TECHNICAL RESEARCH REPORT

A Genetic Algorithm for a Minimax Network Design Problem

by Jeffrey W. Herrmann

T.R. 99-29

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the Glenn L. Martin Institute of Technology/A. James Clark School of Engineering. It is a National Science Foundation Engineering Research Center.

Web site http://www.isr.umd.edu

A Genetic Algorithm for a Minimax Network Design Problem

Jeffrey W. Herrmann
Department of Mechanical Engineering and Institute for Systems Research
University of Maryland
College Park, Maryland 20742

May 12, 1999

Abstract

This paper considers the problem of designing a network to transport material from sources of supply to sites where demand occurs. However, the demand at each site is uncertain. We formulate the problem as a robust discrete optimization problem. The minimax objective is to find a robust solution that has the best worst-case performance over a set of possible scenarios. However, this is a difficult optimization problem. This paper describes a two-space genetic algorithm that is a general technique to solve such minimax optimization problems. This algorithm maintains two populations. The first population represents solutions. The second population represents scenarios. An individual in one population is evaluated with respect to the individuals in the other population. The populations evolve simultaneously, and they converge to a robust solution and a worst-case scenario. Experimental results show that the two-space genetic algorithm can find robust solutions to the minimax network design problem. Since robust discrete optimization problems occur in many areas, the algorithm will have a wide variety of applications.

KEYWORDS: genetic algorithms, robust discrete optimization, network design.

1 Introduction

This paper discusses the problem of designing a transportation network under uncertainty. To find good solutions, we use a novel genetic algorithm that is a general approach for solving minimax optimization problems.

1.1 Network Design Problem

Consider the following network design problem. There are p facilities that can supply the product, and each facility's capacity is known and fixed. There are m sites that need the product. Each site's demand is uncertain. Before the demand is known, the decision-maker must assign, to each of the pm routes, x_{ij} vehicles to move material from facility i to site j.

Each vehicle assigned to the route incurs a fixed cost c_{ij}. All vehicles have the same fixed capacity. After the demand is known, the decision-maker must decide how much material to move from facility i to site j. The amount moved is constrained by the total supply at facility i and the number of vehicles on that route. If the amount moved to a site from the supply facilities does not meet demand, then additional material can be purchased and delivered to the site at a premium. Equivalently, one can consider the problem of building pipelines of different capacities between supply facilities and demand sites. The problem will be formulated more explicitly in Section 2.

Note that the deterministic version of this problem is the transportation problem [1], which is an important model for many network flow and network design problems. Mulvey, Vanderbei, and Zenios [19] consider a similar but more complex problem under the assumption that the probabilities associated with different demand scenarios is known.

1.2 Decision-making Under Uncertainty

Making decisions under uncertainty is a difficult problem. However, accepting and structuring uncertainty can lead to effective decision-making. Consider the problem of selecting some action that minimizes the costs that will be expended. The costs depend on what happens in the future. It is useful to model the possibilities as scenarios.

Some scenario will occur. It may be possible, for each scenario s, to determine the probability p_s that scenario s occurs. If so, the decision-maker could seek the solution (action) that minimizes the expected cost. Let X be the set of solutions (possible actions). Let S be the set of scenarios (possible futures). Let $F(x, s)$ be the cost of solution $x \in X$ if scenario $s \in S$ occurs. Then, the decision-maker is trying to solve the following problem:

$$\min_{x \in X} \sum_{s \in S} p_s F(x, s)$$

Such stochastic programming problems have many applications. See Birge & Louveaux [2] for an introduction to stochastic programs and solution techniques. Some stochastic programs consider the risk associated with a decision and seek to minimize risk or establish the tradeoff

between risk and expected performance. Mulvey, Vanderbei, and Zenios [19] present a general method of handling risk, and they use the term "robust optimization" to describe a model that includes higher moments of the random performance and a penalty function for infeasible solutions.

An alternative to the expected value is the minimax criterion, which evaluates the worst-case performance of each solution:

$$\min_{x \in X} \max_{s \in S} F(x, s)$$

The minimax criterion may be favored when a large cost in any scenario is extremely undesirable, as in medical equipment control (where the cost represents some measure of patient health, and a large cost implies that the patient dies [22]). Mulvey, Vanderbei, and Zenios [19] mention the minimax criterion as a standard form that is a special case of their general robust optimization model.

The minimax criterion is important to risk-averse decision-makers interested in the distribution of a solution's performance. This is especially true for decisions of a unique nature (e.g. facility layout or capacity expansion) where the decision-maker will not see any long-run averages. Of course, other measures of risk describe the distribution of performance also. These include variance [15], expected downside risk [8, 14], and the probability of poor performance [17].

The minimax criterion is also useful when determining the scenario probabilities is impossible or extremely expensive. In such cases, one cannot calculate the expected performance or the other measures of risk mentioned above.

While the minimax criterion can yield solutions that are overly conservative (or pessimistic) and thus accept unnecessary costs in more common, non-extreme scenarios, this behavior can be avoided by using a minimax regret criterion:

$$\min_{x \in X} \max_{s \in S} F(x, s) - F^*(s)$$

where $F^*(s) = \min_{x \in X} F(x, s)$.

The minimax criterion has been employed in many applications. Consider, for instance, the following examples. (See also Yu [26], who provides additional examples.)

Sebald and Schlenzig [22] use the minimax criterion to guide the search for good controller parameter settings in closed-loop, adaptive control of blood pressure for patients in cardiac surgery. The blood pressure response is uncertain since it depends on unknown values of certain patient characteristics. The goal is to design a controller that keeps blood pressure within acceptable limits over the entire range of possible patients.

Tang, Man, and Gu [23] evaluate distillation column controllers under a set of different plant characteristics and attempt to find a controller that performs well in all scenarios.

Chang, Wang, and Lock [3] address the problem of designing power systems that have acceptable worst-case performance over a range of operating conditions.

Rosenblatt and Lee [21] use the minimax regret criterion to compare different plant layouts, since the total material handling cost depends upon the demand for each product, which is uncertain.

Determining the worst-case performance of a solution can be a difficult problem itself. For instance, Dahl, Meeraus, and Zenios [4] present an optimization model that analyzes the worst-case performance of a given portfolio of options, if the interest rates and futures price volatility for each investment are uncertain. Due to the complex objective function, their model will likely have locally optimal scenarios and will be very difficult to solve.

Kouvelis and Yu [13] describe a robust discrete optimization framework that seeks to identify decisions that will perform well under any outcome. They use minimax criteria to identify a robust decision (or solution) as the one that has the best worst-case performance.

In this robust discrete optimization framework, scenarios are used to structure uncertainty. Decision-makers must use their intuition about the decision environment to define a set of scenarios. Each scenario in this set represents a possible outcome. That is, the scenario occurs with some positive but unknown probability.

In general, the minimax optimization problems are more difficult to solve than the de-

terministic versions that have no uncertainty [13]. The minimax optimization versions of many polynomially solvable optimization problems are NP-hard, although some easily solvable cases do exist. (For other results in the more general area of minimax theory, see Du and Pardalos [7].)

Kouvelis and Yu [13] describe a branch-and-bound algorithm that uses surrogate relaxation to generate bounds, and they use this procedure to solve four robust discrete optimization problems. However, there are no general techniques for solving robust discrete optimization problems.

Some authors have proposed genetic algorithms and evolutionary algorithms for specific problems with uncertainty. (For more information on genetic algorithms, see, for example, [6, 9, 12, 18].) Loraschi and Tettamanzi [14] use an evolutionary algorithm to select portfolios with small expected downside risk.

Taranto and Falcao [24] consider robust decentralized power system damping controllers and use a genetic algorithm to find designs that maximize the sum of the spectrum damping ratio over all operating conditions. Marrison and Stengel [17] measure compensator robustness as the probability that the system will behave unacceptably. They use a genetic algorithm to find parameter values that minimize this probability. For the distillation column controller problem mentioned above, Tang, Man, and Gu [23] use a genetic algorithm to solve the problem of minimizing the number of violated constraints for a given scenario. For a problem with four scenarios, they formulate a multiple objective problem and use the genetic algorithm to find non-dominated solutions. The authors note that, when there is a large set of scenarios, the problem of efficiently determining a solution's worst-case performance still remains.

Tsutsui and Ghosh [25] present a genetic algorithm that finds solutions whose performance is insensitive to small perturbations to the solution values. The algorithm evaluates an individual by creating a random perturbation. The individual's fitness is based on the objective function at the perturbed point. They present results for some functions on one-

and two-dimensional search spaces.

Sebald and Schlenzig [22] use a genetic algorithm to search for controller parameter values that minimize the worst-case performance. To evaluate a single combination of controller parameter values, they must find the worst-case plant for that controller. To do this, they use another genetic algorithm to search the set of possible plants.

Closely related to the present work are previous studies of co-evolutionary genetic algorithms. We will briefly describe two here. Hillis [11] uses a genetic algorithm to find optimal sorting networks. Initially, each network is tested against a large set of test cases. The goal is to find a network that passes all of the test cases. He then introduces a second population of "parasites" to improve the search. The algorithm pairs each sorting network with a parasite, which is a set of test cases. A sorting network's performance is the number of the corresponding parasite's test cases that it passes. A parasite's performance is the number of test cases that the sorting network fails. The population of parasites evolves so that difficult test cases survive. This, in turn, leads to better sorting networks.

Paredis [20] uses a coevolutionary genetic algorithm to seek neural nets that can accurately classify a set of 200 points. Because testing each individual on all 200 examples requires excessive computational effort, the coevolutionary GA tests each individual against one of the more difficult examples. The 200 examples are ranked based on their difficulty, and this ranking changes as the population of neural nets changes. In addition, Paredis allows the neural nets to learn when it incorrectly classifies an example.

This paper proposes a two-space genetic algorithm as a general technique to solve minimax optimization problems. This algorithm maintains two populations. The first population represents solutions. The second population represents scenarios. An individual in one population is evaluated with respect to all of the individuals in the other population. This causes the algorithm to converge to a robust solution and its worst-case scenario. Since minimax optimization problems occur in many areas, the algorithm will have a wide variety of applications. To illustrate the algorithm's potential, we use the algorithm to solve a

distribution network design problem with uncertain demands.

The remainder of this paper is structured as follows: Section 2 describes the minimax network design problem formulation. Section 3 presents the two-space genetic algorithm. Section 4 presents the experimental results. Section 5 concludes the paper by identifying topics for future research that could extend the algorithm's utility.

2 Problem Formulation

Let us introduce some additional notation before formulating the problem more explicitly. Let X be the set of all possible vehicle assignments: $x_{ij} \geq 0$ and integer $\forall i = 1, \ldots, p$, $j = 1, \ldots, m$. Let all quantities be expressed in multiples of vehicle capacity. Let $S_1, \ldots, S_p$ be the supply available at each of the p facilities. Let r be the premium cost of delivering material that the assigned vehicles cannot. Let s be a scenario that has some positive but unknown probability of occurence. Let S be the set of all scenarios. Let $D_1^s, \ldots, D_m^s$ be the demand at each of the m sites if scenario s occurs. Let y_{ij}^s (for $i = 1, \ldots, p$ and $j = 1, \ldots, m$) be the amount of material moved from facility i to site j if scenario s occurs. For $j = 1, \ldots, m$, let z_j^s be the extra material needed at site j if scenario s occurs.

2.1 Minimax formulation

With this notation we can formulate the robust discrete optimization problem P as follows:

$$\min_{x \in X} \max_{s \in S} F(x, s)$$

where

$$F(x, s) = \min_{y^s, z^s} \sum_{i=1}^{p} \sum_{j=1}^{m} c_{ij} x_{ij} + r \sum_{j=1}^{m} z_j^s$$

subject to

$$\begin{aligned} y_{ij}^s &\leq x_{ij}, \forall i = 1, \ldots, p, j = 1, \ldots, m, s \in S \\ \sum_{j=1}^{m} y_{ij}^s &\leq S_i, \forall i = 1, \ldots, p, s \in S \\ \sum_{i=1}^{p} y_{ij}^s + z_j^s &= D_j^s, \forall j = 1, \ldots, m, s \in S \end{aligned}$$

$$z_j^s \geq 0, \forall j = 1, \ldots, m, s \in S$$

$$y_{ij}^s \geq 0, \forall i = 1, \ldots, p, j = 1, \ldots, m, s \in S$$

Note, that since $\sum_{i=1}^{p} \sum_{j=1}^{m} c_{ij} x_{ij}$ is independent of the scenario (because those decisions are made ahead of time), evaluating $F(x, s)$ requires minimizing $\sum_{j=1}^{m} z_j^s$, which is equivalent to maximizing $\sum_{j=1}^{m} \sum_{i=1}^{p} y_{ij}^s$. This subproblem is a maximal flow problem, which can be solved by a labeling method [1].

2.2 Scenarios

In general, the scenarios do not necessarily follow any pattern. However, in this work, we will assume that there exist a number of factors that affect the demand at each site. Each factor f_k has a strength that ranges from 0 to 1. A scenario s is a combination of factor strengths $(f_1^s, \ldots, f_q^s)$. The space of scenarios is the space of all factor strengths. Under scenario s, the demand at site j is D_j^s:

$$D_j^s = D_j^0 + \sum_{k=1}^{q} d_{jk} f_k^s$$

Note that d_{jk} could be negative. However, D_j^s must be non-negative.

With this structure, we can show that, for any solution, the worst-case scenario is an extreme point scenario: each $f_k^s = 0$ or 1. See the Appendix for a proof.

2.3 Worst-case scenario

For a given solution, increasing the demand at any site cannot reduce the required flow. Thus, if there is a scenario $t \in S$ such that, for any other scenario $s \in S$, $D_j^t \geq D_j^s$ for all j, then t is a worst-case scenario for any solution. That is, $\max_{s \in S} F(x, s) = F(x, t)$ for all $x \in X$. If all $d_{jk} \geq 0$ and t is the scenario with $f_k^t = 1$, $k = 1, \ldots, q$, then scenario t is this worst-case scenario.

In this case, problem P reduces to $\min_{x \in X} F(x, t)$. So, $x_{ij} = y_{ij}^t$ for all i, j must be an

optimal solution. Then we can formulate problem P_t as follows:

$$\min_{x \in X} \sum_{i=1}^{p} \sum_{j=1}^{m} c_{ij} x_{ij} + r \sum_{j=1}^{m} z_j^t$$

subject to

$$\begin{aligned}
\sum_{j=1}^{m} x_{ij} &\leq S_i, \forall i = 1, \ldots, p \\
\sum_{i=1}^{p} x_{ij} + z_j^t &= D_j^t, \forall j = 1, \ldots, m \\
z_j^t &\geq 0, \forall j = 1, \ldots, m \\
x_{ij} &\geq 0, \forall i = 1, \ldots, p, j = 1, \ldots, m
\end{aligned}$$

2.4 Linear Programming Formulation

One can create the following linear programming formulation P_L as an another form of problem P:

$$\min_{x \in X} \sum_{i=1}^{p} \sum_{j=1}^{m} c_{ij} x_{ij} + rw$$

subject to

$$\begin{aligned}
w &\geq \sum_{j=1}^{m} z_j^s, \forall s \in S \\
y_{ij}^s &\leq x_{ij}, \forall i = 1, \ldots, p, j = 1, \ldots, m, s \in S \\
\sum_{j=1}^{m} y_{ij}^s &\leq S_i, \forall i = 1, \ldots, p, s \in S \\
\sum_{i=1}^{p} y_{ij}^s + z_j^s &= D_j^s, \forall j = 1, \ldots, m, s \in S \\
z_j^s &\geq 0, \forall j = 1, \ldots, m, s \in S \\
y_{ij}^s &\geq 0, \forall i = 1, \ldots, p, j = 1, \ldots, m, s \in S
\end{aligned}$$

3 Two-space Genetic Algorithm

We propose a two-space genetic algorithm for finding good solutions to robust optimization problems. This section will first present the two-space genetic algorithm and then discuss

the motivation behind it.

The two-space genetic algorithm maintains two distinct populations: P_1 has individuals that represent solutions in X, and P_2 has individuals that represent solutions in S. For a solution x in P_1, the objective function $h(x)$ evaluates that solution's worst-case performance with respect to the second population:

$$h(x) = \max\{F(x, s) : s \in P_2\}$$

The algorithm penalizes large $h(x)$ and rewards small $h(x)$, so solutions with better worst-case performance will survive.

Similarly, for a scenario s in P_2, the objective function $g(s)$ evaluates the best solution in the first population:

$$g(s) = \min\{F(x, s) : x \in P_1\}$$

The algorithm penalizes small $g(s)$ and rewards large $g(s)$, so scenarios with worse optimal solutions will survive.

For instance, consider Example 1, which displays hypothetical populations of solutions and scenarios. The entry in each cell of the table is $F(x_i, s_j)$. x_1 is more likely to survive since it has the best worst-case performance ($h(x_1) = 8$), and s_3 is more likely to survive since it has a poor optimal solution ($g(s_3) = 8$).

Example 1

	Scenario			
Solution	s_1	s_2	s_3	$h(x_i)$
x_1	4	7	8	8
x_2	2	10	9	10
x_3	9	6	10	10
$g(s_j)$	2	6	8	

A traditional, simple genetic algorithm has the following steps:

1. Create initial generation $P(0)$. Let $t = 0$.

2. For each individual $i \in P(t)$, evaluate its fitness $f(i)$.

3. Create generation $P(t + 1)$ by reproduction, crossover, and mutation.

4. Let $t = t + 1$. Unless t equals the maximum number of generations, return to Step 2.

The two-space genetic algorithm can be summarized as follows:

1. Create initial generations $P_1(0)$ and $P_2(0)$. Let $t = 0$.

2. For each individual $x \in P_1(t)$, evaluate $h(x) = \max\{F(x, s) : s \in P_2(t)\}$.

3. For each individual $s \in P_2(t)$, evaluate $g(s) = \min\{F(x, s) : x \in P_1(t)\}$.

4. Create generation $P_1(t + 1)$ by reproduction, crossover, and mutation.

5. Create generation $P_2(t + 1)$ by reproduction, crossover, and mutation.

6. Let $t = t + 1$. Unless t equals the maximum number of generations, return to Step 2.

This algorithm is motivated by the need to search two spaces, X and S, when it is not possible to identify, in reasonable time, the worst-case performance of a solution. If the set S is not large, then, while searching the set X, one could evalute $\max_{s \in S} F(x, s)$ for each solution x that the search finds. If the set S is large, however, repeatedly searching S to determine $\max_{s \in S} F(x, s)$ will lead to excessive computational effort.

The two-space genetic algorithm reduces the computational effort needed. It takes a sample population from the set of scenarios and allows this to evolve while the algorithm is searching for solutions. Thus, it searches two spaces simultaneously. Moreover, it is evaluating the solutions in parallel, since it uses the same scenarios for all solutions.

The chosen objective functions encourage the two-space genetic algorithm to converge to a robust solution. Although we do not prove the convergence, the following argument provides the necessary insight. Suppose that there exists a solution $z \in X$ and a scenario $t \in S$ such that

$$F(z, t) = \min_{x \in X} F(x, t) = \max_{s \in S} F(z, s)$$

Consequently,

$$F(z, t) = \min_{x \in X} \max_{s \in S} F(x, s) = \max_{s \in S} \min_{x \in X} F(x, s)$$

If the initial populations are sufficiently large, then for all $x \in P_1$, $h(x)$ is approximately $\max_{s \in S} F(x, s)$. Likewise, for all $s \in P_2$, $g(s)$ is approximately $\min_{x \in X} F(x, s)$. Thus, the populations are likely to converge towards z and t.

Now, consider any generation such that z is in P_1 and t is in P_2. Then, $h(z) = F(z, t)$ and $g(t) = F(z, t)$. For all other $x \in P_1$, $h(x) \geq F(x, t) \geq F(z, t) = h(z)$. Thus, z is more likely to survive. Similarly, for all other $s \in P_2$, $g(s) \leq F(z, s) \leq F(z, t) = g(t)$. Thus, t is more likely to survive.

Consequently, we can see that, in this case, the genetic algorithm will converge to z, the most robust solution, and t, that solution's worst-case scenario.

We used the two-space genetic algorithm to solve a minimax optimization problem. Before discussing the implementation details for the two-space genetic algorithm, we will present the problem under consideration.

4 Experimental Results

To test the two-space genetic algorithm, we randomly created a number of problem instances. A problem instance consists of the following information:

- The number of supply facilities p.
- The number of demand sites m.
- The number of factors that affect demand q.
- The cost r of delivering material that the assigned vehicles cannot.
- The supply S_i available at facility i, for each $i = 1, \ldots, p$.
- The cost c_{ij} of assigning a vehicle to the route from facility i to site j, for all combinations of i and j.
- The nominal demand D_j^0 that will occur at site j, for each $j = 1, \ldots, m$.

Table 1: Problem Sets.

Set	p, m	q	r	$S_{\min}$, $S_{\max}$	$c_{\min}$, $c_{\max}$	$D_{\min}$, $D_{\max}$	$d_{\min}$, $d_{\max}$	$X_{\max}$
A	3, 3	9	20	100, 200	1, 10	100, 200	0, 40	127
B	3, 3	9	20	100, 200	1, 10	100, 200	-20, 40	127

- The change d_{jk} in demand at site j if factor k reaches full strength, for all combinations of j and k.
- The maximum number of vehicles assigned to a route $X_{\max}$.

The following parameters are needed to create a random set of instances.

- The number of supply facilities p.
- The number of demand sites m.
- The number of events that affect demand q.
- The cost r of delivering material that the assigned vehicles cannot.
- The minimum $S_{\min}$ and maximum $S_{\max}$ supply available at a facility.
- The minimum $c_{\min}$ and maximum $c_{\max}$ cost of assigning a vehicle to a route.
- The minimum $D_{\min}$ and maximum $D_{\max}$ nominal demand at a site.
- The minimum $d_{\min}$ and maximum $d_{\max}$ change in demand due to a factor.
- The maximum number of vehicles assigned to a route $X_{\max}$.

Table 1 lists the parameters chosen for two sets. For each set, we generated ten instances. For information like the supply, cost, nominal demand, and demand change, we created values by sampling uniformly distributed random variables that have the specified minimum and maximum values.

To evaluate the two-space genetic algorithm, we tested three procedures: The first procedure (TSGA) was the two-space genetic algorithm. The population of scenarios evolved

as described above. The second procedure (RGA) re-initialized the population of scenarios randomly each generation. Thus, the solutions were evaluated against a random sample of the scenarios. The third procedure (WCGA) did not use a second population. Instead, each solution was evaluated on the known worst-case scenario. That is, the genetic algorithm found solutions to the problem P_t. Under scenario t, the demand at site j is D_j^t:

$$D_j^t = D_j^0 + \sum_{k=1}^{q} \max\{d_{jk}, 0\}$$

(If all $d_{jk} \geq 0$, then t is the true worst-case scenario. Otherwise, it represents an impossible scenario that is the upper bound on demand.)

For each instance, the best solution found in each run of each procedure was evaluated on all scenarios to find its actual worst-case performance. (This was done by completely enumerating all extreme-point scenarios and evaluating $F(x, s)$ for each.) This performance was compared to the optimal solution, and we calculated the relative difference (in percent). We found the optimal solution by creating the problem P_L and using CPLEX to solve it. Each GA was run three times on each instance. The results are for the average relative value over the three runs and for the best found in those three runs. These were, in turn, averaged over the ten instances in the problem set. Table 2 lists the results. Note that all results are listed as percent deviation from the optimal.

More specifically, let z_{cd} be the worst-case performance of the solution that run d of an algorithm found for instance c, and z_c^* be the worst-case performance of the optimal solution for instance c. Then, the relative deviation $r_{cd} = z_{cd}/z_c^* - 1$. We define two measures for the problem set:

$$R_0 = \frac{1}{10} \sum_{c=1}^{10} \min\{r_{c1}, r_{c2}, r_{c3}\}$$

$$R_1 = \frac{1}{10} \sum_{c=1}^{10} \frac{r_{c1} + r_{c2} + r_{c3}}{3}$$

In Problem Set A, each instance has a single worst-case scenario. Note that the results for TSGA and WCGA are very similar. These results show that, in such cases, the two-space genetic algorithm can find the worst-case scenario and locate a robust solution.

Table 2: Experimental Results.

Problem Set	Statistic	TSGA	RGA	WCGA
A	R_0	2.26	2.79	2.38
	R_1	2.76	3.09	2.77
B	R_0	3.87	4.14	6.57
	R_1	6.02	5.06	9.33

In Problem Set B, an instance has no single worst-case scenario. In this case, the WCGA is an inferior approach because it finds good solutions to the wrong problem. The TSGA and RGA have similar results. The second population of the TSGA is converging to a single scenario, and the first population is converging to a solution that, although good for that scenario, is not the most robust. In this problem, the worst-case scenarios are not much worse than other scenarios. Thus, the random sampling in the RGA is able to approximate the worst-case behavior.

5 Summary and Conclusions

This paper discussed a minimax network design problem and presented a two-space genetic algorithm to solve the problem. In this problem, the design must be specified before the actual demands become known. The risk-averse decision-maker wants to avoid worst-case performance.

The particular problem presented can be formulated as a linear program and solved using commercial solvers. This was useful for evaluating the algorithm's performance. However, the two-space genetic algorithm is a more general procedure. For instance, it can search for robust solutions if the cost function has a more general shape

Depending on the nature of the possible scenarios, the results show that a two-space genetic algorithm is a very suitable technique for problems such as these.

This approach can be a general technique for solving minimax optimization problems that arise if robust discrete optimization. It will be particularly useful when determining the worst-case scenario is a difficult problem due to the number of scenarios.

However, it may be possible to improve the algorithm's performance for the more difficult

cases. Maintaining diversity in the population of scenarios may be necessary. If the second population converges to just one scenario, the first population will converge to the solutions that are optimal for that scenario. However, these solutions may have terrible performance on other scenarios. There exist niche mechanisms that can maintain diversity, and these may be suitable. Another approach would let each individual in the second population represent a set of scenarios. Thus, even if the second population converges, the algorithm can approximate a solution's worst-case performance more accurately. This yields a better evaluation of the individuals in the first population and causes that population to converge to truly robust solutions.

6 Appendix

Theorem 1 *For Problem P, there is, for any solution $x \in X$, an extreme point scenario that is a worst-case scenario. An extreme point scenario is a scenario s such that $f_k^s \in \{0, 1\}$ for all $k = 1, \ldots, q$.*

Proof. Suppose $s^* = (f_1^*, \ldots, f_q^*)$ is a worst-case scenario for solution x. If s^* is an extreme point scenario, we are done. Otherwise, let f_v^* be the first element of s^* that is neither zero nor one. Note $0 < f_v^* < 1$. Define $s_0 = (f_1^0, \ldots, f_q^0)$ and $s_1 = (f_1^1, \ldots, f_q^1)$ as follows:

$$f_v^0 = 0$$

$$f_v^1 = 1$$

$$f_k^0 = f_k^1 = f_k^*, \forall k \neq v$$

For example, if $s^* = (1, 0, 0.6, 0.2, 0)$, then $s_0 = (1, 0, 0, 0.2, 0)$ and $s_1 = (1, 0, 1, 0.2, 0)$.

Then $s^* = f_v^* s_1 + (1 - f_v^*) s_0$. By the following lemma, $F(x, s^*) \leq \max\{F(x, s_0), F(x, s_1)\}$, so either s_0 or s_1 is also a worst-case scenario for solution x.

If this point is not an extreme point scenario, we can repeat the above construction (starting with either s_0 or s_1) to find another point that is also a worst-case scenario. Since

there are a finite number of elements, we will eventually find a worst-case scenario that is an extreme point scenario. QED.

Lemma 1 *Consider any $x \in X$, $s_0 \in S$, and $s_1 \in S$. For all h such that $0 \leq h \leq 1$, if $s_h = hs_1 + (1-h)s_0$, then $F(x, s_h) \leq \max\{F(x, s_0), F(x, s_1)\}$.*

Proof. $s_0 = (f_1^0, \ldots, f_q^0)$. $s_1 = (f_1^1, \ldots, f_q^1)$. $s_h = (f_1^h, \ldots, f_q^h)$. By the definition of s_h, $f_k^h = hf_k^1 + (1-h)f_k^0$ for all $k = 1, \ldots, q$.

$D_j^{s_h} = D_j^0 + \sum_{k=1}^q d_{jk} f_k^h$, so $D_j^{s_h} = hD_j^{s_1} + (1-h)D_j^{s_0}$, for $j = 1, \ldots, m$.

Recall the definition of $F(x, s)$:

$$F(x,s) = \min_{y^s, z^s} \sum_{i=1}^{p} \sum_{j=1}^{m} c_{ij} x_{ij} + r \sum_{j=1}^{m} z_j^s$$

Let (y^0, z^0) and (y^1, z^1) be the optimal solutions for scenarios s_0 and s_1. Then, for $a = 0, 1$ the following statements are true:

$$F(x, s_a) = \sum_{i=1}^{p} \sum_{j=1}^{m} c_{ij} x_{ij} + r \sum_{j=1}^{m} z_j^a$$

$$\sum_{i=1}^{p} y_{ij}^a + z_j^a = D_j^{s_a}, \forall j = 1, \ldots, m$$

$$y_{ij}^a \leq x_{ij}, \forall i = 1, \ldots, p, j = 1, \ldots, m$$

Now, form a solution (y^h, z^h) to the problem in scenario s_h as follows:

$$y_{ij}^h = hy_{ij}^1 + (1-h)y_{ij}^0, \forall i, j$$

$$z_j^h = hz_j^1 + (1-h)z_j^0, \forall j$$

This solution is feasible:

$$\sum_{i=1}^{p} y_{ij}^h + z_j^h = hD_j^{s_1} + (1-h)D_j^{s_0} = D_j^{s_h}, \forall j = 1, \ldots, m$$

$$y_{ij}^h \leq hx_{ij} + (1-h)x_{ij} = x_{ij}, \forall i = 1, \ldots, p, j = 1, \ldots, m$$

By the definition of $F(x, s)$, $F(x, s_h) \leq \sum_{i=1}^{p} \sum_{j=1}^{m} c_{ij} x_{ij} + r \sum_{j=1}^{m} z_j^h = hF(x, s_1) + (1-h)F(x, s_0)$. Thus, $F(x, s_h) \leq \max\{F(x, s_0), F(x, s_1)\}$. QED.

References

[1] Bazaraa, Mokhtar S., John J. Jarvis, Hanif D. Sherali, *Linear Programming and Network Flows*, 2nd edition, John Wiley and Sons, New York, 1990.

[2] Birge, John R., and Francois Louveaux, *Introduction to Stochastic Programming*, Springer-Verlag, New York, 1997.

[3] Chang, C.S., F. Wang, and K.S. Lock, "Harmonic worst-case identification and filter optimal design of MRT systems using genetic algorithms," *IEE Proceedings Electric Power Applications*, Volume 144, Number 5, pages 372-380, 1997.

[4] Dahl, Henrik, Alexander Meeraus, and Stavros A. Zenios, "Some financial optimization models: I Risk management," in *Financial Optimization*, S.A. Zenios, editor, Cambridge University Press, 1993.

[5] Daniels, R.L., and P. Kouvelis, "Robust scheduling to hedge against processing time uncertainty in single-stage production," *Management Science*, Volume 41, Number 2, pages 363-376, 1995.

[6] Davis, L., ed., *Handbook of Genetic Algorithms*, Van Nostrand Rheinhold, New York, 1991.

[7] Du, Ding-Zhu, and Panos M. Pardalos, *Minimax and Applications*, Kluwer Academic Publishers, Dordrecht, 1995.

[8] Eppen, Gary D., R. Kipp Martin, and Linus Schrage, "A scenario approach to capacity planning," *Operations Research*, Volume 37, Number 4, pages 517-527, 1989.

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

[10] Grefenstette, J.J., L. Davis, and D. Cerys, "GENESIS and OOGA: Two GA Systems," TSP Publications, Melrose, Massachusetts, 1991.

[11] Hillis, W. Daniel, "Co-evolving parasites improve simulated evolution as an optimization procedure," in *Artificial Life II*, SFI Studies in the Sciences of Complexity, Volume X, C.G. Langton, *et al.*, eds., Addison-Wesley, Redwood City, California, 1991.

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

[13] Kouvelis, Panos, and Gang Yu, *Robust Discrete Optimization and Its Applications*, Kluwer Academic Publishers, Norwell, MA, 1997.

[14] Loraschi, Andrea, and Andrea G.B. Tettamanzi, "An evolutionary algorithm for portfolio selection within a downside risk framework," in *Forecasting Financial Markets*, Christian Dunis, editor, John Wiley and Sons, New York, 1996.

[15] Markowitz, H.M., *Portfolio Selection*, John Wiley, New York, 1959.

[16] Markowitz, Harry M., and Andre F. Perold, "Portfolio analysis with factors and scenarios," *The Journal of Finance*, Volume 36, Number 14, pages 871-877, 1981.

[17] Marrison, Christopher, and Robert F. Stengel, "Robust control system design using random search and genetic algorithms," *IEEE Transactions on Automatic Control*, Volume 42, Number 6, pages 835-839, 1997.

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

[19] Mulvey, John M., Robert J. Vanderbei, and Stavros A. Zenios, "Robust optimization of large-scale systems," *Operations Research*, Volume 43, Number 2, pages 264-281, 1995.

[20] Paredis, Jan, "Coevolutionary lifetime learning," in *Parallel Problem Solving from Nature-PPSN IV*, H.-M. Voight, *et al.*, eds., pages 72-80, Springer, 1996.

[21] Rosenblatt, Meir J., and Hau L. Lee, "A robustness approach to facilities design," *International Journal of Production Research*, Volume 25, Number 4, pages 479-486, 1987.

[22] Sebald, Anthony V., and Jennifer Schlenzig, "Minimax design of neural net controllers for highly uncertain plants," *IEEE Transactions on Neural Networks*, Volume 5, Number 1, pages 73-82, 1994.

[23] Tang, K.S., K.F. Man, and D.-W. Gu, "Structured genetic algorithm for robust H-infinity control systems design," *IEEE Transactions on Industrial Electronics*, Volume 43, Number 5, pages 575-582, 1996.

[24] Taranto, G.N., and D.M. Falcao, "Robust decentralised control design using genetic algorithms in power system damping control," *IEE Proceedings. Generation, Transmissions, and Distribution*, Volume 145, Number 1, pages 1-6, 1998.

[25] Tsutsui, Shigeyoshi, and Ashish Ghosh, "Genetic algorithms with a robust solution searching scheme," *IEEE Transactions on Evolutionary Computation*, Volume 1, Number 3, pages 201-208, 1997.

[26] Yu, Gang, "On the max-min 0-1 knapsack problem with robust optimization applications," *Operations Research*, Volume 44, Number 2, pages 407-415, 1996.