ADAPTIVE PENALTY METHODS FOR GENETIC OPTIMIZATION OF

CONSTRAINED COMBINATORIAL PROBLEMS

David W. Coit and Alice E. Smith[1]
Department of Industrial Engineering
University of Pittsburgh
1048 Benedum Hall
Pittsburgh, PA 15261
aesmith@engrng.pitt.edu

David M. Tate
Decision Sciences Applications, Inc.
1110 N. Glebe Rd., Suite 400
Arlington, VA 22201
dtate@dsava.com

Subject Classification Scheme

1. Computers/computer science - Artificial intelligence

   This research uses the heuristic optimization technique of Genetic Algorithms, which is sometimes considered an artificial intelligence technique.

2. Facilities/equipment planning - Layout

   One application in this paper is the unequal area facilities layout problem.

3. Programming - Integer - Heuristic

   A heuristic optimization technique is used.

4. Reliability - Redundancy / spares

   One application in this paper is the redundancy allocation problem.

---

[1] Corresponding author.

ADAPTIVE PENALTY METHODS FOR GENETIC OPTIMIZATION OF CONSTRAINED

COMBINATORIAL PROBLEMS

Abstract

The application of genetic algorithms (GA) to constrained optimization problems has been hindered by the inefficiencies of reproduction and mutation when feasibility of generated solutions is impossible to guarantee and feasible solutions are very difficult to find. Although several authors have suggested the use of both static and dynamic penalty functions for genetic search, this paper presents a general adaptive penalty technique which makes use of feedback obtained during the search along with a dynamic distance metric. The effectiveness of this method is illustrated on two diverse combinatorial applications; (1) the unequal-area, shape-constrained facility layout problem and (2) the series-parallel redundancy allocation problem to maximize system reliability given cost and weight constraints. The adaptive penalty function is shown to be robust with regard to random number seed, parameter settings, number and degree of constraints, and problem instance.

1. Introduction to Genetic Algorithms

Genetic Algorithms (GA) are a family of parallel search heuristics inspired by the biological processes of natural selection and evolution. In GA optimization, a population of individual solutions is maintained at all times. Individuals are selected from this population to be parents, with preference given to solutions having better objective function value, or fitness, and these solutions are recombined using a crossover operator to produce new feasible solutions called children. At the same time, some individuals in the population are perturbed, or mutated. Some or all of the remaining solutions in the old population are then culled to make room for the new solutions. The resulting group of new children, mutants and unchanged individuals from the old solution set constitutes the next generation in the evolution of the population.

Typically, the crossover and mutation operations are performed at the level of the data structures encoding the solutions, rather than operating on solutions themselves. For example, variables in continuous optimization problems are most commonly encoded as binary strings,

representing coordinates in a discretized lattice of possible variable values. Since parents are chosen based on their fitness, portions of encoding which tend to be found in good solutions will tend to persist in the population, while portions of encoding which characterize poor solutions will tend to disappear from the population. Research has shown that only a weak correlation between solution quality and partial encodings is necessary to ensure that the population will tend to evolve toward better average fitness [15, 19]. Attributes of GA are flexibility in encoding and evolution, and freedom from restrictions to smoothness, continuity or uni-modality of the optimization response surface.

GA optimization was pioneered by Holland [19] and DeJong [9] for multi-dimensional continuous optimization of multi-modal functions. Goldberg [14, 15] expanded the theoretical foundations of GA, as well as the range of applications. GA methods have been successfully extended to classical combinatorial optimization problems, including job shop scheduling [37] the Traveling Salesman Problem (TSP) [16, 22, 42], VLSI component placement [7], quadratic assignment problems [21, 38] and others. This paper discusses previous approaches using GA search for constrained optimization problems, then introduces the general adaptive penalty approach. The adaptive penalty is demonstrated to be both effective and robust, with little user tuning required, on two diverse combinatorial problems.

2. Adapting GA to Constrained Problems

As originally conceived, GAs produce new solutions by recombining the encoded solutions (genotypes) of parents from the current population, and by mutating encoded solutions. In many cases, it is easy to devise an encoding, a crossover operator and a mutation operator such that feasible parents will always produce feasible children and/or mutants. An example using binary integers illustrates this: exchanging bits between two k-bit integers will always produce a k-bit integer. If a one-to-one correspondence between k-bit integers and feasible solutions is established, every newly-produced encoding will correspond to a feasible solution to the original problem.

In other cases, it is not clear how an encoding and genetic operators can be defined to preserve feasibility. Consider a combinatorial problem such as TSP, where the domain of feasible solutions is all possible permutations of the integers 1, 2, ..., n. If TSP solutions are encoded as permutations, then simple swapping of entries between two parent solutions cannot occur without the risk of an encoding which does not correspond to a tour. There are several possible ways to address this problem. One is to reduce the problem to the familiar case of a one-to-one mapping between the $n!$ possible tours and the first $n!$ integers, and then use a binary string encoding. This system has two drawbacks. First, it adds considerable computational overhead in the encrypting of tours as integers and the decrypting of integers as tours. Second, the number of bits required to represent an n-city tour is $\log_2(n!)$, so that even a 50-city TSP would require more than 200 bits per encoded solution. Clearly, the computational burden would only be worse for the many combinatorial problems whose feasible sets grow even more quickly than TSP as a function of problem dimension.

A more appealing approach to preserving feasibility of children is to increase the complexity of the breeding and mutation operators, so that they are guaranteed to produce feasible encodings. For TSP, compensatory conflict-resolution operators (repair operators) could be implemented, so that the resulting child encoding is itself a permutation. Several researchers have taken this approach to problems [24, 25, 30, 38], and it works well without restrictive computational cost. When applicable, there are two reasons why this method is effective. First, it is easy to blend two permutations into a string which is almost a permutation itself. Second, it is easy to modify that string so that it becomes a permutation yet still shares many structural features with its parents.

Unfortunately, many optimization problems involve constraints for which it is not easy to repair an infeasible solution in order to make it feasible. The repair operators may be prohibitively computationally expensive or they may severely disturb the superior aspects of the parents solutions carried in the children, defeating the fundamental strength of reproduction in GA. Furthermore, in many cases the problem of finding any feasible solution is itself NP-hard

[13]. A number of methods have been proposed in the GA literature for applying GA to such highly constrained problems. DeJong and Spears [10] suggest polynomially reducing other NP-Complete problems to the Boolean Satisfiability problem, for which effective (and compact) GA implementations are known. This method has the drawback that the polynomial reduction involved may be extremely complex, and may greatly increase the size of the problem. Michalewicz [26] and Michalewicz and Janikow [27] suggest eliminating the equalities in constraints and formulating special genetic operators which guarantee feasibility. This approach works efficiently for linear constraints. A more generic approach borrowed from the mathematical programming literature is that of exterior penalty methods.

3. Penalty Function Methods

Penalty functions have been a part of the literature on constrained optimization for decades. In the area of combinatorial optimization, the popular Lagrangian relaxation method [2, 11, 32] is a variation on the same theme: temporarily relax the problem's hardest constraints, using a modified objective function to avoid straying too far from the feasible region. In general, a penalty function approach is as follows. Given an optimization problem,

$$\max \quad z(\mathbf{x}) \qquad\qquad\qquad\qquad \text{(P)}$$
$$\text{s.t.} \quad \mathbf{x} \in A$$
$$\qquad\quad \mathbf{x} \in B$$

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

$$\max \quad z(\mathbf{x}) - p(d(\mathbf{x}, B)) \qquad\qquad\qquad \text{(R)}$$
$$\text{s.t.} \quad \mathbf{x} \in A$$

where $d(\mathbf{x}, B)$ is a metric function describing the distance of the vector $\mathbf{x}$ from the region B, and $p(\cdot)$ is a monotonically nondecreasing penalty function such that $p(0) = 0$. If the exterior penalty function, $p(\cdot)$, grows quickly enough outside of B, the optimal solution of (P) will also be optimal for (R). Furthermore, any optimal solution of (R) will provide an upper bound on the

optimum for (P), and this bound will in general be tighter than that obtained by simply optimizing z(**x**) over A.

In practice it can be difficult to find a penalty function which is an effective and efficient surrogate for the missing constraints. The effort required to tune the penalty function to a given problem instance or repeatedly calculate it during search may negate any gains in eventual solution quality. As noted by Siedlecki and Sklansky [35], much of the difficulty arises because the optimal solution will frequently lie on the boundary of the feasible region. Many of the solutions most similar to the genotype of the optimum solution will be infeasible. Therefore, restricting the search to only feasible solutions makes it difficult to find the schemata that will drive the population toward the optimum. Conversely, if too large a region is searched, much of the search time will be used to explore regions far from the feasible region and the search will tend to stall outside the feasible region.

Various families of functions p(·) and d(·) have been studied for GA optimization to dualize constraints. There have been two major approaches. The first is based only on the number of constraints violated, and is generally inferior to the second approach based on some distance metric from the feasible region [15, 33]. Richardson et al. [33] suggest that penalty functions can be effective surrogates for constraints, but that the effectiveness of the search can be quite sensitive to the form of the penalty function. Other authors [3, 20, 29] have successfully used a penalty based solely on distance of the infeasible solution from feasibility.

A variation of distance based penalty functions is to incorporate a dynamic aspect which (generally) increases the severity of the penalty for a given distance as the search progresses. This has the property of allowing highly infeasible solutions early in the search, while continually increasing the penalty imposed to eventually move the final solution to the feasible region. Recent uses of this approach include Joines and Houck [23] and Petridis and Kazarlis [31].

While incorporating distance along with the length of the search into the penalty function has been generally effective, these penalties ignore any other aspects of the search. In this respect,

they are not adaptive to the ongoing success (or lack thereof) of the search and cannot guide the search to particularly attractive regions or away from unattractive regions based on what has already been observed. A few authors have proposed making use of such search specific information. Siedlecki and Sklansky [35] discuss the possibility of self-adapting penalty functions, but their method is restricted to binary-string encodings with a single constraint, and involves considerable computational overhead. Bean and Hadj-Alouane [6, 17] propose penalty functions which are revised based on the feasibility or infeasibility of the best, penalized solution during recent generations. Their penalty function allows both an increase or a decrease of the imposed penalty during evolution and was demonstrated on multiple choice integer programming problems with one constraint. Smith and Tate [36, 39] used both search length and constraint severity feedback in their penalty function which provided the early basis for the general formulation presented in this paper.

The penalty formulation in this paper is distinct from previous research in several respects. First, it makes use of feedback during evolution concerning the current population. In this way it is self adapting to the degree of constraint of the problem. Second, it is formulated in a general way which accommodates continuous and discrete constraints and multiple constraints. Third, although all effective and efficient penalty functions must make use of some problem specific information, the adaptive penalty function has only one key problem specific parameter, a distance metric. Other user specified parameters can be optionally used to improve the efficiency of the search. The adaptive penalty function is demonstrated on two very different combinatorial problems: unequal-area layout and redundancy allocation. These problems include single and multiple constraints, and discrete and continuous constraints. They range from slightly constrained to extremely constrained. The adaptive penalty approach is shown to be robust to these various aspects, properties which suggest that little problem specific user tuning would be required to satisfactorily implement the proposed approach.

4. Proposed Adaptive Penalty Method

An ideal penalty function would be completely nonparametric, so that effective constrained optimization could be performed on any problem without the need for instance-specific knowledge. In practice, this is an unrealistic goal and more realistic objectives are to minimize the amount of instance-specific knowledge required to tune the penalty function, and to facilitate the translation of such knowledge directly into robust parameter settings. The proposed penalty function introduces the notion of a "near-feasibility threshold" (*NFT*) corresponding to a given constraint or set of constraints. Conceptually, the *NFT* is the threshold distance, either discrete or continuous, from the feasible region at which the user would consider the search as "getting warm." The penalty function will encourage the GA to explore within the feasible region and the *NFT*-neighborhood of the feasible region, and discourage search beyond that threshold. The definition of *NFT* is not only problem specific, it is constraint specific.

Additionally, a penalty method should learn to scale itself based on the severity of the constraints presented by a particular problem instance. For many problems, especially those with multiple constraints, the impact of the constraints may not be known. However, GA is an iterative search heuristic and a given GA realization provides access to an excellent source of information about the impact the constraints are having on the search (e.g., the difference between the best feasible solution and the best infeasible solution found by the GA). A wide gap between the best feasible value and the best infeasible value suggests a highly constrained problem, while nearly equal values suggest a very lightly constrained problem. This information can be used adaptively to adjust the severity with which the solutions are penalized for a specific degree of infeasibility, i.e. at a given distance from *NFT*.

Aggregation of these concepts lead to the development of a general penalized objective function $F_p(\mathbf{x})$ of the following form (for a maximization problem) with n constraints.

$$F_p(\mathbf{x}) = F(\mathbf{x}) - (F_{all} - F_{feas}) \sum_{i=1}^{n} \left(\frac{d_i(\mathbf{x}, B)}{NFT_i} \right)^{\kappa_i} \tag{1}$$

$F(\mathbf{x})$ is the unpenalized objective function value for solution $\mathbf{x}$, F_{all} denotes the unpenalized value of the best solution yet found, and F_{feas} denotes the value of the best feasible solution yet found.

The exponent κ_i is a user-specified severity parameter, and NFT_i is the near-feasible threshold described above for constraint i.

The effect of the penalty is to treat as equally attractive the best feasible solution yet found and very attractive infeasible solutions within close proximity to the feasible region. Specifically, the best feasible solution yet found is equivalent to an infeasible solution at distance NFT_i from the feasible region (for a single violated constraint) which has the best unpenalized objective function value yet encountered. As the search evolves, the $(F_{all} - F_{feas})$ term continually and adaptively adjusts the magnitude of the imposed penalty based on the search results.

The primary problem specific parameter, *NFT*, deserves further consideration. The general form of *NFT* is:

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

where NFT_o is some upper bound for *NFT*. Λ is a dynamic search parameter used to adjust *NFT* based on the search history. In the simplest case, Λ can be set to zero and a static *NFT* results. Λ can also be defined as a function of the search, for example, a function of the generation number, g, (i.e., $\Lambda = f(g) = \lambda g$). A positive value of λ results in a monotonically moving *NFT* and a larger λ more quickly decreases *NFT* as the search progresses. Analogies to this dynamic *NFT* concept have been seen in previous work [23, 31] where the penalty for infeasibility monotonically increased with the length of the search. Another variation is where Λ can be increased or decreased depending on the feasibility of the recent best solutions [6, 17].

Use of the *NFT* concept combined with the continuous feedback on problem constraint provided by the difference term of equation (1) provides a superior search strategy. This approach results in an adaptive and dynamic penalty approach which is general and can accommodate any number and severity of constraints, continuous and discrete distance metrics and makes use of any problem specific knowledge at hand to focus the search. For example, if *NFT* is ill defined *a priori*, it can be set at a large value initially with a positive constant λ used to iteratively guide the search to the feasible region. With problem specific information, a more

efficient search can take place by defining a tighter region or even static values of *NFT*. These aspects of the adaptive penalty function are demonstrated on two NP-hard combinatorial problems. The first application is the shape constrained version of the unequal area facility layout problem. For this problem, *NFT* is an integer value with a well defined *a priori* value and thus a static *NFT* is employed. Layout test problems range from exponentially unconstrained to highly constrained depending on the allowable shapes. The redundancy allocation problem complements the facility layout problem by demonstrating the penalty function with multiple constraints of disparate severity and magnitude. The constraints are a combination of discrete and continuous, and the selection of the *NFT* is less intuitive, so the dynamic *NFT* is used for effective optimization.

5. Adaptive Penalty Function Applications

5.1. Unequal-Area, Shape-Constrained Layout

The unequal-area facility layout problem with flow costs is defined as follows. A rectangular region with dimensions H×W is given, along with a collection of *n* "departments" of specified area, whose total area equals the size of the rectangular region. To each ordered pair of departments (*j,k*) is associated a traffic flow *F(j,k)*. The objective is to partition the region into one sector per department, of appropriate area, so as to minimize a cost function, C(Π), given by,

$$C(\Pi) = \sum_{j} \sum_{\substack{k \\ (j \neq k)}} F(j,k)\, \delta(j,k,\Pi) \tag{3}$$

where Π is the particular partitioning of the area into sectors and $\delta(j,k,\Pi)$ is the distance (using a prespecified metric) between the centroid of department *j* and the centroid of department *k* in the partition Π. This formulation first appeared in the operations research literature in 1963 [1]. To ensure that the optimal solution is realistic, it is necessary to impose some restrictions on the allowable shapes of the individual departments. A minimum side length constraint is established for each department, or equivalently, a maximum allowable free-orientation aspect ratio (α), where,

$$\alpha_i = \left(\frac{\max\{L_i, W_i\}}{\min\{L_i, W_i\}} \right) \tag{4}$$

and L_i is the length and W_i is the width of department i.

5.1.1. Encoding and Evolution Parameters

Tong [40] developed a constructive heuristic to restrict the departments to rectangles lying in a bay structure as shown in Figure 1. There are $2^{n-3}n!$ distinct flexible bay layouts. The flexible bay solutions were encoded in data structures with two distinct pieces (or "chromosomes"). The first piece carries a permutation of the integers 1 through n, where n is the number of departments being placed. This sequence represents the sequence of departments, bay by bay, read from left to right and top to bottom. The second piece contains an encoding of the number of bays in the layout and where the breaks between bays occur. This encoding is a straightforward generalization of the single-sequence encoding used for the quadratic assignment problem [38].

INSERT FIGURE 1 HERE

For breeding, a variant of uniform crossover was used, where each location in a child's sequence is occupied by the department in the corresponding location from one or the other parent. A simple repair is used to enforce feasibility of the sequence. The bay structure of the solution is taken without change from one parent or the other, with equal probability. Parents were selected based on a quadratic relationship of their ordinal ranking of their objective function as described in [39]. For mutation, three operators are used; one which splits an existing bay into two adjacent bays, one which merges two adjacent bays into one (by concatenation), and one which reverses a subsequence of departments on the sequence chromosome. Half of all mutations affect only the bay chromosome, and half affect only the sequence chromosome. Of those mutations altering the number of bays, half increase the number of bays and half decrease that number, so as to avoid bias in the tendencies of the population.

In each generation, one new child is produced by breeding and replaces the worst population member. This is a "steady state" GA where the population changes incrementally one by one, rather than the replacement of each entire generation. After breeding, individuals of the current population are then considered separately for mutation, with a fixed probability. Several exploratory runs were made to find effective values for the population size and mutation rate. The final parameters chosen were a population size of 10 and a mutation rate of 50%. The population is maintained in ranked order.

5.1.2. Adaptive Penalty Function

For this problem, a reasonable choice for *NFT* was known *a priori*. The degree of infeasibility of any one department is less important to the search than the number of departments which are infeasible. For example, a solution in which more than half of all departments are slightly infeasible in shape might require extensive modifications in order to yield a feasible solution, while a solution with one extremely infeasible department might be made feasible simply by shifting that department into an adjoining bay. Therefore, the penalty function is taken to be increasing not in the absolute amount of infeasibility of the solution, but rather solely in the number of infeasible departments, a discrete *NFT* metric in one constraint. Following from equation (1), the following functional form was used (for minimization).

$$ C_{ip}(\Pi) = C_i(\Pi) + (C_{feas} - C_{all}) \left(\frac{n_i}{NFT} \right)^{\kappa} \tag{5} $$

where n_i is the number of infeasible departments (distance metric), κ the severity parameter, $C_i(\Pi)$ the unpenalized objective function value and $C_{ip}(\Pi)$ the penalized objective function value.

5.1.3. Test Problems and Results

The original 20 department unequal-area facility layout problem of Armour and Buffa [1] was the largest problem studied. (A typographical error in the flow cost matrix originally published in [1] was corrected, following Scriabin and Vergin [34] and Huntley and Brown [21]). This problem involves partitioning a 200-foot by 300-foot rectangular area, and a minimum side length constraint for each department was added as shown in Table I. In general,

the tightest constraints (in terms of α_i) were placed on the smaller departments. These side-length constraints were difficult to satisfy, with only 1.6% of 100,000 randomly generated solutions being feasible. An *NFT* of 2 was used.

INSERT TABLE I HERE

Several smaller unequal area facility layout test problems were also studied. The smaller test problems [5, 18, 41] have 10 to 14 departments. For these smaller test problems, the minimum side lengths were fixed as closely to the original problem formulation as possible, and the penalty function was adjusted to compensate for the smaller number of departments by assigning the *NFT* to 1. For the van Camp et al. [41] problem, a minimum side length of 5 was used, as directly specified in that paper. For the Bazaraa [5] problems, a minimum side length of 1 was used, following van Camp et al.

Ten independent replications were run with each of five different exponent values (κ), with no penalty function (looking for the best feasible solution encountered during an unpenalized search), and with infeasible solutions immediately discarded (only feasible solutions were allowed in the population). The Armour and Buffa problem was terminated after 750 thousand solutions had been generated and the smaller test problems were terminated after 60 thousand solutions had been generated.

The performance of the adaptive penalty function was extremely encouraging. For the Armour and Buffa problem shown in Table II, the average cost of approximately 5500 was by far superior to their best unconstrained cost of 7862.1 [1], and far better than any randomly generated solutions observed. Tong [40] imposed a common minimum side length constraint of 10 feet on all departments, which is less restrictive than this constraint set. Running ten replications with a linear penalty function ($\kappa = 1$), using Tong's constraints, yielded a best solution of 4389.7, which is a cost reduction of more than 15%. Even more encouraging was the robustness of the method, with respect to both severity of penalty function as measured by κ and random number stream. The three best penalty functions showed average solution values within 0.3% of each other, with standard deviations of less than 5% of the mean (across the ten

replications). The overall mean solution value over these 30 runs was 13.5% better than the mean solution found by the all-feasible GA runs, and more than 30% better than the mean unpenalized GA solution. Figures 2 and 3 show graphically the effects of the penalty function on both final solution quality and speed of evolution convergence.

<div align="center">INSERT TABLE II HERE</div>

<div align="center">INSERT FIGURES 2 AND 3 HERE</div>

Additionally, the adaptive penalty approach dominated all other published results of the smaller test problems as shown in Table III, even in its worst of 10 runs. The best solutions improved upon the previously published best solutions by 16.3%, 16.2% and 19.9% respectively.

<div align="center">INSERT TABLE III HERE</div>

5.2. Redundancy Allocation Problem

The redundancy allocation problem serves as a valuable complement to the facility layout problem because it is fundamentally different. For this problem, there are multiple constraints (for system weight and cost), which are measured on a continuous scale. Also, selection of an appropriate *NFT* is not as obvious, and a dynamic *NFT* is employed. The redundancy allocation problem is stated as follows: the system (series-parallel as illustrated in Figure 4) is partitioned into a specific number of subsystems, *s*, and for each subsystem, there are different component types available with varying costs, reliabilities, weights and possibly other characteristics. It is often required to use components in parallel within each subsystem because of the necessary subsystem function (i.e., a minimum number of components, *k*, is needed) or the need to increase overall system reliability. The problem is then to select the optimal combination of component type and levels of redundancy for each subsystem to collectively meet reliability and weight constraints at a minimum cost, or alternatively, to maximize reliability given cost and weight constraints:

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

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

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

$$k_i \leq \sum_{j=1}^{m_i} x_{ij} \leq n_{max,i} \quad \forall \ i = 1,2,...,s$$

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

where,

C	=	cost constraint
W	=	weight constraint
$\mathbf{x_i}$	=	$(x_{i1}, x_{i2}, ..., x_{i,mi})$
x_{ij}	=	number of the jth component used in subsystem i
n_i	=	total number of components used in subsystem i
	=	$x_{i1} + x_{i2} + ... + x_{i,mi}$
$n_{max,i}$	=	maximum number of components used in subsystem i (specified)
$R_i(\mathbf{x_i}|k_i)=$		reliability of subsystem i, given k_i
$C_i(\mathbf{x_i})$	=	total cost of subsystem i
$W_i(\mathbf{x_i})$	=	total weight of subsystem i

5.2.1. Encoding and Evolution Parameters

Each possible solution to the redundancy allocation problem is a collection of n_i parts in parallel ($k_i \leq n_i \leq n_{max,i}$) for s different subsystems. The n_i parts can be chosen in any combination from among the m_i available components. The m_i components are indexed in descending order in accordance with their reliability (i.e., 1 representing the most reliable, etc.). The solution encoding is an integer vector representation with $\sum n_{max,i}$ positions. Each of the s subsystems are represented by $n_{max,i}$ positions with each component listed according to their reliability index. An index of $m_i + 1$ is assigned to a position where an additional component was not used (i.e., $n_i < n_{max,i}$). The subsystem representations are then placed adjacent to each other to complete the vector representation. As an example, consider a system with $s = 3$, $m_1 = 5$, $m_2 = 4$, $m_3 = 5$ and $n_{max,i}$ predetermined to be 5 for all i. The following example,

$$v_k = (1\ 1\ 6\ 6\ 6\ |\ 2\ 2\ 3\ 5\ 5\ |\ 4\ 6\ 6\ 6\ 6)$$

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

Previous experimentation [8] indicated that a population size of 40 converged quickly and produced good solutions. The selection and crossover operators used were the same as for the facility layout problems. For a solution vector chosen for mutation, each integer value is changed with probability equal to a preselected mutation rate. If selected to be mutated, it is changed to an index of $m_i + 1$ (no component) with 50 % probability and to a randomly chosen component, from among the m_i choices, with 50 % probability. After crossover breeding, the p best solutions from among the previous generation and the new child vectors were retained to form the next generation, a form of $\mu + \lambda$ replacement adapted from evolutionary strategies (see for example [4]). Mutation was performed after culling inferior solutions from the population.

5.2.2. Adaptive Penalty Function

The redundancy allocation problem is formulated with two independent constraints so the penalty function became a linear summation as shown in equation (1) with $n = 2$. Selection of NFT values for weight and cost constraints were less intuitive than in the unequal-area layout problem because the concept of "near feasible" is relative to the particular constraint and the characteristics of the search space. Therefore NFT was relaxed from a static to a dynamic value with g, the generation number.

The penalty function used to analyze the reliability maximization problem is as follows,

$$R_{ip}(\mathbf{x}_i) = R_i(\mathbf{x}_i) - (R_{all} - R_{feas})\left(\left(\frac{\Delta w_i}{NFT_w}\right)^\kappa + \left(\frac{\Delta c_i}{NFT_c}\right)^\kappa\right) \tag{6}$$

where NFT_c and NFT_w are the near feasible threshold for the cost and weight constraints respectively and took dynamic formations:

$$NFT = \frac{NFT_o}{1 + \lambda g} \qquad \text{where } \lambda = 0.04, NFT_{co} = 100 \text{ and } NFT_{wo} = W/1.3. \tag{7}$$

While an effective value of λ took brief experimentation to establish, the values of NFT_{co} and NFT_{wo} chosen initially worked well. The main concern in choosing values of these latter two parameters is that they are relatively large so that enough of the infeasible region is considered during early phases of the search.

To make comparisons, the problem was also solved with a static NFT ($\lambda = 0$), with dynamic NFT_o's defined as 5%, 3% and 1% of each constraint, and by allowing only feasible solutions in the population. Δw_i and Δc_i in equation (6) represent the magnitude of any constraint violations for the ith solution vector. The other parameters are as they were previously defined. For the facility layout problem, the penalty function was demonstrated to be largely insensitive to the specific value chosen for the severity parameter (κ), although the results with the lowest coefficient of variation was observed for κ equal to 2. For the redundancy allocation problem, a value of κ of 2 was used, resulting in a penalty of Euclidean distance from the infeasible solution to the feasible region over all constraints.

5.2.3. Test Problems and Results

The test problems studied are the original problem posed by Fyffe, Hines and Lee [12] in 1968 and the 33 problem variations from Nakagawa and Miyazaki [28]. The problem objective is to maximize reliability for a system with 14 subsystems with three or four component choices for each, and $k = 1$ for all subsystems. For each component alternative, there is a specified reliability, cost and weight. For the original problem, the cost constraint is 130 and the weight constraint is 170. For the 33 problem variations, the cost constraint is maintained at 130 and the weight constraint is decreased incrementally from 191 to 159, which increases the severity of the constraint. The size of the search space is larger than 7.6 x 10^{33}.

Fyffe, Hines and Lee [12] used a dynamic programming approach with a Lagrangian multiplier to accommodate the weight constraint within the objective function. Nakagawa and Miyazaki [28] showed that the use of a Lagrangian multiplier is often inefficient. They deployed a surrogate constraint approach combining the cost and weight constraints into one, which must be iteratively updated with different surrogate multipliers. In both instances, the

formulations necessitated that only identical components could be placed in parallel. That is, a higher reliability component could not be placed in parallel with a functionally similar, but lower reliability component, although this commonly occurs in practice. Therefore, the "optimal solution" found only pertains to their restricted search space, and in fact, they generally did not find the global optimal solution to the problem.

The results are presented in Tables IV and V. For each table, the results are pooled from the 33 different problems with 10 runs of each (each table entry is the mean of 330 GA runs). The results in Table IV present, for each of the five *NFT* alternatives, the percentage of trials where the GA converged to a final feasible solution. Additionally, the table presents the percentage of problems where the best feasible solution encountered during the GA search was superior to the dynamic programming results from Nakagawa and Miyazaki. This data is reported for both the best solution (of the 10 trials) and for all GA trials. Table V presents the best feasible solution, average feasible solution and the standard deviation averaged over all 33 problems.

The results from the tables are very enlightening. The results from this problem clearly indicate that use of the proposed adaptive penalty function (with any of the *NFT* criteria) is preferable to a GA which considers only feasible solutions and discards infeasible solutions as they are generated. Another interesting result from the tables is the comparisons of different *NFT* criteria. If a relatively large static *NFT*, i.e., 5 %, was used the GA thoroughly searched the infeasible region and, in doing so, often found good feasible solutions but ultimately converged to an infeasible solution in greater than 98% of GA trials. Conversely, if the *NFT* was relatively small (i.e., 1 %), the GA converged to a feasible solution in all cases but the relative solution quality was poor. An intermediate *NFT* value (i.e., 3 %) was able to balance both the benefits and deficiencies but was clearly inferior to the dynamic *NFT* considering all comparison criteria. The GA with a dynamic *NFT* consistently converged to a final feasible solution and had statistically significant superior solutions at $\alpha = 0.001$ over other penalty strategies.

Furthermore, the dynamic *NFT* had less sensitivity to random number seed and problem instance as evidenced by the smaller coefficient of variation.

6. Conclusions

There are many optimization problems for which it is difficult to find any feasible solution, much less near-optimal feasible solutions. Adaptive penalty-guided genetic search shows promise as an optimization method for such highly constrained problems. Using both feedback from the GA search along with any problem specific information at hand provides an adaptive, dynamic penalty which is robust and effective. The quality of the solutions generated does not seem to be particularly sensitive to the precise penalty parameters used, the random number seed, the degree or numbers of constraints, or the particular problem instance, so that no extensive tuning of the penalty function is necessary.

For combinatorial optimization, the adaptive penalty method would be best suited to problems which arise as well-structured optimization problems with a small number of less well-behaved side constraints. An ideal GA implementation for such a problem would use encodings and operators guaranteed to preserve feasibility with respect to the primary constraints, and adaptive penalty functions to guide the search for feasibility with respect to the side constraints. For constraints where identification of the *NFT* is not intuitive based on problem structure, the results here indicate that a dynamic *NFT* produces the best results and consistently converges to a feasible solution. Where better *a priori* knowledge is at hand, a confined or static *NFT* can be more efficient. This general approach should also be applicable to continuous optimization problems.

References

[1]   G. C. Armour and E. S. Buffa, 1963, *A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities*, **Management Science 9**, 294-309.

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

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

[4]   T. Baeck and H.-P. Schwefel, 1993, *An Overview of Evolutionary Algorithms for Parameter Optimization*, **Evolutionary Computation 1**, 1-23.

[5]   M. S. Bazaraa, 1975, *Computerized Layout Design: A Branch and Bound Approach,* **AIIE Transactions 7**, 432-438.

[6]   J. C. Bean and A. B. Hadj-Alouane, 1992, *A Dual Genetic Algorithm for Bounded Integer Programs*, University of Michigan Technical Report 92-53, to appear in **R.A.I.R.O. - R.O.**

[7]   J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. S. Richards, 1991, *Distributed Genetic Algorithms for the Floorplan Design Problem*, **IEEE Transactions on CAD 10**, 483-492.

[8]   D. W. Coit and A. E. Smith, 1996, *Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm*, **IEEE Transactions on Reliability**, in print.

[9]   K. A. DeJong, 1975, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral Dissertation, University of Michigan, Ann Arbor, MI.

[10] K. A. DeJong and W. M. Spears, 1989, *Using Genetic Algorithms to Solve NP-Complete Problems*, **Proceedings of the Third International Conference on Genetic Algorithms**, June 4-7, 1989, 124-132.

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

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

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

[14] D. E. Goldberg, 1983, *Computer-aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning*, Doctoral Dissertation, University of Michigan, Ann Arbor, MI.

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

[16] D. E. Goldberg and R. Lingle, 1985, *Alleles, Loci, and the Traveling Salesman Problem*, **Proceedings of the International Conference on Genetic Algorithms and Their Applications**, J. J. Grefenstette (ed.), 154-159.

[17] A. B. Hadj-Alouane and J. C. Bean, 1992, *A Genetic Algorithm for the Multiple-Choice Integer Program*, University of Michigan Technical Report 92-50, to appear in **Operations Research**.

[18] M. M. D. Hassan, G. L. Hogg and D. R. Smith, 1986, *SHAPE: A Construction Algorithm for Area Placement Evaluation*, **International Journal of Production Research 24**, 1283-1295.

[19] J. H. Holland, 1975, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI.

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

[21] C. L. Huntley and D. E. Brown, 1991, *A Parallel Heuristic for Quadratic Assignment Problems*, **Computers and Operations Research 18**, 275-289.

[22] P. Jog, J. Y. Suh and D. Van Gucht, 1991, *Parallel Genetic Algorithms Applied to the Traveling Salesman Problem*, **SIAM Journal of Optimization 51**, 515-529.

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

[24] G. E. Liepins, M. R. Hilliard, J. Richardson and M. Palmer, 1990, *Genetic Algorithm Applications to Set Covering and Traveling Salesman Problem*, in **OR/AI: The Integration of Problem Solving Strategies**.

[25] G. E. Liepins and W. D. Potter, 1991, *A Genetic Algorithm Approach to Multiple-Fault Diagnosis*, in **Handbook of Genetic Algorithms** (L. Davis, editor), Van Nostrand Reinhold, New York.

[26] Z. Michalewicz, 1992, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin.

[27] Z. Michalewicz and C. Z. Janikow, 1991, *Handling Constraints in Genetic Algorithms*, **Proceedings of the Fourth International Conference on Genetic Algorithms**.

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

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

[30] D. Orvosh and L. Davis, 1994, *Using a Genetic Algorithm to Optimize Problems with Feasibility Constraints*, **Proceedings of the First IEEE Conference on Evolutionary Computation**, 548-553.

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

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

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

[34] M. Scriabin and R. Vergin, 1975, *Comparison of Computer Algorithms and Visual Based Methods for Plant Layout*, **Management Science 22**, 172-181.

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

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

[37] R. H. Storer, S. D. Wu and R Vaccari, 1992, *New Search Spaces for Sequencing Problems With Application to Job Shop Scheduling*, **Management Science 38**, 1495-1509.

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

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

[40] X. Tong, 1991, *SECOT: A Sequential Construction Technique for Facility Design,* Doctoral dissertation, University of Pittsburgh, Pittsburgh, PA.

[41] D. J. van Camp, M. W. Carter and A. Vannelli, 1991, *A Nonlinear Optimization Approach for Solving Facility Layout Problems*, **European Journal of Operational Research 57**, 174-189.

[42] D. Whitley, T. Starkweather and D. Shaner, 1991, *The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination*, **Handbook of Genetic Algorithms** (L. Davis, editor), Van Nostrand Reinhold, New York, 350-372.

Table I. Department Specifications for Armour and Buffa Problem.

Dept.	Area	Minimum Side Length	Dept.	Area	Minimum Side Length
A	2700	15	L	6000	25
B	1800	15	M	4200	20
C	2700	15	N	1800	15
D	1800	15	P	2400	15
E	1800	15	R	2700	15
F	1800	15	S	7500	30
G	900	15	T	6400	25
H	900	15	U	4100	20
J	900	15	V	2700	15
K	2400	15	W	4500	20

Table II. Performance of Different Penalties for the Armour and Buffa Problem.

Evolution Method	Best of 10 Seeds	Mean of 10 Seeds	CV (%) of 10 Seeds
Random*	8705.6	12508.3	10.46
No Penalty	7499.9	8319.3	5.76
All Feasible	5687.8	6393.7	8.14
$\kappa = 0.5$	5305.0	5658.1	5.23
$\kappa = 1.0$	5140.6	5511.3	4.47
$\kappa = 2.0$	5231.6	5512.8	3.47
$\kappa = 3.0$	5278.3	5529.6	4.00
$\kappa = 5.0$	5225.5	5679.0	6.94

* Random results are for 100,000 randomly generated solutions.

Table III. Results for the van Camp and Bazaraa Test Problems with κ=3 Over 10 Runs.

Solution	van Camp	Bazaraa	Bazaraa
GA Best	20472.2	8861.0	5080.1
GA Mean	21745.7	9509.9	5318.9
GA Worst	23612.6	9894.3	5506.8
van Camp's NLT [41]	24445.0	11910.0	6875.0
Bazaraa [5]	- -	14079.0	8170.5
Hassan's PLANET [18]	- -	11664 - 11808	6399 - 6480
Hassan's SHAPE [18]	- -	10578 - 11140	6339 - 6462

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

Comparison	Only Feasibles	5% *NFT*	3% *NFT*	1% *NFT*	Dynamic *NFT*
% Feasible	100.00%	1.21%	80.00%	100.00%	100.00%
% Best > N&M	0.00%	63.64%	45.45%	9.09%	81.82%
% Total > N&M	0.00%	27.27%	15.45%	0.91%	44.85%

Table V. Comparison of Final Solutions for the 33 Reliability Optimization Problems Over 10
Runs of Each.

| GA Performance	Only Feasibles	5% *NFT*	3% *NFT*	1% *NFT*	Dynamic *NFT*
Best Solution	0.97096	0.97337	0.97302	0.97180	0.97366
Average Solution	0.96894	0.97239	0.97167	0.96956	0.97288
Coeff. of Variation (%)	0.14933	0.08218	0.11305	0.15847	0.06573
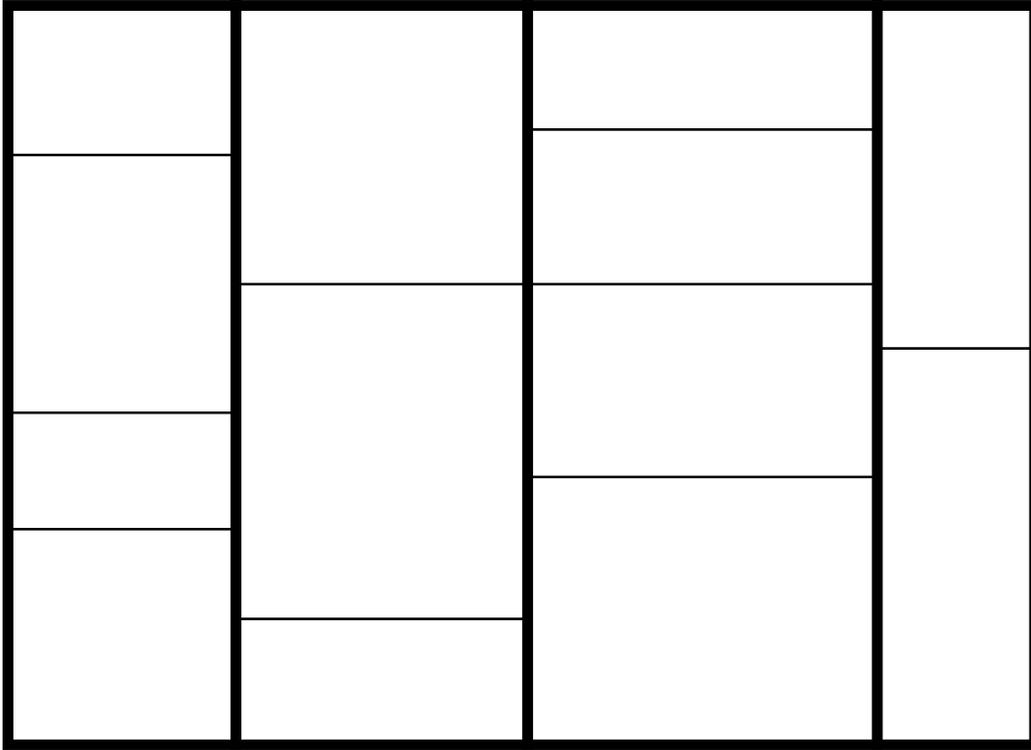
Figure 1. Typical Flexible Bay Layout.
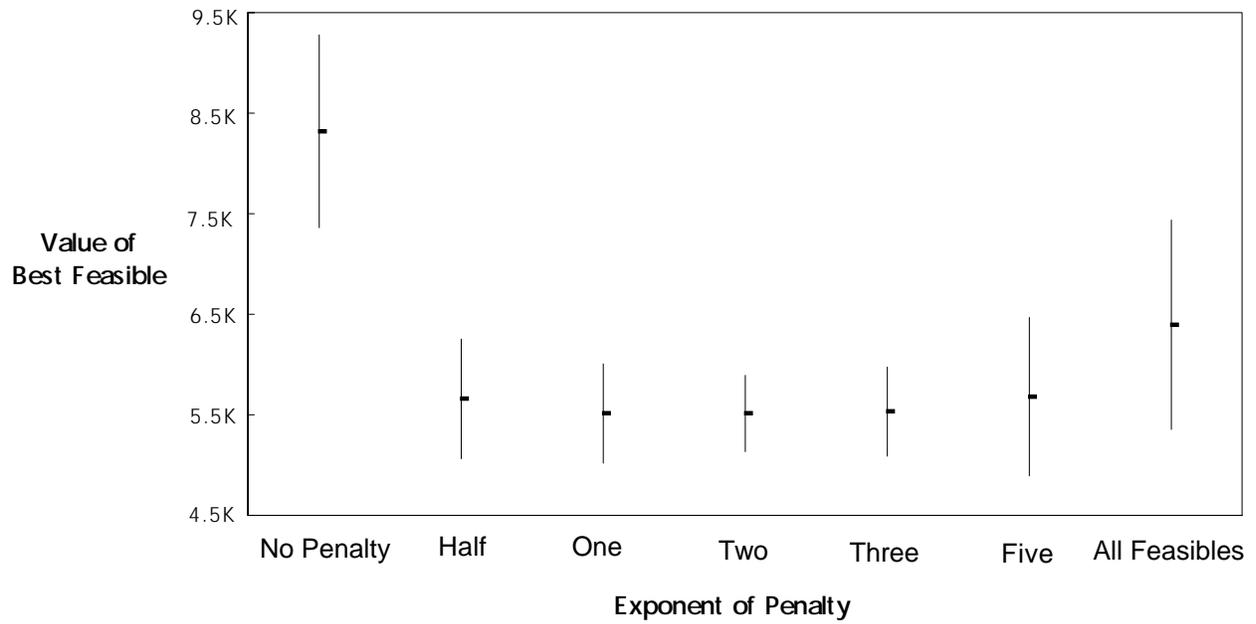
Figure 2. Mean and Two Standard Deviation Spread of Best Solution to the Armour and Buffa Problem Over 10 Seeds.
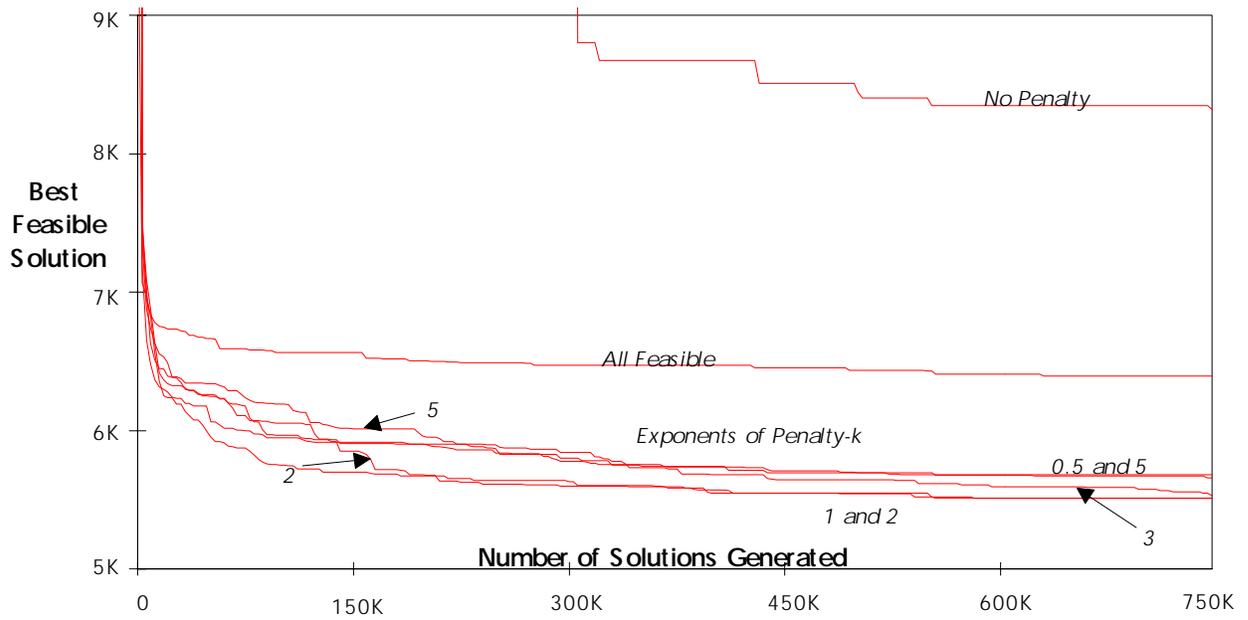
Figure 3. Mean Best Solution Over 10 Seeds During Evolution for the Armour and Buffa
Problem.
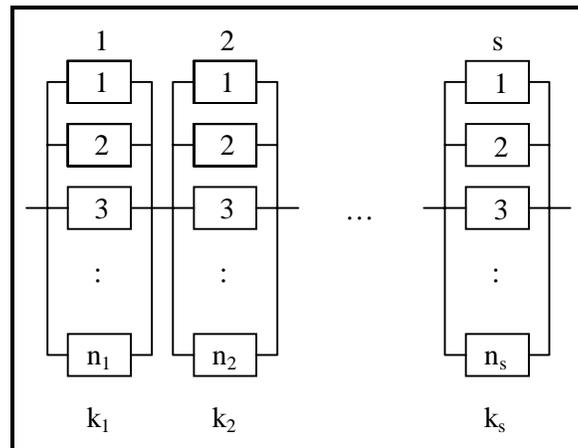
Figure 4. Series-Parallel System Configuration.