A Genetic Algorithm for Minimax Optimization Problems

Jeffrey W. Herrmann
Department of Mechanical Engineering
and Institute for Systems Research
University of Maryland
College Park, Maryland 20742
jwh2@eng.umd.edu

Abstract- Robust discrete optimization is a technique for structuring uncertainty in the decision-making process. The 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 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 the individuals in the other population. The populations evolve simultaneously, and they 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 its potential, we use the two-space genetic algorithm to solve a parallel machine scheduling problem with uncertain processing times. Experimental results show that the two-space genetic algorithm can find robust solutions.

1 Introduction

Many decisions involve uncertainty. Recently, researchers have begun to study such problems and to develop approaches for finding robust solutions that have the best worst-case performance over a set of possible scenarios. However, this is a difficult optimization problem, and no general techniques exist. Kouvelis and Yu [9] discuss approaches for handling uncertainty, and they review robust discrete optimization problems, complexity results, and solution procedures.

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 the individuals in the other population. This causes the algorithm to converge to the robust solution and its worst-case sceanrio. 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 parallel machine scheduling problem with uncertain processing times. The objective is to find a schedule that minimizes the worst-case makespan. Experimental results show the two-space genetic algorithm can find robust solutions.

The remainder of this paper is structured as follows: Section 2 describes robust discrete optimization problems. Section 3 presents the two-space genetic algorithm. Section 4 describes the parallel machine scheduling problem. Section 5 presents the experimental results. Section 6 concludes the paper by identifying topics for future research that could extend the algorithm's utility.

2 Robust Discrete Optimization Problems

Making decisions under uncertainty is a difficult problem. However, accepting and structuring uncertainty can lead to effective decision-making. Stochastic optimization recognizes uncertainty but asks the decision-maker to assign probability distributions to possible outcomes. Then, after solving the associated optimization problem, one can select the decision that has the best average performance over time or one can select the decision that has the best performance in the expected outcome.

Robust discrete optimization, on the other hand, seeks to identify decisions that will perform well under any circumstances. Although many criteria are available, one reasonable choice is the minimax criterion, which allows one to identify a robust decision (or solution) as one that has the best worst-case performance.

Robust discrete optimization (as described by Kouvelis and Yu [9]) uses scenarios 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 future. That is, the scenario occurs with some positive but unknown probability.

In general, a robust discrete optimization problem can be formulated as follows. Let X be the set of all solutions. Let S be the set of all possible scenarios. The performance of a solution $x \in X$ in scenario $s \in S$ is $F(x, s)$. The problem is to find the solution that has the best worst-case performance, which is the same as minimizing (over all solutions) the maximum (over all scenarios) performance:

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

In general, robust discrete optimization problems are more difficult to solve than the deterministic versions that have no uncertainty. The robust discrete optimization versions of many polynomially solvable optimization problems are NP-hard, although some easily solvable cases do exist. (For results in the more general area of minimax theory, see Du and

Pardalos [3].)

Kouvelis and Yu [9] describe a branch-and-bound algorithm that uses surrogate relaxation to generate bounds, and they use this procedure to solve four robust optimization problems. However, there are no general techniques for solving robust discrete optimization problems.

Some authors have proposed genetic algorithms for problems with uncertainty. (For more information on genetic algorithms, see, for example, [2, 4, 7, 12].) Tsutsui and Ghosh [16] present a genetic algorithm that applies noise to the decoding procedure. This allows the algorithm to find solutions whose performance is insensitive to small changes in the solution values. They present results for some functions on one- and two-dimensional search spaces. However, they do not explicitly consider the scenarios present in robust discrete optimization problems.

The design of robust control systems is an important related area, and a few authors have proposed genetic algorithms for this problem. Taranto and Falcao [15] 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 [10] 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. This is a stochastic optimization approach, though their genetic algorithm does handle uncertainty in estimating the cost function, which cannot be evaluated directly. Tang, Man, and Gu [14] use a genetic algorithm to search for distillation column controllers that perform well for all possible plant characteristics. For a problem with four plants, 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. This is the problem that the two-space genetic algorithm addresses.

3 Two-space Genetic Algorithm

We propose a two-space genetic algorithm for finding good solutions to minimax 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 evaluate $\max_{s \in S} F(x,s)$ 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.

To test its potential, we used the two-space genetic algorithm to solve a specific robust discrete optimization problem. Before discussing the implementation details for the two-space genetic algorithm, we will present the scheduling problem under consideration.

4 Parallel Machine Scheduling

There are a wide variety of parallel machine scheduling problems. Pinedo [13] provides a review of the most important research results. Most previous work has considered deterministic scheduling, where all information is known ahead of time, and stochastic scheduling, where some information has a known probability distribution. We will consider a problem with more general uncertainty. Previously, Daniels and Kouvelis [1] and Kouvelis *et al.* [8] have studied scheduling problems with uncertain processing times and developed methods to optimize the worst-case performance. In addition, researchers have developed genetic algorithms to find good solutions to deterministic scheduling and sequencing problems (see, for example, [5, 6, 11]).

We will consider the following problem. There is a finite set of jobs and a set of parallel, identical machines. All jobs are available at the current time. Each job requires processing on any one of the machines. Each job's processing time is uncertain. The only available data are a lower bound and an upper bound on the required time. The problem is to create a schedule that minimizes the total time needed to complete all jobs. No preemption is allowed. No rescheduling can occur once processing begins.

Let $\{J_1, \ldots, J_n\}$ be the set of jobs. Each job J_j has a minimum processing time p_j and a maximum processing time q_j, where $0 < p_j < q_j$. There are m machines M_1, ..., M_m. The decision variables are the nm binary variables x_{jk}, where $x_{jk} = 1$ if and only if J_j is assigned to machine M_k. An assignment x is a feasible assignment (or solution) if, for each job J_j, $\sum_{k=1}^{m} x_{jk} = 1$. Let X be the set of all solutions.

A scenario s is a combination of realized processing times. Thus, $s = (p_1^s, \ldots, p_n^s)$. For each job J_j, $p_j \leq p_j^s \leq q_j$. S is the set of all possible scenarios.

$F(x,s)$ is the makespan of a solution x in scenario s. The makespan is the maximum total processing time on any machine:

$$F(x,s) = \max_{1 \leq k \leq m} \{\sum_{j=1}^{n} x_{jk} p_j^s\}$$

The problem is to minimize (over all schedules) the maximum possible makespan:

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

We will use the two-space genetic algorithm to find robust solutions.

For this simple problem it is easy to find the worst-case scenario, and thus we can judge the two-space genetic algorithm's performance. Note that, for any solution, one of its worst-case scenarios is the one when all jobs require their maximum processing time. Let t represent this scenario: $t = (q_1, \ldots, q_n)$. Then, $F(x,t) = \max_{s \in S} F(x,s)$. Thus, $\min_{x \in X} F(x,t)$ is the minimum worst-case performance. A lower bound on this worst-case performance is $L_a = \frac{1}{m} \sum_{j=1}^{n} q_j$. In the experiments described later, we will consider instances with four machines and nine jobs. Thus, at least one machine will have at least three jobs. A second lower bound is the sum of the three smallest maximum processing times: $L_b = q_1 + q_2 + q_3$, if we number the jobs so that $q_1 \leq q_2 \leq q_3 \leq \ldots \leq q_n$. $L = \max\{L_a, L_b\}$ combines these lower bounds for $\min_{x \in X} F(x,t)$.

5 Experimental Results

To implement the two-space genetic algorithm, we used GENESIS, a genetic algorithm software developed by John J. Grefenstette. We modified the code so that the algorithm creates and maintains two populations. Each individual in each population is evaluated with respect to the individuals in the other population. Each run of the algorithm created 100 generations of each population. Each population had 50 individuals. The algorithm used an elitist strategy that copied the best individual in one generation to the next generation.

Recall that a solution assigns every job to a machine, and a scenario is a combination of processing times. The objective is to minimize the worst-case schedule makespan.

5.1 Problem Generation

We tested the two-space genetic algorithm on randomly generated problem instances. All instances had $m = 4$ identical parallel machines. We created various problem sets, each with ten instances. Various parameters governed the instance generation. The problem size was $n = 9$ jobs. The parameters α_1 and α_2 governed the processing times. Each job's minimum processing time p_j was selected from a uniform distribution with the range $[10, 50\alpha_1]$. α_1 equals 0.2, 0.6, or 1.0. Each job's maximum delay d_j was selected from a uniform distribution with the range $[0, \alpha_2 p_j]$. α_2 equals 0.2, 0.6, or 1.0. The maximum processing time $q_j = p_j + d_j$.

Each set of ten instances can be denoted by the combination of the parameter values. We used all nine combinations of α_1 and α_2. Thus, there were 90 instances altogether.

To solve these problems, the genetic algorithm uses 27-bit strings. Each individual in both populations has nine genes, one for each job. Each gene contains three bits. Individuals in the first population represent assignments. After transforming the gene's value to an integer in the set $1, \ldots, m$, gene j identifies a machine k, and the job is assigned to that machine. That is, $x_{jk} = 1$ and $x_{jp} = 0$ if $p \neq k$. Each individual in the second population represents a scenario s. After transforming the gene's value to a real number in the interval $[p_j, q_j]$, gene j describes the job's realized processing time p_j^s.

5.2 Results

To evaluate the algorithm, we compare the solutions to the lower bound L, as described in Section 4. In addition, we used a standard version of GENESIS to find good solutions to the problem $\min_{x \in X} F(x, t)$. Recall that scenario t is the worst-case scenario, and the optimal makespan of this problem is the optimal makespan to the robust scheduling problem.

All results are relative to the lower bounds and averaged over the ten instances in the problem set. If, for each instance $i = 1, \ldots, 10$, an algorithm finds a solution x_i^* whose worst-case performance is $h(x_i^*)$, and the lower bound for that instance is L_i, then, the algorithm's average relative performance is H:

$$H = \frac{1}{10} \sum_{i=1}^{10} \frac{h(x_i^*)}{L_i}.$$

Table 1 lists the results. The deviation from the lower bound is the same for both algorithms. This shows that the two-space genetic algorithm is able to find robust solutions.

Problem Set			Two-space	Worst-case
n	α_1	α_2	Genetic Algorithm	Optimization
9	0.2	0.2	1.000	1.000
9	0.2	0.6	1.002	1.008
9	0.2	1.0	1.028	1.037
9	0.6	0.2	1.044	1.041
9	0.6	0.6	1.058	1.047
9	0.6	1.0	1.053	1.053
9	1.0	0.2	1.056	1.064
9	1.0	0.6	1.068	1.062
9	1.0	1.0	1.062	1.062

Table 1: Results for the Parallel Machine Scheduling Problem

6 Summary and Conclusions

This paper presented a two-space genetic algorithm and suggested that it can be a general technique for solving minimax and robust discrete optimization problems. The two-space genetic algorithm should be useful for solving minimax optimization problems in a wide variety of domains. It will be particularly useful when determining the worst-case scenario is a difficult problem due to the number of scenarios.

To illustrate its potential, we used the algorithm to solve a parallel machine scheduling problem with uncertain processing times. For this particular problem, we can find good lower bounds and thus evaluate the algorithm's performance. The results show that a two-space genetic algorithm is a very suitable technique for robust discrete optimization problems such as these. For the specific problem considered here, one can verify that there exists a solution $z \in X$ and a scenario $t \in S$ such that

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

However, the algorithm may require adjustment to solve problems when this condition does not hold. Preliminary results suggest that 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. Evaluating how often a scenario is a worst-case scenario may be an appropriate objective function, and sharing mechanisms can maintain a variety of worst-case scenarios in the second population. This yields a more accurate evaluation of the individuals in the first population and causes it to converge to truly robust solutions.

Another possibility is to give each individual a memory so that, if an individual survives intact from one generation to the next, it can compare its performance (with respect to the current population of scenarios) to its worst-case performance in the past. This also yields more accurate evaluations.

Bibliography

[1] 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.

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

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

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

[5] Herrmann, Jeffrey W., and Chung-Yee Lee, “Solving a class scheduling problem with a genetic algorithm,” *ORSA Journal of Computing*, Vol. 7, No. 4, 1995.

[6] Herrmann, Jeffrey W., Chung-Yee Lee, and Jim Hinchman, “Global job shop scheduling with a genetic algorithm,” *Production and Operations Management*, Volume 4, Number 1, pp. 30-45, 1995.

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

[8] Kouvelis, P., and R.L. Daniels, and G. Vairaktarakis, “Robust scheduling of a two-machine flow shop with uncertain processing times,” working paper, Fuqua School of Business, Duke University, 1996.

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

[10] 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.

[11] Mattfeld, Dirk C., *Evolutionary search and the job shop : investigations on genetic algorithms for production scheduling*, Heidelberg : Physica-Verlag, 1996.

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

[13] Pinedo, Michael, *Scheduling : theory, algorithms, and systems*, Englewood Cliffs, N.J.: Prentice Hall, 1995.

[14] 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.

[15] 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.

[16] 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.