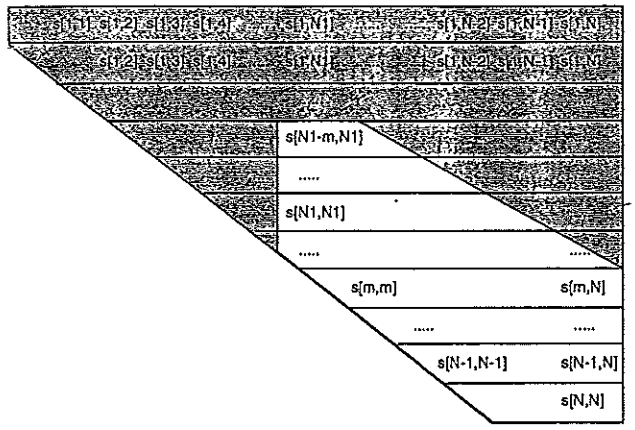

Fig. 3 If there are a large number of components of one type only the nonshaded elements need be computed.

types as a point in a discrete J-dimensional space. This point is described by the J-dimensional vector $(x_1, x_2, \ldots, x_J)$, where each position in the vector corresponds to the number of components of a given type included in the system. If the choice is between three types of components, the combination of 2 components of type 1, 25 components of type 2, and none of type 3 corresponds to point (2, 25, 0). An extension of Eq. (2) to three dimensions determines if a given point in the J-dimensional space corresponds to a system that fulfills the reliability requirement or not.

Since each component has a known per item cost, if component type i has cost c_i, the cost of point (2, 25, 0) is $2(c_1)+25(c_2)+0(c_3)$. The question remains as to how to find the combination that minimizes the fitness function:

$$\sum_{i=0}^{J} c_i Q_i . \tag{4}$$

This is a combinatorial optimization problem. Unfortunately it can not be solved by techniques such as linear programming or integer programming. These techniques are inappropriate since Eq. (2), which defines dependability requirements, is nonlinear.[25] Another approach is needed.

The region with valid solutions is the feasible set in the J-dimensional space.[26] We determine a surface that divides points in the feasible set or with proper subsets in the feasible set, from the rest of the J-dimensional space. We use this surface to restrict our search for the optimal configuration to a small portion of the configurations in the feasible set.

When $N/2$ failures can be tolerated, adding fewer than two components to configuration C giving configuration C', may cause the dependability of C' to be less than the dependability of C. Note that C' may contain more components which may fail than C but the same number of failures can be tolerated in both C and C'. For this reason, the region above the surface may contain configurations that do not meet dependability requirements. However, all points above the surface either fulfill the dependability requirement or contain proper subsets that do.

Lemma 1. An optimal answer must lie on the surface dividing the J-dimensional problem space into two regions: one region containing points satisfying the dependability requirements or with proper subsets satisfying the dependability requirement and one region containing points that neither satisfy the dependability requirements nor contain subsets that satisfy the dependability requirements.

Proof. Three distinct types of points exist: points beneath the surface in the region that does not satisfy reliability constraints, points on the surface, and points above the surface.

Points below the surface can be dismissed trivially since they do not satisfy the dependability constraints. Any point K above the surface that satisfies dependability constraints can be dismissed since there is at least one other point L that satisfies the dependability constraints with at least one component of at least one type fewer. Point L has a lower cost than point K, since all costs are positive. Similarly, any point K above the surface that does not satisfy dependability constraints can be dismissed trivially. Thus the minimum cost point being sought must be located on the J-dimensional surface defining the set of points that satisfy the reliability requirements, or contain subsets that do.

5 Exhaustive Search on the Multidimensional Surface

First, we present a method that finds the minimal cost solution by considering all points on the surface defined in Section 3. For decisions involving a choice between a small number of component types, this search will be reasonable. If the choice is among a large number of types, the time required will be prohibitive, and the heuristics presented in Section 7 should be considered. To search the surface for the minimum cost point we consider the surface's exact shape. We start with the 1-D case and then extend it to higher dimensions.

Component dependability is constrained to a range of values between 0 and 1. A value of 1 (0) indicates a component never fails (functions). The dependability value of a system tends toward zero, remains about 1/2, or asymptotically approaches 1 with the increase in the number of components comprising the system, provided the dependability measure of the components is less than 1/2, exactly 1/2, or greater than 1/2, respectively. It, thus, depicts the "S-shaped property" described in Ref. 27. If components are perfect, system reliability will be 1 no matter how many components are used. Fault masking systems are feasible for components with dependability values between 50 and 100%. Only dependability values within this range should be considered. This paper assumes components fit this requirement. Figure 4 shows the problem space when only one type of component is considered. The feasible set that fulfills the dependability requirements, or contains a subset of components that fulfills the requirements, is a half line and the bounding surface is a point N_1.

As long as the system dependability constraints is less than 1 and individual component dependability is greater than 1/2, the fact that system dependability asymptotically approaches 1 as the number of redundant components in-

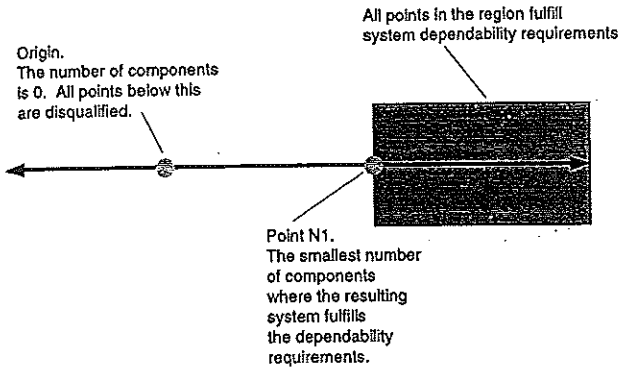

Fig. 4 Half space satisfying constraints and bounding surface in the 1-D case.

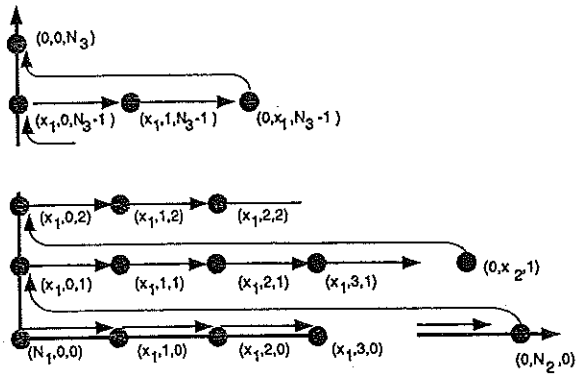

Fig. 6 Path taken by an exhaustive search in three dimensions.

creases proves that number of components N_1 exists that fulfills system dependability constraints. For one dimension, N_1 is all the search needs to find.

Figure 5 shows the feasible set and bounding surface for the 2-D case. The $x(y)$ axis is the number of components of type 1 (2) in the system configuration. Ignoring the possible exception of a few points in the neighborhood of $N_1(N_2)$ due to border effects, the feasible set includes all points greater than $N_1(N_2)$ on the axis. Although part of the bounding surface, they are not considered by the search since they are of higher cost than configurations $(N_1,0)$ and $(0,N_2)$, respectively. The portion of the surface the algorithm searches is between $(N_1,0)$ and $(0,N_2)$. The shape of the bounding curve is defined by Eq. (2). In general increasing the value of the second dimension variable will tend to decrease the value needed for the first dimension variable, but the curve is not smooth. Adding one module to the second dimension may lower the reliability of the system and increase the number of first dimension components required to meet dependability requirements. The experimental results presented in Section 6 confirm this, as do the results presented in Ref. 28.

Extending the search from two to three to J dimensions is straightforward; J represents the number of types of components being considered. As an example, Figure 6 considers a 3-D case and the arrows show the path taken by the search. The search starts with configuration $(N_1,0,0)$, which consists only of components of type 1. The procedure finds the components of type 1 needed if one component of type 2 is present giving configuration $(x_1,1,0)$. Components of type 2 are added one at a time until configuration $(0,N_2,0)$ is reached. After $(0,N_2,0)$, the procedure finds the number of type 1 components needed if none of type 2 and one of type 3 are present, i.e., configuration $(x_1,0,1)$. As before, components of type 2 are added one at a time until configuration $(0,x_2,1)$ is reached. The diagram shows this procedure adding components of type 3 until the final configuration $(0,0,N_3)$ is reached.

When J is greater than 3, the planes defining a 4-D subspace can be traversed in order, similarly the 4-D subspaces defining a 5-D subspace can be traversed, etc., until the entire J-dimensional space has been traversed.

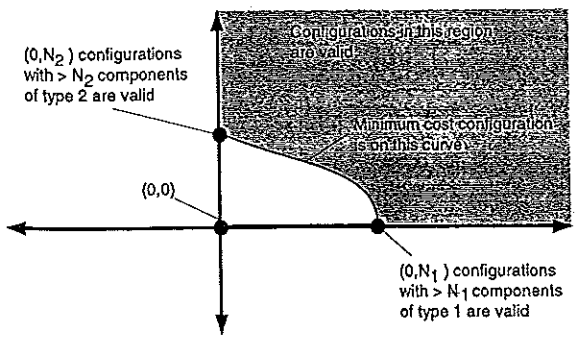

Fig. 5 Valid half space and bounding surface for the 2-D problem.

6 Experimental Results of the Search Algorithm

The search algorithm from Section 5 has been implemented and tested on several sample data sets of fewer than five dimensions. This section discusses representative examples and observations. We use an inflated availability constraint for the first example to illustrate the shape of the J-dimensional surface. All other examples use the availability constraint 0.995. Failure and repair rates are given in terms of FITs.

Example 1 consists of three components (refer to Table 1). The solo configuration describes the number of components of one single type needed to fulfill the availability constraint. Solo cost refers to the cost of that configuration. All components are comparable in that the solo configurations do not vary greatly in cost. It is interesting to note that the minimum cost solution does not contain any components of type 1 although its solo configuration is the least expensive. The minimum cost configuration represents a savings of 6.5% compared to the lowest cost solo configuration.

Figures 7 and 8 represent the surface on which the minimal solution is located. Figure 7 depicts the number of components of type 1 needed for various combinations of components of types 2 and 3. Figure 8 shows the cost of each configuration, again in terms of the number of components of types 2 and 3. Note the shape of the surface is basically the same. Macroscopically the surface is represented by a series of jagged lines. The jaggedness of the lines are caused by two factors: the problem space is dis-

Table 1 Parameters and costs for Example 1.

Parameters	Component 1	Component 2	Component 3
Failure rate	0.0001	0.01	0.005
Repair rate	0.5	0.1	0.3
Unit cost	36.57	5.25	12.35
Solo configuration	5	35	15
Solo cost	182.85	183.75	185.25
Minimum configuration	0	2	13
Minimum cost	171.05		

crete, and tolerating failures of less than 50% of the components means two components must be added to increase system reliability. Adding one component increases the number of items that may fail without increasing the number of failures that can be tolerated; this effect was also noted in Ref. 28. This jaggedness indicates that the search space has many local minima and search methods that depend only on information in the neighborhood of a point will be unsuited to solving this problem, which indicates that genetic algorithms may be an appropriate metaheuristic for solving this problem.

Table 2 illustrates the parameters and results for two examples. Examples 2 and 3 are identical except that the unit prices of components 3 and 4 have each been modified by \$0.01. Note that this minor change has radically altered the minimum cost configuration. This further illustrates the jagged nature of the surface.

7 Metaheuristic Methods

As mentioned in Section 4, the algorithm presented in Section 5 has an exponential growth rate. Moreover, the jaggedness of the search space makes minimal cost configuration detection unsuitable for many search methods, such as hill-climbing, gradient descent, etc. that make decisions based only on information local to the point being considered. These methods stop at a locally optimal solution and will generally be unable to find the globally optimum solution. However, methods such as genetic algorithms,[29] simulated annealing,[9] neural networks,[30] or tabu search[31] are less sensitive to the effects of local minima and should be considered as alternative approaches. This section finds minimal cost configurations by utilizing genetic algorithms and simulated annealing. We observe that they provide good, if not always optimal, combinations in less than exponential time for typical problems of medium size. Both genetic algorithms and simulated annealing use metaphors from other branches of science to solve combinatorial optimization problems. These methodologies apply stochastic processes in the search for optimal solutions, and have been found to be successful in dealing with search spaces which contain local minima.[32,33]

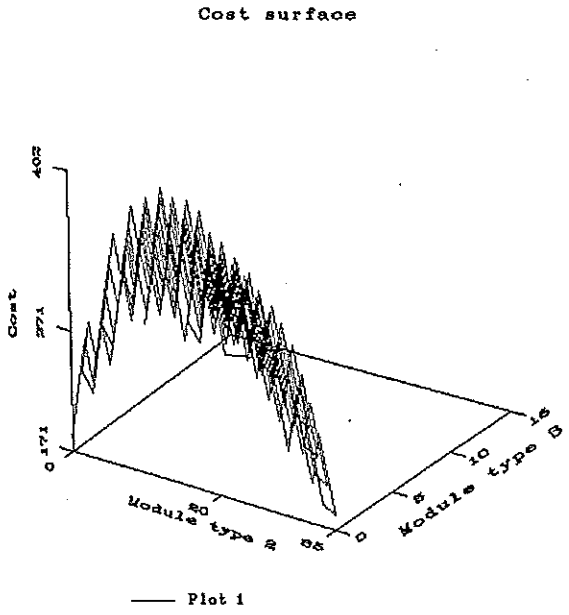

Fig. 8 Cost of resulting system versus number of components of types 2 and 3 needed to fulfill constraints.

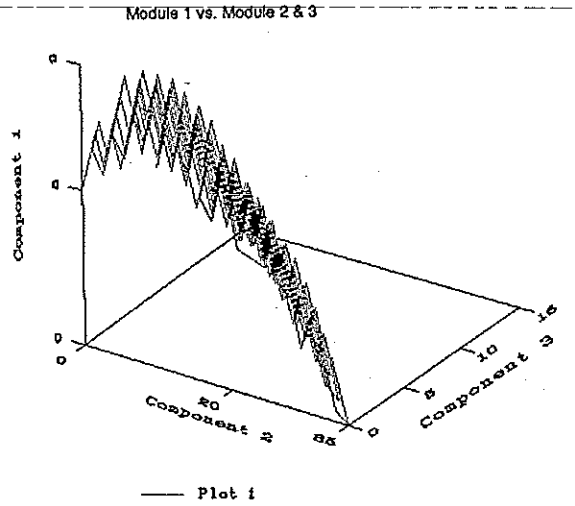

Fig. 7 Number of components of type 1 needed versus number of components of types 2 and 3 needed to fulfill constraints. This shows the shape of the surface containing the optimal configuration.

7.1 *Genetic Algorithms*

Genetic algorithms, first developed in the mid-1960s, attempt to apply a Darwinian concept of survival of the fittest to optimization problems. Possible solutions to a problem are represented in a mathematical structure referred to as a *chromosome* and a diverse set of chromosomes are grouped into a *population*. The relative quality of these answers are determined using a *fitness function*, and this quality is used to determine whether or not the chromosomes will be used in producing the next generation of chromosomes. The next generation is generally formed via the processes of *crossover*, combining elements of two chromosomes from the gene pool, and *mutation*, randomly altering one or more elements of a chromosome (refer to Ref. 32 for details and Ref. 34 for a survey of more recent developments). An in depth comparison of genetic algorithms versus exhaustive search for another reliability design problem can be found in Ref. 35. Several different reproduction strategies are at-

Table 2 Parameters and costs for Examples 2 and 3:

Parameters	Component 1	Component 2	Component 3	Component 4
Failure rate	0.04	0.04	0.03	0.01
Repair rate	0.24	0.19	0.13	0.04
Unit cost (Example 2)	15.00	10.45	10.45	8.76
Unit cost (Example 3)	15.00	10.45	10.44	8.77
Solo configuration	9	13	13	15
Solo cost (Example 2)	135.00	135.85	135.85	131.40
Solo cost (Example 3)	135.00	135.85	135.72	131.55
Minimum configuration (Example 2)	0	6	0	7
Minimum configuration (Example 3)	2	1	8	0
Minimum cost (Example 2)	124.02			
Minimum cost (Example 3)	124.04			

tempted in the literature. We apply an *elitist strategy*, given in Ref. 29 and also discussed in Ref. 20, to the configuration problem.

The solution space to this optimization problem consists of component configurations. For this reason, the chromosomes used by the genetic algorithm consist of a vector that describes a possible system configuration. Position i of the vector, where i is between 1 and J, is an integer ranging from 0 to N_i giving the number of components of type i in the system. The number of elements of type 1 is calculated by the program and set to the smallest number of components of type 1 and needed to fulfill the system dependability requirements. Using component 1 as a dependent variable forces all chromosomes to be points on the surface that contains the optimal answer. The fitness function used in determining the relative quality of chromosomes is the cost of the configuration described by the chromosome.

We used an elitist reproduction strategy consisting of three major steps. First, the best 20% of the gene pool is copied intact into the gene pool for the next generation. Second, 75% of the next generation is determined by randomly mixing elements from two chromosomes chosen at random from the gene pool of the current generation; since our implementation takes multiple genes from both chromosomes this process is called *multipoint crossover*. The remaining 5% of the next generations gene pool consist of random mutations. This strategy has been found to be stable since the quality of the best answers will be monotonically increasing. The large number of mutations is useful since it provides a steady stream of new data to the algorithm; this guards against the algorithm converging prematurely to a suboptimal answer. This mutation strategy is different from strategies used in other reproduction schemes where mutations are done at random throughout the entire population. Mutations applied to members of the elite 20% could make them less optimal. More information on the difference between elitist and classical reproduction strategies can be found in Refs. 29, 36, and 37.

The algorithm used here starts by initializing the gene pool using a set of reasonable answers. These answers include all single component type configurations, and many configurations containing a small number of components. The reproduction scheme and the genetic algorithm determines the following generations. The algorithm was performed for 500 iterations and the best solution present in the gene pool at that point was taken to be the configuration proposed by the genetic algorithm. There are no deterministic means of finding the values for many parameters of a genetic algorithm: such as gene pool size, crossover rate, mutation rate, and stopping criteria, these values must generally be determined experimentally. We used a genetic algorithm as proposed by Holland[32] with the elitist reproduction strategy advocated in Ref. 29 and modified it to suit our application. The algorithm is summarized as follows:

Algorithm: genetic_search
Inputs: J, d_i, for $1 \leq i \leq J$, c_i for $1 \leq i \leq J$, and D.
Outputs: Vector L of length J with the minimum cost dependable configuration.
Procedure:

```
Step 1: Compute N_i, 1≤i≤J. Note, N_i is the number of components of type i needed to meet
    requirement D when no components of another type are used.
Step 2: Sort component types in increasing order of the value N_i*c_i.
Step 3: Generate initial gene pool GP of 150 integer vectors of length J. The size,
    150, of GP was determined experimentally. GP contains all solo configurations, and
    combinations of small numbers of component types.
Step 4: for k=1 to 500 begin /* 500 determined experimentally, */
                             /* as are percentages for elite, */
                             /* crossover and mutation */
```

```
for h=1 to 150 begin
   x1=number of components of type 1 needed for configuration
      GP[h] to fulfill dependability requirement D.
   cost[h]=cost of configuration GP[h] modified so that
      position 1 is x1.
end
for h=1 to 30 begin     /* keep the elite */
 GP_next[h]=the h'th least expensive configuration in GP.
end
for h=31 to 142 begin     /*new combinations from crossover */
 GP_next[h]=randomly combine 2 configurations chosen at random
     from GP.
end
for h=143 to 150 begin    /*mutants */
 GP_next[h]=randomly create configurations.
end
 GP=GP_next
end
Step 5: for h=1 to 150 begin
     x1=number of components of type 1 needed for configuration
      GP[h] to fulfill dependability requirement D.
      cost[h]=cost of configuration GP[h] modified so that
         position 1 is x1.
 end
L=configuration in GP with the lowest cost
output L
```

Genetic algorithms are not sensitive to the presence of local minima since they work on a large number of points in the problem space simultaneously. By comparing many possible solutions they achieve what Holland has termed *implicit parallelism*, which increases the speed of their search for an optimal solution.[32] Discussion of the advantages gained by using genetic algorithms instead of exhaustive search for a different optimization problem based on system reliability can be found in Ref. 38. The elitist reproduction strategy used here is useful in that it guarantees the quality of the answers found will increase monotonically. Note as well that as the number of components increases the computational complexity of this approach changes very little. Only step 1, the length of the vectors and the calculation required to determine the fitness function are affected. This means that increasing the number of component types will not strongly affect the run time of this heuristic.

7.2 *Simulated Annealing*

Simulated annealing attempts to find optimal answers to a problem in a manner analogous to the formation of crystals in cooling solids. A material heated beyond a certain point will become fluid, if the fluid is cooled slowly the material will form crystals and revert to a minimal energy state. Refer to Ref. 33 for a full description of simulated annealing and a discussion of its scientific basis.

The strategy of the algorithm is again based on a *fitness function* comparing the relative merit of various points in the problem space. As before, the problem space is described by vectors corresponding to possible system configurations, and the fitness function is the cost of the configuration described by the vector. The algorithm starts at a point in the search space. From the algorithm's current position a neighboring point is chosen at random. The cost difference between the new point and the current point is calculated. This difference is used together with the current system temperature to calculate the probability of the new position being accepted. This probability is given by a Boltzmann distribution $\exp(-\Delta C/\tau)$. The process continues with the same temperature τ for either a given number of iterations, or until a given number of positions have been occupied, at which time the value τ is decreased. The temperature decreases until no transitions are possible, so the system remains frozen in one position. This occurs only when ΔC is positive for all neighboring points, therefore the position must be a local minimum and may be the global minimum.[39]

The simulated annealing method used in our research is based on the algorithm given in Refs. 33 and 39. The algorithm has been modified so that the parameters being optimized and the fitness function are appropriate for our application, and a cooling schedule has been found which allows the algorithm to converge to a reasonable solution.

Pseudocode of the simulated annealing algorithm is summarized as

```
Algorithm: simulated_annealing
Inputs: J, d_i for 1≤i≤J, c_i for 1≤i≤J, and D.
Outputs: Vector L of length J with the minimum cost dependable configuration.
Procedure:
Step 1: Compute N_i, 1≤i≤J. Note, N_i is the number of components of
  type i needed to meet requirement D when no components of another
  type are used.
Step 2: Sort component types in increasing order of the value N_i*c_i.
Step 3: CC=(N_1,0,....,0) /* Initial position in search space */
 τ=1.0                    /* Initial temperature */
Step 4: CC_mod=1
 step4_iter=0
 While (CC_mod not=0) and (step4_iter<maximum number for step 4)
 do
 begin
  CC_mod=0
  inner_loop_iter=0
  While (CC_mod<maximum number of transitions) and
   (inner_loop_iter<maximum number for inner loop) do
   begin
    x_1=number of components of type 1 needed to fulfill the
     dependability constraint for CC
    CC=CC with first position x_1
    new_CC=random modification of CC
    x_1=number of components of type 1 needed to fulfill the
     dependability constraint for new_CC
    new_CC=new_CC with first position x_1
      ΔC=cost(CC)−cost(new_CC)
      if (ΔC<0) then
        begin
          CC=new_CC
          CC_mod=CC_mod+1
        end
      else with probability according to Boltzmann distribution
      using ΔC and τ do
        begin
          CC=new_CC
          CC_mod=CC_mod+1
        end
      inner_loop_iter=inner_loop_iter+1
    end
   τ=0.9*τ
   step4_iter=step4_iter+1
  end
 Step 5: Output CC as the minimal cost configuration.
```

Just as many different reproduction schemes exist for genetic algorithms, several possible cooling schedules exist for simulated annealing. A cooling schedule is defined by the initial temperature, the number of iterations performed at each temperature, the number of position modifications allowed at a given temperature and the rate of decrease of the temperature. The answers found by the algorithm are directly dependent on the cooling schedule and no definite rules exist for defining the schedule.[33,39] The cooling schedule used in this application started with a temperature of 1.0 which decreased at a rate of 10%. The total number of iterations at a given temperature was limited to $100J$, and the maximum number of positions visited at a given temperature was limited to $10J$. The cooling schedule is important in that it determines the rate of convergence of the algorithm as well as the quality of the results obtained. The

Table 3 Parameters and costs for Example 4.

	Comp. 1	Comp. 2	Comp. 3	Comp. 4	Comp. 5	Comp. 6	Comp. 7	Comp. 8
Failure rate	0.03	0.05	0.05	0.01	0.08	0.03	0.04	0.01
Repair rate	0.45	0.95	0.3	0.05	0.45	0.97	0.3	0.1
Unit cost	22.06	25.00	12.25	10	9.50	36.7	15.50	15.10
Solo config.	5	5	9	11	11	3	7	7
Solo cost	110.300	125.00	110.25	110.00	104.50	110.10	108.50	105.70
Min. config.	0	1	0	0	0	0	0	4
Min. cost	85.40							
GA config.	0	1	0	0	0	0	0	4
GA cost	85.40							
SA config.	0	1	0	0	0	0	0	4
SA cost	85.40							

complexity of this approach could potentially increase in the order of J^2. This increase is significantly less than the exponential growth of the exhaustive search, but greater than the increase for the genetic search.

The starting configuration is taken as the lowest cost solo configuration. New configurations are generated randomly from the current configuration. In choosing a new configuration each position in the vector had a 40% chance of being modified. Those positions chosen for modification had a 25% chance of being incremented, a 25% chance of being decremented, and a 50% chance of staying the same.

7.3 *Experimental Results*

Examples 4 and 5 (see Tables 3 and 4) are test cases of multiple dimensions which have been used to test the exhaustive search, simulated annealing (SA), and genetic algorithms (GAs) approaches to this problem. They consist of eight and eleven dimensions respectively. The number of dimensions was kept relatively small to allow the use of the exhaustive search algorithm for verifying the global minimum.

For both cases, SA succeeded in finding the global minimum. In the tests performed here, SA was also the least computationally intensive of all three approaches providing its answer within minutes instead of hours. While these results are positive, it should be noted that this algorithm is not guaranteed to provide the globally optimal answer.

The GA approach found the global minimum only for Example 4, however its solution for Example 5 is very close to the global minimum. This algorithm was more computationally intensive than the SA algorithm, and there is no clear criteria for stopping the GA. This approach does have two positive characteristics: it is less computationally expensive than the exhaustive search method, and the elite reproduction scheme guarantees that the cost of the best answers will be monotonically decreasing.

Figure 9 shows the run times of the three approaches for problems consisting of seven, eight and nine components. The graph shows the number of second required versus the number of components considered. All tests were run on a 60 MHz Pentium system. The exponential growth rate of exhaustive search makes it impractical for problems of large size. SA was the fastest approach in our tests running in minutes instead of hours. The growth rate for GAs was fairly low, however, indicating that it may be reasonable for large problems.

8 Discussion and Future Extensions

The methods described in this paper are useful for finding system configurations made up of individual components

Table 4 Parameters and costs for Example 5.

	C. 1	C. 2	C. 3	C. 4	C. 5	C. 6	C. 7	C. 8	C. 9	C. 10	C. 11
Failure rate	0.05	0.10	0.15	0.3	0.01	0.20	0.03	0.13	0.01	0.09	0.12
Repair rate	0.30	0.39	0.50	0.95	0.30	0.97	0.15	0.59	0.07	0.42	0.62
Unit cost	17.80	10.00	8.00	7.00	53.00	12.00	14.00	10.75	18.50	11.00	12.60
Solo config.	9	15	19	21	3	13	11	13	9	13	11
Solo cost	160.20	150.00	152.00	147.00	159.00	156.00	154.00	139.75	166.50	143.00	138.60
Min. config.	0	0	0	0	0	0	0	1	0	4	6
Min. cost	130.35										
GA config.	0	0	0	0	0	0	0	0	0	5	6
GA cost	130.60										
SA config.	0	0	0	0	0	0	0	1	0	4	6
SA cost	130.35										

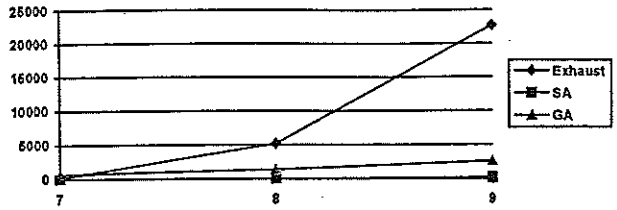

Fig. 9 Run times of exhaustive search, SA and GAs for problems with 7, 8 and 9 component types.

and relying on fault masking. They verify that the system will fulfill dependability constraints as long as component failures are statistically independent. Examples were given to illustrate how this methodology can be put into practice and result in cost savings.

Note that Example 4 has provided savings of approximately 20% compared to the lowest cost solo configuration. This is a sizable improvement. The dual problem of the problem studied here, maximizing system reliability within fixed cost or weight bounds, is equally important and can easily be solved by switching the cost function and constraint functions proposed. For large scale applications using redundant sensors improvement of 10% or more in weight or dollars results in considerable savings.

For small scale problems the exhaustive search method is preferable, since it is guaranteed to find the lowest cost combination of systems components. This paper has shown that both GAs and SA are viable approaches to finding optimal, or near optimal configurations for large scale problems. In these tests, SA performed remarkably well and appears to be the methodology best suited to solving the problem. A reasonable approach would be to use GAs as a secondary approach to verify the results found by the simulated annealing algorithm.

It should be noted, however, that GAs and SA both have drawbacks. Both are relatively insensitive to the presence of local minima in the search space. This insensitivity is partially obtained by the creative application of nondeterminism. Nondeterminism also means that the quality of the answers found by the algorithms will vary from case to case. It is impossible to know how long the algorithms will need to find the global minima, if they ever find the global minima.

GAs are sensitive to the reproduction strategy chosen, including mutation rates and how elements are chosen for crossover. SA is sensitive to the cooling schedule, which includes the initial temperature and the rate of decrease of temperature. The quality of the answers found and the amount of time needed to find reasonable answers are directly dependent on the reproduction strategy of a GA and the cooling schedule of an SA approach. Both the reproduction strategy and cooling schedule must be found through a process of trial and error. For neither is there a guarantee that a particular strategy or schedule will be appropriate for all cases encountered. In spite of this, our research has shown that both can be applied effectively as heuristics for solving this problem.

It is interesting to note that SA appears to work better than GAs on this problem. This supports a supposition made by Hooker[40] that SA works best on problem spaces containing narrow and deep basins of attraction such as the one shown in Figure 8.

For many applications, engineers should loosen the assumptions used in defining these methods. One assumption taken is that the sensors share the same coverage of the area being sensed. This will most often not be the case. Configurations can be derived that cover a given region but where individual sensors cover a small portion of the region, by changing the constraints used to reflect this requirement.

It has been assumed that component failures are statistically independent. Sensors of the same type will be sensitive to the same type of interference, where sensors of another type may not be sensitive to that particular noise. This factor would tend to favor building systems using many different types of components, and the cost function can be modified to reflect this.

Another factor to be considered is that real time systems have specific time constraints. For this reason, it would often be preferable to have systems using a smaller number of components to reduce the amount of bandwidth needed for reading and comparing the inputs from the individual components. This can be implemented as either a constraint or a part of the cost function.

Acknowledgments

The authors would like to thank the guest editor and the anonymous reviewers whose comments helped improve the content and presentation of this paper. This work was supported in part by the Office of Naval Research grant No. 00014-94-1-0343.

References

1. R. Luo and M. Kay, "Data fusion and sensor integration: state-of-the-art 1990s," in *Data Fusion in Robotics and Machine Intelligence*, Abidi and Gonzales, Eds., pp. 7–136, Academic Press, Boston (1992).
2. S. S. Iyengar, L. Prasad, and H. Min, *Advances in Distributed Sensor Integration: Applications and Theory*, Prentice Hall (1995).
3. F. Maurin, "Keynote address," in *Fault Tolerant Design Concepts for Highly Integrated Flight Critical Guidance and Control Systems, NATO Advisory Group for Aerospace Research and Development Conf. Proc.* No. 456 AGARD-CP-456, pp. KE-1–KE-4 (1990).
4. J. Franklin, L. Davis, R. Shumacher and P. Morawski, "Military applications," in *Encyclopedia of Artificial Intelligence*, pp. 604–614, Wiley, New York (1986).
5. J. G. Balchen and F. Dessen, "Structural solution of highly redundant sensing in robotics systems," in *Highly Redundant Sensing in Robotics Systems*, NATO Advanced Science Institutes Series, Vol. F-58, Tou and Balchen, Eds., pp. 263–275, Springer Verlag, (1990).
6. J. T. Tou, "A knowledge-based system for redundant and multisensing in intelligent robots," in *Highly Redundant Sensing in Robotics Systems*, NATO Advanced Science Institutes Series, Vol. F-58, Tou and Balchen, Eds., pp. 3–14, Springer Verlag, (1990).
7. T. Sadeghi and G. Mayville, "Fault-tolerant flight-critical control systems," in *Fault Tolerant Design Concepts for Highly Integrated Flight Critical Guidance and Control Systems, NATO Advisory Group for Aerospace Research and Development Conf. Proc.* No. 456 AGARD-CP-456, pp. 26-1–26-12 (1990).
8. Y. Levendel, "Fault tolerance cost effectiveness," in *Hardware and Software Architectures for Fault Tolerance: Experience and Perspectives*, Banâtre and Lee, pp. 15–20, Springer Verlag, (1994).
9. J. Wahrburg, "Control concepts for industrial robots equipped with multiple and redundant sensors," in *Highly Redundant Sensing in Robotics Systems*, NATO Advanced Science Institutes Series, Vol. F-58, Tou and Balchen, Eds., pp. 277–291, Springer Verlag (1990).
10. J. L. Michaloski, P. G. Backes, and R. Lumin, "Integration of sensor feedback and teleoperation into an open standard," in *Sensor Fusion and Networked Robotics VIII, Proc. SPIE* **2589**, 206–217 (Oct. 1995).
11. J. Kershaw, "Dependable systems using VIPER," in *Fault Tolerant Design Concepts for Highly Integrated Flight Critical Guidance and Control Systems, NATO Advisory Group for Aerospace Research and Development Conf. Proc.* No. 456 AGARD-CP-456, pp. 25-1–25-7 (1990).

12. T. Krol, "(N,K) concept fault tolerance," *IEEE Trans. Comput.* **C-35** (4), 339–349 (1986).
13. D. P. Siewiorek and R. S. Swarz, *Reliable System Design and Evaluation*, Digital Press, Bedford, MA (1992).
14. L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.* 4(3), 382–401 (1982).
15. M. Barborak, M. Malek, and A. Dahbura, "The consensus problem in fault tolerant computing," *ACM Comput. Surv.* 25(2), 171–220 (1993).
16. P. Chew and K. Marzullo, "Masking failures of multidimensional sensors," in *Proc. Tenth Symp. on Reliable Distributed Systems*, pp. 32–41 (1991).
17. R. R. Brooks and S. S. Iyengar, "Algorithm for resolving interdimensional inconsistencies in redundant sensor arrays," in *Proc. of Indo-US Workshop on Parallel and Distributed Signal and Image Integration Problems*, pp. 113–116, Pune, India (Dec. 1993).
18. R. R. Brooks and S. S. Iyengar, "Optimal matching algorithm for multi-dimensional sensor readings," in *Sensor Fusion and Networked Robotics, Proc. SPIE* (1995).
19. R. R. Brooks and S. S. Iyengar, "Methods of approximate agreement for multisensor fusion," in *Signal Processing, Sensor Fusion and Target Recognition IV, Proc. SPIE* (1995).
20. R. R. Brooks and S. S. Iyengar, *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall PTR, Upper Saddle River, NJ (1997).
21. Roseaux, *Exercises et Problèmes Résolus de Recherche Opérationelle*, pp. 147–148, Editions Masson, Paris (1987).
22. K. L. Chung, *Elementary Probability Theory with Stochastic Processes*, Springer Verlag, Heidelberg (1974).
23. S. Rai, "Evaluating FTRE's for dependability measures in fault tolerant systems," *IEEE Trans. Comput.* 44(2), 275–285 (1995).
24. R. Sahner and K. Trivedi, "Performance and reliability analysis using directed acyclic graphs," *IEEE Trans. Software Eng.* **SE-13**(10), 1105–1114 (1987).
25. R. Faure, *Guide de la Recherche Opérationelle, tome 1 les fondements*, pp. 159–164, Editions Masson, Paris (1986).
26. G. Strang, *Linear Algebra and Its Applications*, Academic Press, New York (1976).
27. R. Barlow and F. Proschan, *Mathematical Theory of Reliability*, Wiley, New York (1965).
28. N. H. Vaidya and D. K. Pradhan, "Fault-tolerant design strategies for high reliability and safety," *IEEE Trans. Comput.* 42(11), 1195–1206 (1993).
29. J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA J. Comput.* 6(2), 154–160 (1994).
30. E. Davalo and P. Daïm, *Des Reseaux de Neurones*, Editions Eyrolles, Paris (1989).
31. E. Taillard, "Parallel taboo search techniques for the job shop scheduling problem," *ORSA J. Comput.* 6(2), 108–117 (1994).
32. J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor (1975).
33. P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel, Dordrecht (1987).
34. M. Srinivas and L. M. Patnaik, "Genetic algorithms: a survey," *IEEE Comput.* 27(6), 17–26 (1994).
35. A. Kumar, R. Pathak, and Y. Gupta, "Genetic-algorithm-based optimization for computer network expansion," *IEEE Trans. Reliabil.* 44(1), 63–72 (1995).
36. R. R. Brooks, S. S. Iyengar, and J. Chen, "Automatic correlation and calibration of noisy sensor readings using elite genetic algorithms," *Artif. Intell.* 18(1–2), 339–354 (1996).
37. R. R. Brooks, "Robust sensor fusion algorithms: calibration and cost minimization," PhD Dissertation, Department of Computer Science, Louisiana State University, Baton Rouge (1996).
38. L. Painton and J. Campbell, "Genetic algorithms in optimization of system reliability," *IEEE Trans. Reliabil.* 14(2), 172–178 (1995).
39. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in Fortran*, pp. 436–448, Cambridge University Press (1986).
40. J. N. Hooker. "Needed: an empirical science of algorithms," *Operat. Res.* 42(2), 201–212 (1994).

Richard R. Brooks is an assistant professor of applied computing at California State University, Monterey Bay. He received a PhD in computer science in 1996 from Louisiana State University and a BS in mathematical sciences in 1979 from Johns Hopkins University. Dr. Brooks also studied operations research and computer science at the Conservatoire National des Arts et Metiers in Paris. His work experience includes projects with Goddard Spaceflight Center, Radio Free Europe/Radio Liberty Munich and the French stock exchange authority. As a consultant for the World Bank he helped implement their network in Africa, Eastern Europe and Central Asia. Dr. Brooks is a member of ACM and INFORMS.

S. S. Iyengar is fellow of the IEEE and chairs the Computer Science Department and is a professor of Computer Science at Louisiana State University (LSU). He has been involved with research in high-performance algorithms and data structures since receiving his PhD in 1974, and has directed over 22 PhD dissertations at LSU. He has served as a principal investigator on research projects supported by the Office of Naval Research, the National Aeronautics and Space Administration, the National Science Foundation, the California Institute of Technology's Jet Propulsion Laboratory, the Department of Navy—NORDA, the Department of Energy, LEQFS—Board of Regents and U.S. Army Office. His publications include several books and over 200 publications, including 105 archival journal papers. He is a series editor for *Neuro Computing of Complex Systems* and an area editor for the *Journal of Computer Science and Information*. He has served as a guest editor for the *IEEE Transactions on Software Engineering*, the *IEEE Transactions on System, Man and Cybernetics* and the *IEEE Transactions on Knowledge and Data Engineering*. He is a fellow of the IEEE for his research contributions in the area of algorithms and data structures in image processing.

Suresh Rai is working with the Department of Electrical and Computer Engineering at Louisiana State University, Baton Rouge. Dr. Rai has taught and researched in the areas of reliability engineering, fault diagnostics, and parallel and distributed processing. He is a coauthor of the book *Waveshaping and Digital Circuits* and the tutorial texts *Distributed Computing Network Reliability* and *Advances in Distributed System Reliability*. He has guest edited a special issue of *IEEE Transactions on Reliability* on the topic Reliability of Parallel and Distributed Computing Networks. Dr. Rai was a committee member for the IPCCC'91 (Phoenix) conference. Currently, he is an associate editor for *IEEE Transactions on Reliability*. Dr. Rai received his BE degree from Benaras Hindu University in 1972, his ME from University of Roorkes in 1974, and his PhD from Kurukshetra University in 1980. He joined the Regional Engineering College at Kurukshetra in 1974 and was there for 6 years. After a brief stay at MMM Engineering College, Gorakhpur, he transferred to University of Roorkee in 1981 as an associate professor. He was also with the School of Engineering at North Carolina State University, Raleigh, for 2 years. Dr. Rai is a senior member of the IEEE and a member of the ACM.