# Granular Optimization: An Approach to Function Optimization†

**Yu-Chi Ho & Jonathan T. Lee‡**
Division of Engineering and Applied Sciences
Harvard University, Cambridge, MA 02138 USA
Email: ho@hrl.harvard.edu or jtlee@hrl.harvard.edu

**Abstract**

Finding a function that minimizes a functional is a common problem, e.g., determining the feedback control law for a system. However, it remains to be a challenge due to the large and structureless search space. In this paper, we present a search algorithm, *granular optimization*, to deal with this type of problems under some mild constraints. The algorithm is tested on two different problems. One of them is the well-known Witsenhausen counterexample. On the counterexample, the result from our automated algorithm comes close to the currently known best solution, which involves much human intervention. This shows the potential usefulness of the algorithm in more general problems.

## 1 Introduction

In many real life circumstances, we encounter problems where the directive is to identify the optimal strategy, a mapping from the information space to the decision space.[1] A good example would be the determination of a feedback control law for some systems. In this paper, we establish a new search technique, *granular optimization* (GO), which is designed to search for the optimal function in the space of functions. Our goal is to understand the underline dynamics of this search technique and demonstrate its usefulness on two test problems, e.g., the function approximation problem and the Witsenhausen counterexample [10].

First, we state the problem in more precise terms. In Section 2, we explain the intuitions behind the algorithm and define the algorithm for GO. Afterward, we state the test problems, namely the Witsenhausen counterexample and the function approximation problem. In Section 3.2, we present the empirical results of GO on the two test problems. Following that, we compare our method to the traditional descent method on the Witsenhausen counterexample. Then, analyze of the effect of the various parameters in the algorithm on the search result is presented in Section 6. We conclude in Section 7. The definitions of all the symbols used in this paper are summarized in Appendix 1.

### 1.1 Problem Statement

First, we formally state the problem of interest. Without loss of generality, we focus on finding the global minimum.[2] The problem can be states as follows:

$$\min_{\theta \in \Theta} J(\theta)$$

where θ is a function, Θ is the space of all feasible functions and $J(\bullet)$ is the performance criteria functional. In particular, the input and output space of all feasible functions are bounded. Furthermore, they are discretized, thus, represented by piece-wise constant functions.[3] [4] In trying to solve this type of problems, we are often faced with three difficulties:

1. Large search space;
2. Noise in performance evaluation; and
3. Structureless search space.

In this paper we will be dealing primarily with difficulties induced by 1 and 3 and only indirectly with 2. First we make the following assumption.

† The work in this paper is partially supported by Army contracts DAAL03-92-G-0115 and DAAH04-0148, Air Force Grant F49620-98-1-0387 and Office of Naval Research Contract N00014-98-10720 and Electric Power Research Institute Contract WO8333-03.

‡ The corresponding author.

[1] In this paper, we use the following terms interchangeably: function, design, mapping and solution.

[2] For finding the global maximum, one could turn such a problem into finding global minimum by negating the performance criteria $J(\bullet)$, i.e., $-J(\bullet)$.

[3] In this paper, we refer to the piece-wise constant functions as step function.

[4] There is no loss of generality to represent function as step function since any finitely supported and bounded function with a finite number of discontinuities can be approximated to any degree of accuracy by a step function, given there are enough number of steps.

**Assumption**: We restrict consideration to problems where $J(\theta)$ satisfies the Lipschitz condition but otherwise arbitrary.[5] [6]

Given the assumption, there is no loss of generality to represent the feasible function as step functions. Of course the computational burden goes up as the size of the input and output set increases. In fact if the cardinality of the input space and of the output space are $|I|$ and $|O|$ respectively, then the total size of the function space, $\Theta$, is $|O|^{|I|}$, a humongous number. Since we assume little further structural information on $J$, we have a difficult search problem which cannot be attacked efficiently by blind search. A more intelligent learning approach must be adopted if we are going to have some success. This is the main thrust of this paper.

If in addition noise complicated the evaluation of the functional $J(\theta)$ and statistical experiment must be used to help the calculation, then the computational problem further increases. Techniques such as ordinal optimization ([4], [6]) can be useful. We shall touch upon these later.

## 2 Granular Optimization with Learning Automata

In this section, we present the intuition behind the algorithm. We also provide a brief introduction to learning automata. In the end, we define the algorithm itself.

### 2.1 The Intuition

The main idea here is that blind search in a huge space is rather inefficient. Using assumption 2 we clump a large space into *granules*, i.e., similar functions would result in similar performance values because of assumption 2. In other words, a set of similar designs can be granulated into one representative granule. Thus, if we sample and evaluate a design from a particular granulate, we would know, approximately, the performance values of all the designs in the same granule [12].[7] Using granulation, we can first search in a coarse space where elements represent granules from finer spaces. More precisely, we start our search in the 1-step or 2-step space. The advantage is that the search space is very small. Once we located good designs in the coarse space, we de-granulate these designs into a finer (say, 3-step) space. Now in this finer space we bias our search around the neighborhood of these granules which have good performance values in the coarse space. This way, we learn to bias our search in the finer space to the more promising subsets. We can gradually and iteratively move from 1-step to the maximum discretization $N$-step space. A good analog to the ideas of GO is visual identification. One can think of granulation as resolution in the context of vision. The goal is to identify an object correctly. When a picture appears on the computer screen, often, we first see it in a very coarse resolution. Gradually, the resolution becomes finer and finer. Thus, in the beginning, we could only identify a human like figure in the foreground with a mountain in the background. As more details are being filled in, we could see that it is a man in the foreground. When the picture is in its highest resolution, we could see that he is wearing a watch.

In addition, granular optimization takes advantage of the representation of the feasible solutions. In this paper, we choose to represent the possible solutions using step functions. The approximation property of such a scheme is that the more steps allot, the better the approximation becomes. On the other hand, the more parameters are needed to define the step functions. With fewer steps allot, the search space is much smaller due to a smaller number of parameters. On the other hand, the quality of the approximation is poorer. In contrast, with more steps allot, the quality of the approximation becomes better. Unfortunately, the search space grows exponentially — the *curse of dimensionality*. Our algorithm, GO, balances these two conflicting trends to search efficiently via learning. Here is how. Since all $i$-step functions can be thought of as $(i+1)$-step function with two of the adjacent steps having the same output values, $i$-step functions are in the sub-set of the $(i+1)$-step space. Thus, all of the $(i+1)$-step functions are granulated into some $i$-step function. Given the Lipschitz condition, all the $(i+1)$-step designs are in the same granule and would have similar performance value as the $i$-step design. In other words, all the designs in $(i+1)$-step space that are similar to the same design in $i$-step space would have similar performance values. We use the information that we obtained from $i$-step space to locate the neighborhood of good designs in the $(i+1)$-step space. We pay more attention to these neighborhoods (subsets) in our search.

### *2.2* Learning Automata

Learning automata, as the name suggests, is a learning system.[8] Its goal is to find the optimal solution in the solution space while operating in an unpredictable environment. A learning automaton outputs a design, $\theta_i$. The design is then evaluated by the random environment. A feedback, $\beta_i$, is returned to the automaton where a learning algorithm resides. Based on the feedback, the learning algorithm updates the probability of sampling a particular design. Then, another design would be sampled based on the new probability distribution and evaluated by the random environment. The cycle goes on, as illustrated in Figure 1.

---

[5] Since the search (solution) space is finite, we do not have continuity of $J(\theta)$ in $\theta$.

[6] This implies similar designs will have similar performance values.

[7] Our idea of granule is similar to Zadeh's idea in spirit but applied in different context. In fact, this work is partially inspired by Zadeh's use of granule in [12].

[8] For a more through treatment on this subject, please refer to [7] and [9].

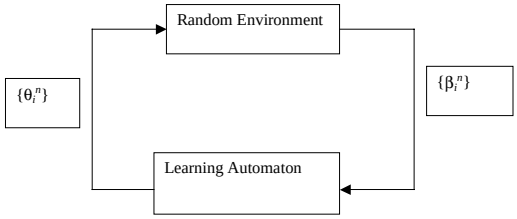


**Figure 1. Automaton Interacting with the Random Environment**

In our case, the automaton outputs a set of designs from their space of *n*-step, $\{\theta_i^n\}$. The random environment, then, evaluates each design and orders them according the performance values. The feedback form the random environment is the order of the designs, $\{\beta_i^n\}$. Based on the order, the automaton would update the probability distribution of where to sample for (*n*+1)-step space. In our study, the specific learning algorithm used in the learning automaton is known as the "reward and inaction" ($L_{R-I}$ automaton). If a design is among the good enough designs in *n*-step space, e.g., top-50 designs, the probability of sampling from its neighborhood is boosted by the amount of $\Delta P$.[9] The probability of sampling from the entire space is *CProb*. Then, the total probability is normalized. (Since *CProb* is not reduced by the total probability boosted for the top designs, the total probably is greater than 1. The overall probably would be renormalized to 1.)

We utilize the learning automata, based on the result from *n*-step space, to gain knowledge of the possible location of good designs in the (*n*+1)-step space. Let us elaborate. Due to Lipschitz assumption, designs in (*n*+1)-step space would granulate into a design in *n*-step space with similar performance value. Thus, better designs in (*n*+1)-step would granulate into better designs in *n*-step. After observing the good enough designs in *n*-step space, we boost the probability of sampling from designs in (*n*+1)-step that are similar to them. Therefore, we have enhanced the probability of sampling of good designs from (*n*+1)-step space.

### 2.3 Algorithm

Figure 2 is a flow chart where the main components of the algorithm are represented. Below is a brief description of those components:

0. Initialization: enter the value for $\delta$ (the amount of deviation allowed when we search in the neighborhood of a good enough design), $\Delta P$ (the amount probability boosted), *Top* (the number of top designs that are designated as good enough) and *Sam* (the number of samples taken at each iteration).
   Set *CProb* (the compliment probability: the probably of sampling the entire space) = 1, set *n* (the iteration as well as the resolution of the search space, e.g., *n*-step space)= 1.
1. Coin flip with (*CProb*) probability to go to 2A and (1 – *CProb*) to go to 2B.
2. A: Uniformly sample from the space $\Theta_n$.
   B: Uniformly sample from the neighborhood of the good enough designs.[10]
3. Evaluate the sampled design.
   Repeat 1 through 3 for *Sam* number of samples.
4. Order the designs by their performance values.
5. Boost the probability of sampling each of the good enough designs by $\Delta P$ then renormalize the total probability if the best design in this iteration has a better performance value than the best design from the previous iteration.
6. Take the top-*Top* designs and set them as the good enough designs.
   Repeat 1 through 6 till *N*-step space (the finest resolution under investigation).

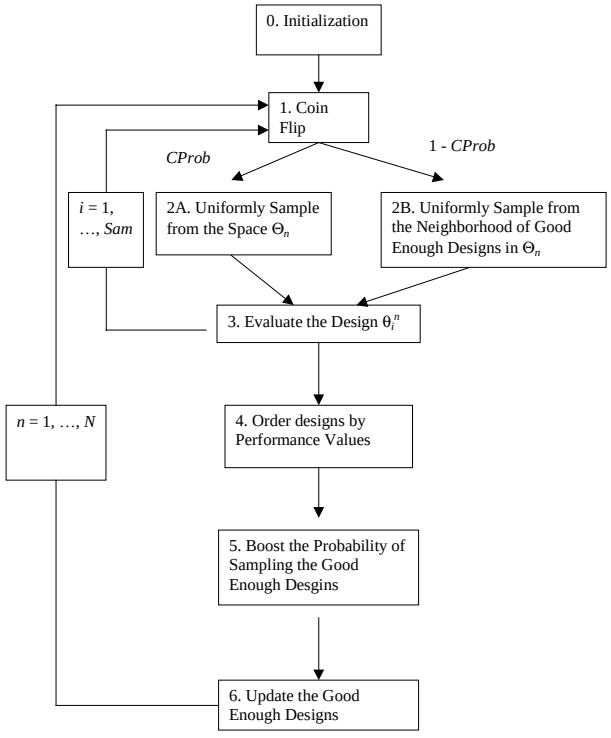


**Figure 2. Granular Optimization Algorithm Flow Chart**

[9] The neighborhood is defined as the functions in (*n*+1)-step that the total deviation over all the steps is at most $\delta$ from the design in *n*-step space.

[10] This is also referred to as *scatter search* since the resulting samples are scattered in the neighborhood of some design.

For the problems that we investigate, both the domain and range are set to be the closed intervals between 0 and 25. The independent variable is discretized into segments of length 1. The dependent variable is discretized into values of 0.1 apart. Thus, there are 25 segments in the input variable and 251 possible values in the output variable. Due to the constraint in monotonicity in our test problems, the search space is about $2 \times 10^{35}$, which is still a rather large search space.

### 2.3.1 Descend-Scatter Subroutine

Instead of uniformly sampling in the neighborhood of the good enough design, we have incorporated a descend technique into it, component 2B above.[11] In place of scatter search in the neighborhood of the good enough designs along, we scatter search until we find a design with better performance. Once a better design is found, gradient is calculated and a design along the gradient is sampled. The algorithm continues to sample along this direction until no further improvement can be made. Then, the procedure calls the program to do scatter search, from the best design so far in this search. To distinguish the descend-scatter subroutine from just scatter routine, we will refer to them as GO with descend-scatter and GO with scatter, respectively. Figure 3 is a flow chart where the main components of the subroutine are represented. Below is a brief description of those components:

0. Initialize: input $\delta$ (the amount of maximum deviation between the sampled design and the good enough design), $\theta_t^{n-1}$ (the $t^{th}$ good enough design from the previous iteration) and $s_t$ (the amount of computational budget allocate to the descent-scatter search in the neighborhood of the $t^{th}$ top design).
1. Randomly add a new breakpoint to the current set of breakpoints in the good enough design.
2. Scatter search around the good enough design in the space of functional values.
3. Evaluate the new design.
   If the new design is worse than the good enough, go to step 4A.
   If the new design is better than the good enough, go to step 4B.
4. A. If exceeds computational budget, exit subroutine. Otherwise, go to step 1.
   B. If exceeds computational budget, exit subroutine. Otherwise, go to step 5.
5. Calculate the gradient between the current design and the previous design. If a new design would result out of the search space, go to 1. Otherwise, proceed to step 6.
6. Create a new design by descending along the gradient direction.
7. Evaluate the new design. If the new design is better than the previous design, go to step 4B. Otherwise, proceed to step 8.
8. Do scatter search in the function value space around the best design so far.

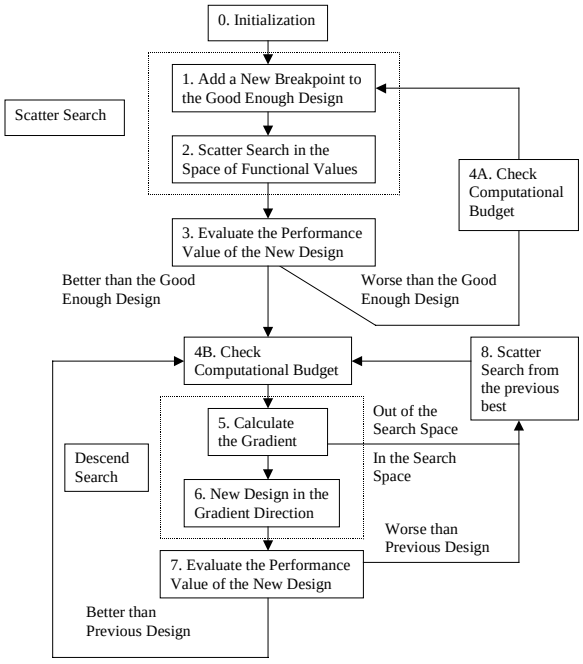


**Figure 3. Flow Chart for the Descend-Scatter Subroutine**

## 3 Test Problems

In this paper, we test our algorithm on two problems. One is the function approximation problem. The other is the well-known Witsenhausen counterexample [10] which has been outstanding since 1968.

### 3.1 Function Approximation

With the function approximation problem, the goal is to find the function among all the feasible ones that is closest to the *target* function, denote as $f^T(x)$. The performance criteria of function $f(x)$, $J(f)$, is as follows:

[11] Modification is made to the algorithm, e.g., Figure 2. Instead of coin flip to decided whether the next design is sample from the entire space or from the good enough neighborhood, the computational budget is allocated to each component, deterministically, as Ceil(*CProb*) * Sam and Floor(1 – *CProb*) * Sam, respectively. Then, the computational budget allocated to the good enough designs is further assigned to each of the good enough designs. Ceil would round to the nearest integer toward positive infinity and floor would round to the nearest integer toward negative infinity. Those values are approximately the expected number of samples from each component.

$$(1) \qquad J(f) = \int_0^{25} \left( f(x) - f^T(x) \right)^2 dx .$$

There are three test functions. The first two of which obey the monotonic property.[12] The third does not. All of them lie in the discretized search space we specified.[13] Thus, the optimal solution is would have a performance value of 0. Below are the plots of the target functions: M25 (Figure 4), MH25 (Figure 5) and R25 (Figure 6). The target function MH25 could be represented well with far fewer than 25 steps since the rightmost 12 steps have very similar values. Thus, it could be relatively easy to locate a solution which well approximates it with as few as 13 steps. On the other hand, the target function R25 is very jagged and would be very difficult to approximate well using a small number of steps. The reason for the three different target functions is to see the effect of the difficulty of the search problem on the search result given our algorithm. We will comment on this in later.

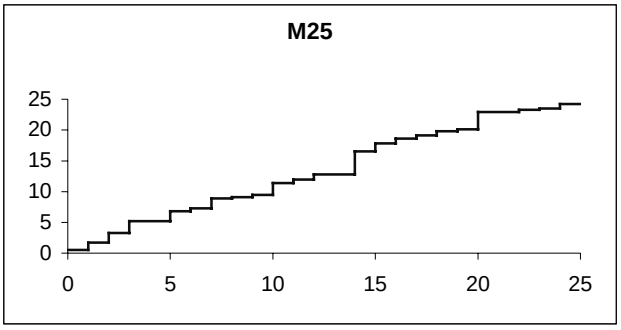


**Figure 4. Target Function M25**

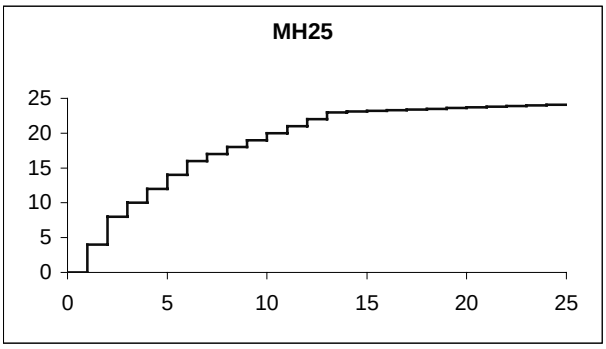


**Figure 5. Target Function MH25**

[12] This is to compare the results when GO is applied to the Witsenhausen problem where the optimal solution has been shown to have monotonic property. The only difference is that one is a convex problem vs. a nonconvex problem.

[13] The search space is the same as the one specified in the Witsenhausen counterexample.

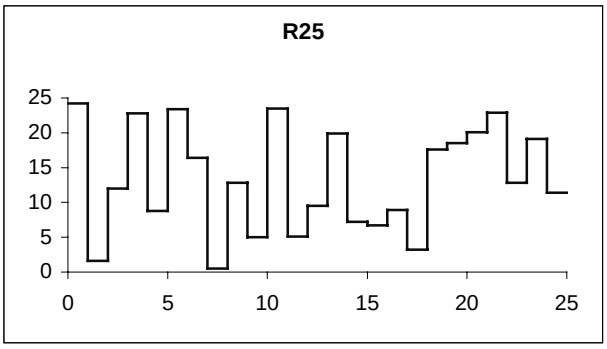


**Figure 6. Target Function R25**

### 3.2 The Witsenhausen Counterexample

The Witsenhausen counterexample [10] is the first example which shows that a LQG problem does not necessary have a linear solution as the global optimum. The form of the optimum is thought to be unknown till recently, see [5]. Essentially, the counterexample is a two-stage team decision problem where the second player does not have the information that is available to the first player, i.e., *non-classical information structure*. The cost term is as follows:

$$\min_{f \in \Theta} J(f) = E[k^2 (x - f(x))^2 + (f(x) - g^*(f(x) + \nu))^2]$$

where $x \sim N(0,\sigma^2)$, $k$ is a constant and $\nu \sim N(0,1^2)$ is the noise corruption to the information received by the second player. For a given strategy for the first player, $f(x)$, Witsenhausen has shown that the optimal strategy for the second player is

$$g^*(y \mid f) = \frac{E[f(x)\phi(y - f(x)]}{E[\phi(y - f(x)]}$$

where $\phi(\bullet)$ is a standard normal distribution function. The goal is to find the optimal $f(x)$. Witsenhausen has also shown that the optimal $f(x)$ has to satisfy: $E[f(x)] = 0$, $Var[f(x)] \le 4\sigma^2$ (cf. Lemma 1 in [10]) and non-decreasing function. Thus, we are only investigating functions that are symmetric about the origin, finitely supported over the interval $[-5\sigma, 5\sigma]$, bounded by $[-5\sigma, 5\sigma]$ and monotonic. In this paper, we are only investigating the benchmark instance, where $\sigma = 5$ and $k = 1/\sigma = 0.2$. Thus, we limit our search to functions that are symmetric to the origin. Due to the symmetry, the function is uniquely determined in the space of $[0, 25] \times [0, 25]$.

For this benchmark case, considerable data exists on the numerical values of the optimal $J(\theta)$. Since the conception of the counterexample, many attempts have been made on tackling it. Here is a brief summary of the major advances. Witsenhausen has shown in his original paper [10] that the optimal linear solution has a performance value of 0.96 yet the 1-step function he proposed has a value of 0.4042, which is far superior. Later, Banal and Basar [2] optimized over the 1-step

function and reached a value of 0.363. Through ordinal optimization, Deng and Ho [3] were able to further improve the performance to 0.1901. Most recently, Lee et al. [5] obtained a solution that has a performance value of 0.1673. This is the value that we use for comparative purpose in this paper.

## 4 Empirical Results

Here, we present our empirical results of GO on the two test problems.

### 4.1 Function Approximation

Three types of target function are used for testing. One is a staircase like function with a monotonic property, as shown in Figure 4. The second one is a hockey-sticker like function with monotonic property, as shown in Figure 5. The last one is a randomly generated jagged function, as shown in Figure 6. As stated earlier, the domain and range of this test problem is the same as the one used in the Witsenhausen counterexample. Furthermore, the discretization is kept the same. Thus, the search space is exactly the same. The only difference is the performance criteria.[14]

GO with scatter is tested first. The results are promising. For the target function M25, the best solution has a performance value of 0.94. In other words, on average, there is an error of less 0.05 for each of the 25 steps. That is less than 1 discretization away from the target function. This is, approximately, among the top $5 \times 10^{-25}$% quantile. Thus, it would have taken $2 \times 10^{26}$ sample, on average, to obtain a solution of similar quality using blind search. There is similar result for the target function MH25. The best solution has a value of 0.55. This is, approximately, among the top $5 \times 10^{-25}$% quantile. For the target function R25, the best solution has a value of 349.28. It is among the top $2 \times 10^{-19}$% quantile.

The solution from GO with scatter for the jagged target function, R25, is significantly worse than the other two monotonic target functions, in terms of performance. Nevertheless, the algorithm obtained a solution that is unlikely to be captured by random search. The difference in performance for the different target function is due to the nature of our algorithm. Let us elaborate here. Since we are starting from the coarser space, the functions with less variation between adjacent intervals, i.e., the monotonic functions, could be more accurately captured. Thus, a better performance would result. Hence, it would lead the algorithm to bias in the correct neighborhood. On the other hand, if a target function that is very jagged, it could not be easily captured in the coarser space. With poorer performance values, the algorithm could not easily identify the good neighborhood to sample, i.e., all the neighborhoods are equally bad. Thus, it would take longer to converge. Since the same amount of computational budget is imposed on all three target functions, the jagged one, R25, would result in poorer performance.

Notice, the function approximation problem is a convex problem, due to the performance criteria, see Equation (1). Applying GO with descend-scatter greatly improved the performance of the algorithm for the target function R25. In particular, it is able to deliver a solution with a performance of 2.17, which is among the top $2 \times 10^{-19}$% quantile. With the monotonic target functions, M25 and MH25, GO with descend-scatter can deliver solutions of similar quality as the ones from GO with scatter with less computational budget. The enhanced performance is mainly due to the application of descend technique in a convex problem.

### 4.2 The Witsenhausen Counterexample

The Witsenhausen counterexample [10] is the second test problem The counterexample has been outstanding since its conception, 1968. Recently, Lee et al. [5] have arrived at a solution that is 13% better than the previous known best solution. In our test, we compare our search result against theirs. The important difference is that in [5], their parameter space is real valued whereas ours is discrete. During our initial run, we have set $\delta$ to be 25 and $\Delta P$ to be 0.1%. We sample 1,000 designs in each space and keep the top-50 as the good enough designs.

Our best solution has a performance value of 0.1717 vs. 0.1673 from Lee et al. [5].[15] We arrive at this solution with only 24,251 evaluations and samplings. Our solution comes closer to theirs. One important distinction is that our algorithm is fully automated whereas Lee et al. used much human intervention during their search process. To further demonstrate the quality of our solution from GO, we uniformly generated 25,000 samples from 25-step space — roughly the same amount of computational budget as GO. The best solution that we have obtained though the blind search has a performance value of 0.3717 which is far from our best.

Order statistics result shows that the 25,000 samples above, on average, marks the quantile 100 ($i$ / 25,001)%. Using this order statistics result, we take the top-5,000 points from the samples to extrapolate the quantile of our top solution. As it turns out, our solution belongs to top 0.000004% ± 0.00000009%. Figure 7 shows the extrapolation results. Using GO, the total computational resource used is 24,251 evaluation and sampling. The order statistics result above suggests, on average, blind search (random sampling) would take about 25 million samples in order to obtain a design as good as ours.

[14] We set $\delta$ to 25 and $\Delta P$ to be 0.1% like in the Witsenhausen counterexample.

[15] This result is obtained using GO with scatter. Similar result is also obtained by GO with descend-scatter.

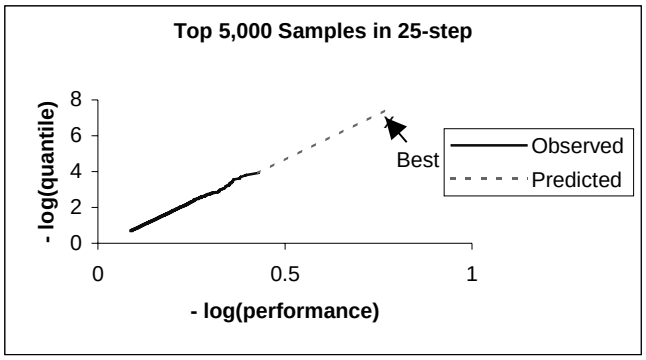


**Figure 7. Prediction of the Quantile of the Best Solution from GO**

## 5 Comparison to Traditional Descent Method

In an attempt to illustrate the usefulness of GO, we compare our search result to the traditional descent method. In particular, we are going to use the Witsenhausen counterexample to demonstrate the effectiveness of our algorithm.

Since we do not have continuity due to the fact that the solution space is discretized, we could not compute the gradient but only the finite differences. Instead, we compute the performance values along the 25 major axes. We move along the direction where there is a change. The step size is determined using Armijo algorithm. To make it a fair comparison, we allow the same amount of computational budget, 24,251 performance evaluations. Then, we compare the two results. We use 11 different functions as the starting point of the descent. In particular, we take the best function that we have found in 1-step and realize it as a 25-step function where all the function values are the same. Furthermore, we use the top-10 functions from the 25,000 samples which we obtained earlier as starting points. All of the 11 functions are trapped in the local minimums. The best among them have a value of 0.3639 which is far from the best from GO with scatter which has a value of 0.1717. As one can see, the solution from GO is far superior to the solution from the descent method given the same amount of computational allowance. Once more, this shows the difficulty of the Witsenhausen counterexample and its nonconvex nature.

## 6 Analysis and Insights to the Parameters

In this section, we explore the interactions between the parameters in the algorithm and the search result.[16] In particular, we are interested in the following pairs relations: $\delta$ and the search result; and $\Delta P$ and the search result. Due to the time constraint, we are only able to investigate the effect of the parameters on the function approximation problem but not on the Witsenhausen counterexample. We believe that the results are general and they would continue to be true in the Witsenhausen counterexample.

### 6.1 Effect of $\delta$

In this section, we present the effect of $\delta$ on the search result itself.[17] The parameter $\delta$ defines the size of the neighborhood of a good enough design, top-50 designs. As $\delta$ decreases, the size of the neighborhood decreases. If $\delta$ is too small, the performance may improve slowly, similar to being trapped in a local minimum. This could be seen in Figure 8. We set $\delta$ = 10 and 25. As we get deeper into our search, it is evident that when $\delta$ = 10 the performance value of the best function is not improving as fast as when $\delta$ = 25. This is similar to the effect of small step size in a descent algorithm.

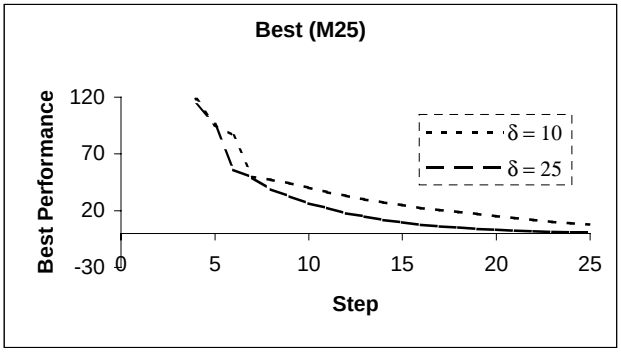


**Figure 8. Effect of Small $\delta$ on Target Function M25**

On the other hand, as $\delta$ increases, the size of the neighborhood increases as well. Increasing $\delta$ enlarges the neighborhood of the good enough designs. If $\delta$ is too large, then the random sampling from the neighborhood might not yield a better design. Here, we set $\delta$ = 25 and 50. As seen in Figure 9, the search results in better solutions when $\delta$ = 25 compare to the results from $\delta$ = 50. Thus, the search with $\delta$ set to be 25 outperform the case when $\delta$ = 50 in the space of 23-step and onward. This figure also shows the possible advantage of a time varying $\delta$ which is consistent with intuition.

[16] The discussion here is based on the result using GO with scatter. Similar observations hold true for GO with descend-scatter.

[17] The value of $\Delta P$ is set to be constant at 0.1% during this investigation.

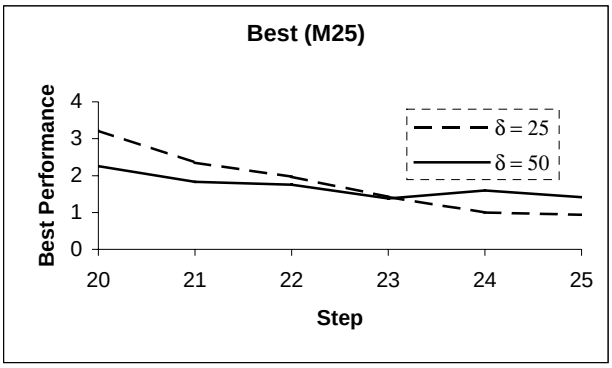


**Figure 9. Effect of Large δ on Target Function M25**

### 6.2 Effect of Δ*P*

The quantity Δ*P* defines the boosting probability.[18] It is a very difficult process to determine the optimal value for each problem. If Δ*P* is too large, we run into the possibility of being trapped in a local minimum early in the search. If Δ*P* is too small, we run the possibility of slow convergence on the optimal solution. This is similar to the parameter "temperature" in simulated annealing which controls how far a jump is allowed from the current solution. Below is a summary of the search results as we vary the parameter Δ*P*.

To demonstrate the case when Δ*P* is too small, we compare the cases when the values are set to be 0.01% vs. 0.1%. As can be seen in Figure 10, during the initial phase, the search with the smaller Δ*P* has a better performance. Yet during the later stage, the search with the larger Δ*P* has a better performance. This is due to the difference in the size of the search space. Initially, the search space is small. Thus, the search algorithm can perform well without focusing on a particular sub-set of the space. As the number of step increase, the size of the search space increases as well. Without increasing the probably of sampling from a sub-set of the space, it becomes more difficult to search in such a big space. Thus, the poor result in Figure 10 with small Δ*P*.

[18] The value δ is set to be constant at 25 during this phase of our investigation.

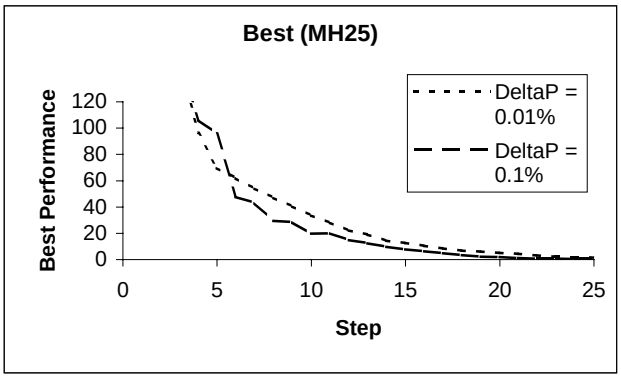


**Figure 10. Effect of Small Δ*P***

As Δ*P* becomes larger, the probability of sampling from the good enough designs increases more rapidly. As seen in Figure 11 where Δ*P* = 0.1% and 1%, during the initial phase of the search, the search result from Δ*P* = 1% is poorer than the one from Δ*P* = 0.1%. As we get deeper into the search, the search space becomes larger, the large value of Δ*P* focuses more on the good enough designs. This helps to reduce sampling in a large unwanted space. Thus, toward the end of the search, both search process from both values of Δ*P* results with similar performance value although Δ*P* = 0.1% has better designs earlier during the search.

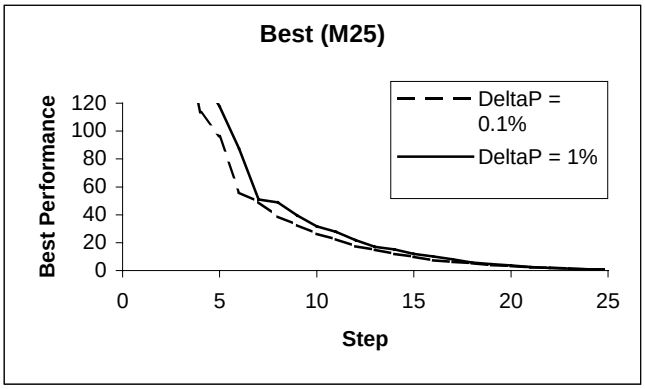


**Figure 11. Effect of Large Δ*P***

## 7 Discussion and Conclusion

In this paper, we have developed a new search algorithm, granular optimization. In Section 3.2, we have shown its potential by comparing our results, which are based on a fully automated algorithm, to the result by Lee et al. [5], which has much human intervention, on the well known Witsenhausen counterexample. The result from GO is very close to the one found by Lee et al. Furthermore, in Section 4.2, we compared the result from GO to the one from the descent method on the counterexample. It shows once again that GO is a very general approach, even when faced with a problem with many local minimums.

Although this paper has shown the potential worth of GO in search of optimal function, there is still

much we do not understand. In particular, how far can we relax the assumption of Lipschitz condition? We also will further investigate the proper parameter settings. Moreover, we need to look at problems where noise is a part of the function evaluation. Our ultimate goal is to be able to apply this to problems of higher dimensions — to face the curse of dimensionality.

## Reference


[1] Baglietto, M., T. Parisini and R. Zoppoli, "Nonlinear Approximations for the Solution of Team Optimal Control Problems," *Proceedings of CDC 1997*, San Diego, **5**:4592-4594, 1997.

[2] Banal, R. and T. Basar, "Stochastic Teams with Nonclassical Information Revisited: When is an Affine Law Optimal?" *IEEE Trans. on Automatic Control*, **32**(6):554-559, June 1987.

[3] Deng, M. and Y.-C. Ho, "Sampling-Selection Method for Stochastic Optimization Problem," *AUTOMATICA*, **35**(2):331-338, 1999.

[4] Ho, Y.-C., R. S. Sreenvias and P. Vakili, "Ordinal Optimization in DEDS," *Journal of Discrete Event Dynamics Systems*, **2**(1):pp.61-88, 1992.

[5] Lee, J. T., E. Lau, and Y.-C. Ho, "The Witsenhausen Counterexample: A Hierarchical Search Approach for non-Convex Optimization Problems," to appear in *IEEE Trans. on Automatic Control*.

[6] Lee, L. H., T. W. E. Lau and Y.-C. Ho, "Explanation of Goal Softening in Ordinal Optimization," *IEEE Transaction on Automatic Control*, **44**(1):94-99, 1999.

[7] Narendra, K. and M. A. L. Thathachar, *Learning Automata: An Introduction*, Prentice Hall, 1989.

[8] Ozden, M. and Y.-C. Ho, "A Probabilistic Solution Generator of Good Enough Designs for Simulation," submitted for review, *European Journal of Operations Research*, 1999.

[9] Poznyak, A. S. and K. Najim, *Learning Automata and Stochastic Optimization*, Springer, 1997.

[10] Witsenhausen, H. S., "A Counterexample in Stochastic Optimum Control," *SIAM Journal on Control*, **6**(1):131-147, 1968.

[11] Woplert, D. G. and W. G. Macready, "No Free Lunch Theorems for Search," *IEEE Trans. on Evolutionary Computing*, April 1997.

[12] Zadeh, L. A., "Fuzzy Logic = Computing with Words," *IEEE Trans. on Fuzzy Systems*, **4**(2):103-111, 1996.


**Appendix 1. List of Variables In the Paper**

- $\theta$: a solution, a function, a mapping.
- $\Theta$: the search space.
- $J(\bullet)$: the performance criteria.
- $n$: the number of steps in a function.
- $N$: the maximum number of steps allotted.
- *CProb*: the probability of sampling from the entire space.
- *Left*: the left boundary of the bounded domain.
- *Right*: the right boundary of the bounded domain.
- $\delta$: the amount of deviation from a particular allowed, the neighborhood.
- $\Delta P$: the boosting probability.
- *Top*: the number of top designs as the good enough designs.
- *Sam*: the number of samples taken at each iteration.