MODIFIED TABU SEARCH TO SOLVE COMPLEX, CONTINUOUS-VARIABLE PROBLEMS THROUGH GRADIENT ANALYSIS AND CONTOUR JUMPS

A Thesis presented to the faculty of the Graduate School at the University of Missouri-Columbia

---

In partial fulfillment of the requirements for the degree, Masters of Science

---

By

TYLER BURNS

Dr. Cerry Klein, Primary Advisor

Dr. James Noble, Academic Advisor

Dr. David Maslar, Academic Advisor

JULY 2014

Approval Page

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

MODIFIED TABU SEARCH TO SOLVE COMPLEX, CONTINUOUS-VARIABLE PROBLEMS THROUGH GRADIENT ANALYSIS AND CONTOUR JUMPS

presented by Tyler Burns, a candidate for the degree of *Masters of Science*, and hereby certify that, in their opinion, it is worthy of acceptance.

Professor Cerry Klein

Professor James Noble

Professor David Maslar

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Cerry Klein, who played an integral role in the completion of this thesis. His patience, guidance and counseling provided an atmosphere that was imperative to the successful completion of this research.

In addition, I would like to thank my other advisors Dr. James Noble, and Dr. David Maslar. Their intense questioning and high expectations helped me achieve a thesis that I could proudly stand behind.

1. Introduction

This paper introduces a novel method of solving np-hard problems with continuous variables by using a modified Tabu search algorithm. Traditionally, Tabu search is a metaheuristic that deals with binary or integer variable type problems and finds "good" solutions to the problem.

This study explores the Tabu search algorithm and presents a methodology by with the algorithm can be modified to handle continuous variables as well as integer and binary. The modifications include an appropriate method of defining the neighborhood solution sets and then an effective manner of moving through the neighborhood to converge to an acceptable solution as quickly as possible.

In addition to moving through the neighborhood, a method of locating and moving to local optima is included. From there this technique will incorporate a method of "jumping" to the base of a new, improving local optima in the state space. From there the algorithm will return to locating and reaching the local optima point.

In this fashion, the modified Tabu search will explore the state space for continually improving points and will "jump over" areas that, while feasible, are not worth exploring. This method saves computation time as well as achieving a more direct route to an acceptably good solution.

To test the modified Tabu search, a common problem from financial analysis is defined. This problem is easy to understand, yet hard to solve without the specific heuristic that has been developed to solve it optimally. This function is used as a

benchmark to test the outcome of the modified Tabu search against the known optimal value of the problem that can be found using the predetermined heuristic.

2. Literature Review

The idea of heuristics has been used to solve difficult problems ever since humans first began encountering problems without an immediately identifiable solution. The general idea is how to discover an optimally good solution with the least amount of work involved. As mankind progressed, the types and breadth of problems grew to a point where many problems are unsolvable without the aid of modern day computers.

As the problems increased in difficulty, the need for efficient methods to solve the problems became more and more prevalent. Many of these difficult problems are solved using metaheuristics rather than an actual optimizing program. An optimizer, while guaranteeing an optimal solution will most likely take more time to compute than is feasible for the scope of the problem. Instead, a metaheuristic is implemented to get to a "good" solution that, while not optimal, will still be an acceptable result.

There are many types of metaheuristics, coming from many different inspirations. A large portion of them are inspired by nature. Xin-She Yang gives a good account of the history of metaheuristics that develop in the mid to late 1900 in his paper, "Review of Metaheuristics and Generalized Evolutionary Walk Algorithm" (Yang 2011). He shows the early progression of integrating metaheuristics into normal computing algorithms.

Of all the metaheuristics, Tabu search is one of the oldest along with others such as simulated annealing (SA) and genetic algorithms (GA). F. Glover and M. Laguna were the first to present it as a valid technique in their book, *Tabu Search* published in 1997, although they first began work on the algorithm in 1986. (Glover and Laguna 1999). Tabu search is a powerful approach to difficult combinatorial problem as it provides the methodology to escape local optima and search the entire solution space for an acceptable solution (Gendreau and Potvin 2010).

While Tabu search is a well-known and implemented metaheuristic, very little research has been done as far as using the algorithm for problems with continuous variables. In 1992, N. Hu took the first step of adapting Tabu for continuous random variables by including the idea of using random walks to define subsequent points within the neighborhood. However, there is very little mention of the Tabu list or how it was generated. The list is generated by adding the set of 'steps' that the algorithm goes through to the Tabu list. In this way the algorithm avoids local minimum and explores the entire state space (Hu 1992).

Then, Siarry and Berthiau created their own method of handling these problems in their paper, "Fitting of Tabu Search to Optimize Functions of Continuous Variables." They were dissatisfied with the work that Hu conducted on the basis of it being too far removed from the original structure of the Tabu algorithm. In their paper, they proposed treating the current solution, *s* in the state space as a point contained in *k* concentric circles or spheres. The neighborhood would then be defined as *k* random points within these circles. The objective function would be evaluated at each of these points and the algorithm would then move to the best of the *k* points, even if it was less than the current

optimal solution. Then, the ball located at *s* with radius h_0 would be added to the Tabu list (h_0 being the radius of the inner most concentric circle). (Siarry and Berthiau 1997)

Siarry then revisited this idea with Chelouah in 2010 in the paper, "Tabu Search applied to global optimization" in which they changed the spheres to concentric cubes. Their additional contributions involve breaking the algorithm into two parts, *diversification* and *intensification*. During *diversification*, the state space is evaluated and broken up into different 'promising areas' based on the average objective function values of points in that region. Then, the best 'promising area' is selected for evaluation in the *intensification* step. In this step, the Tabu search presented above is conducted with the addition of an extra parameter called the reduction criteria. If this criteria is met, the size of the Tabu cube and the size of space where neighbors are defined are both reduced. This allows the algorithm to slowly refine its search to a smaller and smaller area and will eventually reach a "good" solution within the 'promising area' (Chelouah and Siarry 2000).

Siarry then continuous his refinement in the paper, "Continuous ant colony system and Tabu search algorithms hybridized for global minimization of continuous multi-minima functions" with Karimi and Nobarhari. In this paper, they attempt to combine the idea of a Tabu ball or cube, *diversification/intensification*, and promising lists with the well-known ant colony optimization in the algorithm Tabu Continuous Ant Colony System (TCACS). This algorithm maintains much of the structure of the Tabu algorithm already discussed, but adds a flavor of the ant colony optimization in the fact that the move directions for the neighborhood are randomly generated vectors (ants) that move the current solution in random directions and random distances. This forces the

algorithm to be much more effective in enumerating the entire state space. Once at the new locations, the Tabu list is updated with the sphere of the old solution and the new promising areas are identified according to the procedure explained above (Karimi, Nobahari et al. 2010).

3. Problem Statement

The extensive literature on continuous optimization presents many effective methods of solving optimization problems. However, even with all of these techniques, there are still some problems that cannot be handled by these methods. This is because of the parameters or "metas" that these techniques require to run effectively. In all these cases, if the "metas" are not set correctly, the algorithm will most likely fail to achieve a sufficiently good solution.

This paper seeks to propose a method that progressively learns the structure of the optimization problem and then adjusts the movement through the solution space accordingly. This is done without the need for "metas" that would require the user to have a predetermined knowledge of the behavior of the problem.

Therefore, the proposed solution has two main uses. First, this process may not be as effective as other methods for well studied and understood problems. However, it will be usable for any problem that is not well understood as an exploratory method. This will make it useful for initial testing and for finding a quick, good solution without extensive research. Secondly, this process will be usable as a method for finding "good" initial

solutions that can be used for other techniques. This is useful as a good initial solution will help increase the effectiveness of other metaheuristics.

4. Solution Methodology

The literature on Tabu search for continuous random variables defines its Tabu list by the so-called "Tabu Spheres" which prevents the future neighborhoods from being assigned to points too close to the current solution. In this way, predefined areas in the state space are Tabu for the algorithm.

However, consider an optimization problem where there is a multitude of local optima and little to no apriori knowledge of the state space. If the experimenter decides to use a Tabu search similar to what is described in Siarry's papers, then the experimenter will first have identify an appropriate k number of neighborhood solutions within k spheres with radius h_i $(i \text{ in } k)$. This could be difficult to obtain the correct parameters for the specific problem without rigorous trial and error to find the most appropriate values. The TCACS method aids the user by dynamically adjusting these parameters as the algorithm runs. However, the added formulation and techniques used make the algorithm more complex to understand and draws the algorithm further from the original (simple) Tabu search method. It also adds significant time to the overall solution time.

In contrast, this paper seeks to define a more simple method of defining the state space and guiding how the algorithm moves throughout it. Additionally, it

attempts to find a quicker route to a ‘good’ solution in fewer iterations of the algorithm.

In order to accomplish these items, we first identify the method of choosing a move direction. To move points, it is proposed that the most increasing (for max) direction is chosen if available. Then, the algorithm search moves along that direction until a) a boundary condition is met, or b) the OF value stops increasing. If the search function is initially increasing, then the search will keep moving along the direction until a local optima (or saddle) is found. The new point is found at a new local optima.

Once at the new optima, the gradient is then reevaluated. If there is a direction where the gradient increases with respect to the objective function, the algorithm is at a saddle and chooses a direction with the positive gradient (for max) to move along. If there is no positive gradient, the algorithm is at a local optima.

To proceed, a new constraint is added to prevent the algorithm from moving to a less optimal optima. This constraint runs along the same dimension as the objective function and is defined as:

$$\text{OFvalue} >= M$$

In this equation, M is the value of the objective function at the current local optima. Notice that the new constraint will be a linear plane that will be referenced to the same dimension as the objective function equation. Therefore, for some range of M, there will be a number of intersections of the objective function and the new

constraint. This will be all of the solutions where the objective function will be the same as the objective function value at the current point.

After the algorithm has identified the level of M, a "check" is then run to find the intersection of the objective function equation and the equation:

$$\text{OFvalue} = \text{M} + \varepsilon$$

In this equation, ε is a very small increment relative to the problem at hand. The intersection(s), if they exist, must be checked for feasibility. If a feasible intersection is discovered, it is known that some solution exists that improves on the current location. If not, then the algorithm stops and the current solution is optimal.

In this way, the algorithm will continue to jump from local optima to the base of an improving local optima. From there, the gradient is unknown and must be recalculated for a new move direction. Therefore, at the base of the new optimal, the process restarts and the algorithm climbs to the top of the new local optima.

With this methodology, there is a slight issue that must be dealt with. After leaving an optima, the algorithm may be moving in a direction in which there are no other optima that exist, and are feasible (the new constraint will make many of the optima in the state space infeasible). To correct for this, the algorithm must have a method of bending or turning the move direction so that it will reach another optima that is still feasible.

To accomplish this, the algorithm has the flexibility to be manipulated so that multiple variables may be adjusted together to create a new direction for the algorithm to travel along. This new direction may or may not be parallel to the

direction of a single variable, but a combination of multiple variable moves. In this way, the algorithm can more efficiently reach the local optima.

Next, we redefine the Tabu list from a local area around the current point to a set of "levels" in which the state space will never again be evaluated. This is achieved by the added constraint which eliminates, or makes infeasible, any solution that is not going to lead to an improving objective function value. In a way, the new Tabu list becomes a set of "steps" below the current level of the optima. However, unlike the conventional Tabu method, areas on the Tabu list in this algorithm are permanently Tabu. This is because the new constraint presents a method for the algorithm to jump to a new improving area without "backtracking" to less optimal points. In this way, the algorithm leaves behind a trail of local optimas through which it will not return throughout the entirety of the Tabu algorithm. In addition, the areas, or "balls" similar to Siarry's "Tabu balls" are recorded so that the general areas that lead back to the current optimas can be avoided.

In this way the search function goes through a series of 'optima hopping.' Starting with the first iteration, the algorithm will find the closest local optima to the current solution and move to that point. Then the search will move to the next nearest optima, taking into account the relative size of the optima (a larger optima further away may be weighted the same or greater than a close, small optima).

To visualize this motion, imagine an explorer in a mountain range at a time when the world is flooding. This would be a two dimensional problem as an explorer can have both latitude and longitude. Now, imagine that the explorer is on the top of a small hill, but he would like to get to the top of the tallest distant mountain.

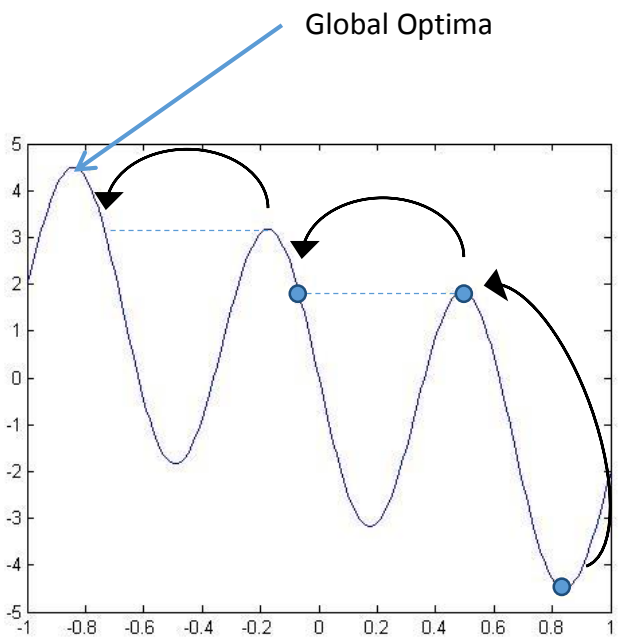


Figure 1: Algorithm Overview

Assume the world flooded so that the water is up to the level of the explorer's feet. The explorer can then get in his boat, and sail to where he can see land (the flood is helpful in this scenario). Once he reaches land, he climbs up to the very top with his boat. The flood again rises to the very tip of the mountain he is standing on (local optima). Again, he sees distant land above the water so he knows that he has not yet reached the highest point. He gets in his boat and sails again. This process repeats until the explorer can no longer see land and he knows he has reached the highest mountain peak.

By raising the flood level at each step, the explorer avoids having to climb back down the hill he is on and possibly climbing other mountains that may not be the tallest. Assuming, he is a fast rower, he can very quickly reach the tallest mountain peak.

Algorithm

In order to use this idea for optimization problems, a computer algorithm must be able to travel in the same way as the explorer. The general algorithm to achieve this is as follows:

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    'Create Initial Solution
    'Do Until No Contour Intersection is found at M+e
        'Do until no improving direction (gradient=0)
            'Find all Neighborhood candidates s.t. Tabu list
            'Eliminate non-Feasible Solutions (constraints, etc.)
            'Find Best Candidate (Steepest gradient)
            'Find Nearest Optima along path of best candidate solution
        'loop
        'If Cost(Best Candidate)<Cost(Best Solution) then best solution=bestcandidate
        'update current solution
        'Check for existing, feasible and improving contour points and jump to point
        'Update Tabu List
    'loop
    'Output Best Solution
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

Create Initial Solution- To Begin, the algorithm creates an initial feasible solution from which it will proceed to move through the solution space.

Do until no contour intersection is found at M+e- The algorithm will run until the point where it is determined that there are no more feasible solutions that improve the objective function from the currently identified best solution.

Do until no improving direction- This sub part of the algorithm runs to find the local optimum of the current point. It is run until the algorithm identifies that there is no direction in which the gradient is greater than zero.

Find all neighborhood candidates subject to the Tabu List- At this step the gradient is found in all directions. If there is a direction in which the gradient is greater than zero, then a point along that direction is identified that is close to the current point.

Eliminate non-feasible directions- All the identified points are tested for feasibility and eliminated if they are found to be infeasible.

Find Best Candidate- The gradient along all of the improving directions are compared and the direction with the highest/steepest gradient is chosen as the best candidate move direction.

Find nearest optima along path of best candidate solution- Then the algorithm continues to move along this direction in large increments until the OF-value begins to decrease. At this point, the algorithm moves back one step to the point where the OF-Value was still increasing. The increment length is divided in two and the process restarts. This process continues until the increment is sufficiently small and the current point is taken to be the local optima according to that variable that defined the move direction. At this point the gradient will be zero.

If Cost(Best Candidate)<Cost(Best Solution) then best solution=bestcandidate- If the new identified optima has a better objective function value than the previous best solution, then a new best value has been identified and it is recorded.

Update current Solution- All arrays and variables that were used to calculate everything up to this point are re-initialized and the solution that creates the best value is recorded.

Check for existing, feasible and improving contour points and jump to point- If there are no improving points (gradient=0) then the current position is a local optima and there is no logical direction for the algorithm to move. Therefore, the contour of the objective function is identified, and all the points where the objective function is equal to M+e are collected. If any of these points are feasible, then the algorithm jumps to this point, re-initializes the variables, and records the solution since by definition it is a improving point.

Update Tabu List- Add the current jump point and the local optima that was jumped from to the tabu list.

Output Best Solution- At the end, output the best identified solution as the optimally good solution.

5. Initial Testing

To prove the validity of this method, a two-step testing procedure was implemented. First, the algorithm was tested against a very basic, easy to compute problem. This is to show that the algorithm does indeed produce a sufficiently good answer to a given problem. Next, an attempt is made to implement the algorithm in a much more complicated problem which has a known, global optima. In this way the validity of the algorithm is displayed and the

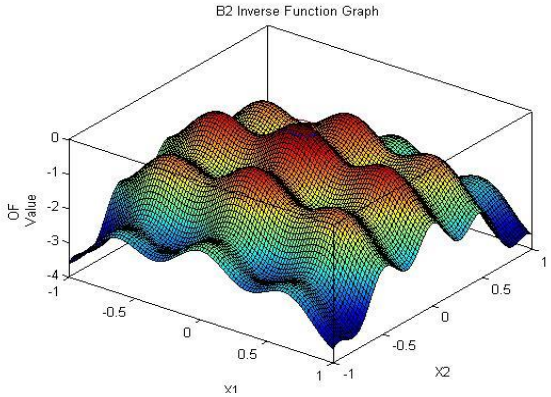


Figure 2: B2 Inverse Function Graph

increased difficulties that are presented as the problem difficulty increases are also apparent.

The first round of testing involves a random, non-linear, yet simple problem to model. The function used is the inverse of the B2 function and is defined with the equation as follows:

$$B2_{INVERSE} = -X_1^2 - 2X_2^2 + .3COS(3\pi X_1) + .4COS(4\pi X_2) - .7$$

As it is easy to see, the objective function is non-linear. The B2 inverse function has multiple optima with a global maxima at (0,0) and an OF-Value at the global maxima=0 (Karimi, Nobahari et al. 2010). Figure 1 is a graphical representation of the B2-Inverse function for both X1 and X2 are limited to be between -1 and 1.

To limit the state space of the problem, the following constraints are added:

$$-1 \leq X_1 \leq 1$$

$$-1 \leq X_2 \leq 1$$

$$X_2 \geq X_1^2$$

Like the objective function itself, the constraint set is also non-linear. To begin, the point (1,-1) is chosen as the initial solution (notice that this is not at all a good solution). This solution is one of the furthest feasible points from the optimal solution given the set of constraints. From this point, the optimizing algorithm is implemented to solve the problem.

First, the OF-value is calculated to be -3.6. Since this is the first step, and we are not guaranteed to be at any optima we proceed to find the nearest optima. The gradient at (1,-1) is determined. Using the $B2_{inverse}$ equation, the gradient can be determined to be:

$$\nabla B2_{\text{Inverse}}(X_1) = -2X_1 - .9\pi \sin(3\pi X_1)$$

$$\nabla B2_{\text{Inverse}}(X_2) = -4X_2 - 1.6\pi \sin(4\pi X_2)$$

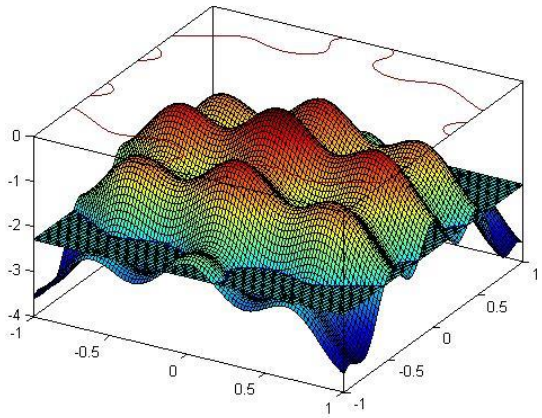


Figure 3: Intersection of M+e and Objective Function

Therefore, at (1,-1) the gradient is (-2, 4) so we choose to move first along the positive X_2 direction until the gradient becomes zero. This happens at (1, -0.9334). We check the gradient again and find that the gradient is now (-2, 0). Therefore, we move in the negative X_1 direction until the gradient in that direction is zero. This happens at (.6186,-.9334). The gradient is now (0,0) so the algorithm has reached a local optima.

At this point the OF-value is calculated to be -2.2875 which indeed improved on the initial solution. At this point however, there is no immediately obvious direction in which to move. Therefore, we add the M+e constraint where M=-2.2875. The value of e can be any small number compared to the problem scope. In this example, e was taken to be .0001. Figure 2 shows a graphical representation of this

new plane and the objective function. As the picture shows, the new state space includes only points that are improving when compared to the current solution.

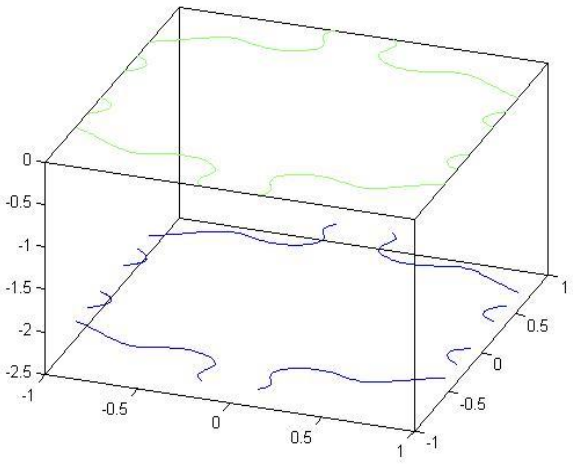


Figure 4: Contour Map

At this point, the algorithm must choose a new point to jump to in order to search for a new optima. Based on the new constraint, there are many different points on the plane that intersect the objective function. Using MATLAB, a contour map of these points was generated (figure 3).

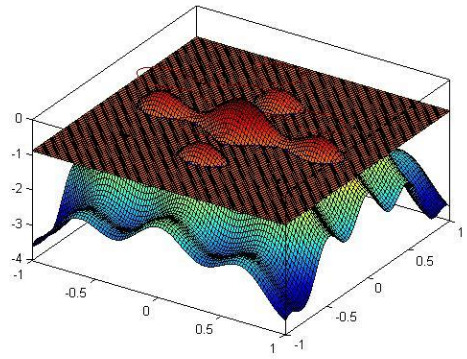
The points along the specific contour (M+e) are all potential jump points subject to the problem constraints. The modified Tabu algorithm moves along this contour looking for feasible points. The algorithm first evaluates the nearest point to the current point. If this point is feasible then it is accepted. Otherwise, it moves to the next closest point. This process continues until it finds the first, feasible contour point is identified. It then jumps to that point to continue the algorithm. If there are no feasible contour points at any step in the process then the current point is the global (feasible) optimum.

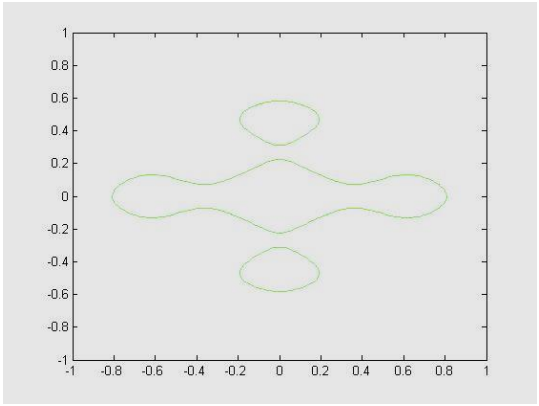
In this example, the nearest contour point is also feasible. Had it not been feasible, the algorithm would have skipped past this point and continued along the contour line until a feasible point is encountered.

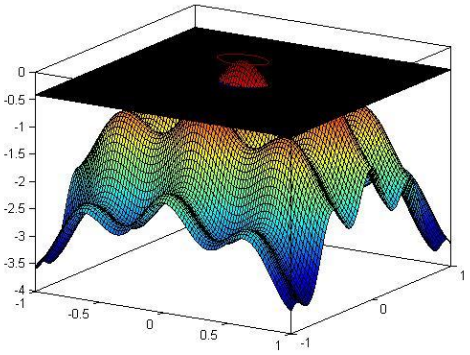
The closest, feasible contour point to the current location is at (0.6186, -.07356). By definition, this point has an OF-value=M+e, where M is -2.2875. From here, the algorithm restarts. A summary of the sequential moves is described in Table 1.

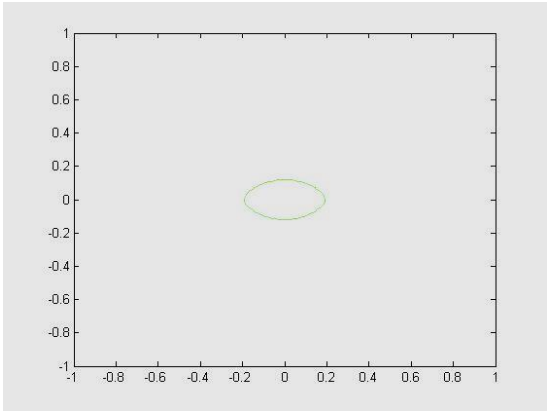
Table 1: Concurrent Algorithm Steps

Move	Current Gradient		Move Direction	New Point		OF-Value	Jump Point	
	delta-X1	delta-X2		X1	X2		X1	X2
2	0	4.306676	Positive X2	0.6186	-0.46953	-0.88281		
	0	0	None-Jump				0.61616	-0.13201
3	0	5.535098	Positive X2	0.6186	0	-0.41293		
	0	0	None-Jump				0.19259	0
4	-3.12864	0	Negative X1	0	0	0		
	0	0	None-Global Maxima Reached					









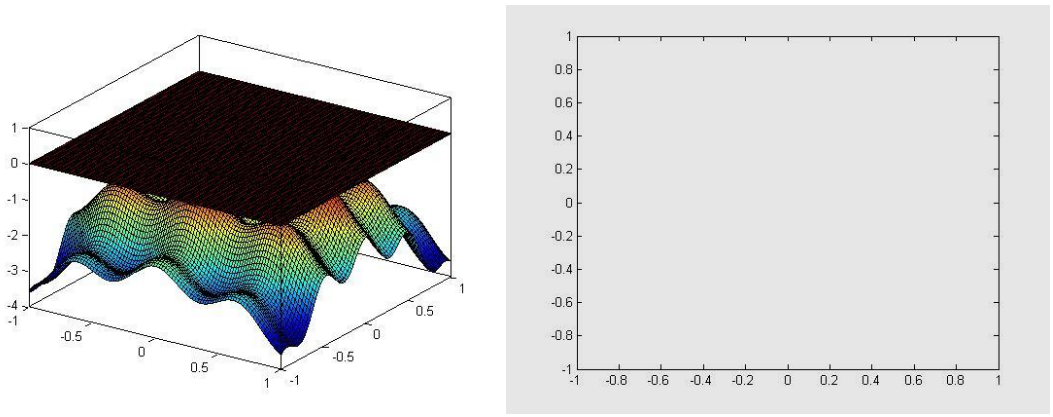


Figure 5: Graphs (Intersection and Contour) for concurrent algorithm steps

As the last set of graphs show, the last iteration of the algorithm results in a contour graph that is empty. Therefore there are no improving points in the objective function (feasible or not). The lack of improving contour points signals that the algorithm is at the global optimum point (0,0). Therefore, the algorithm stops and the objective value corresponding to the optimum point is calculated (OF-Value=0). Using this method, within four total moves, the algorithm can move from one of the worst possible points in the state space to the global maxima.

6. More Rigorous Problem

Formulation

For a more complicated, real world application test of the algorithm, a capital budgeting problem was created. This problem is relatively easy to solve by hand using general heuristics, but would be hard for a computer to solve without knowing the heuristics which solve it. This makes the problem an ideal test problem for the proposed algorithm as the global optimum can be readily identified and compared to the result of the problem using the new technique.

The problem is, given some initial amount of money *PRINC*, and a period contribution *CONT,* along with some number of required assets (*ASTS*) with depreciation (INT_{ASTS}), debts (*DBTS*) with interest (INT_{DBTS}), and investments (*INVS*) with interest (INT_{INVS}). What is the percentage of available funds in each period, $p_{i,n}$ that should be contributed to each asset, debt and investment to maximize the value (*ROI*) in period *n* of N. In this case, ROI will be calculated as:

$$\text{ROI} = \frac{\text{Total Worth}}{\text{Total \$ Invested}} - 1$$

Therefore, the objective function (OF) in period *n* is defined as:

(1)

$$\text{MAX ROI}_n = \frac{\text{RESV}_n + \sum_{i=1}^{\text{ASTS}} \text{ASTS}_{i,n} - \sum_{i=1}^{\text{DBTS}} \text{DBTS}_{i,n} + \sum_{i=1}^{\text{INVS}} \text{INVS}_{i,n}}{\text{PRINC} + n * \text{CONT}} - 1$$

For each period *j*, there is a set of $p_{i,n}$ (*i* in L where L is the sum quantity of all assets, debts, and investments +1) that define what proportion of last periods remaining money plus money added this period will be designated for each of the L elements in that period. There is also a $p_{i,n}$ associated with the remaining money left over after the money has been dispersed, $p_{R,n}$. The amount of this reserve is defined as:

(2) $$\text{RESV}_n = \text{PRINC} * \prod_{i=1}^{n} p_{R,i} + \text{CONT} * \sum_{i=1}^{n} \prod_{j=i}^{n} p_{R,j}, \ \forall n > 0, \text{where } p_{R,0} = 1$$

$$\text{RESV}_0 = \text{PRINC}$$

There are many constraints that could be added to the problem, but the ones that were chosen for experimentation are:

-non-negativity of proportions

$$p_{i,n} \geq 0 \quad \forall n \text{ in } N, \forall i \text{ in } L$$

-sum of proportions for a period must sum to 1

$$\sum_{i=1}^{L} p_{i,n} = 1 \quad \forall n \text{ in } N$$

-maximum investment proportion MAX_{INVSa} for a certain investment, *INVSa*

$$p_{INVa,n} \leq \ MAX_{INVSa} \ \forall n \text{ in } N$$

-amount contributed each period to an asset must be enough to cover a repurchase of the asset at the end of its lifecycle.

$$RESV_{n-1} * p_{i,n} \geq \ INT_i * ASTS_{i,0} \quad \forall i \text{ in ASTS}, \forall n > 0$$

The behavior of the values of $ASTS_{i,n}$, $DBTS_{i,n}$, and $INVS_{i,n}$ are as follows:

$$ASTS_{i,n} = \text{Max}(ASTS_{i,n-1} - ASTS_{i,0} * INT_i + p_{i,n} * (RESV_{n-1} + CONT), 0)$$

$$DBTS_{i,n} = \text{Max}(DBTS_{i,n-1} * (1 + INT_i) - p_{in} * (RESV_{n-1} + CONT), 0)$$

$$INVS_{i,n} = INVS_{i,n-1} * (1 + INT_i) + p_{i,n} * (RESV_{n-1} + CONT)$$

Or, in general:

(3) $$ASTS_{i,n} = \text{Max}\left(ASTS_{i,0} * (1 - n * INT_i) + CONT * \sum_{j=1}^{n} p_{i,j} + \sum_{j=1}^{n} p_{i,j} * RESV_{j-1}, 0\right)$$

$$\forall i = \text{ASTS, where } RESV_0 = \text{PRINC}$$

(4) $\mathrm{DBTS_{i,n}} = \mathrm{Max}(\mathrm{DBTS_{i,0}} * (1 + \mathrm{INT_i})^{n} - \sum_{j=1}^{n}[\mathrm{p_{i,n}} * (\mathrm{RESV_{j-1}} + \mathrm{CONT}) * (1 + \mathrm{INT_i})^{n-j}], 0)$

$$\forall \mathrm{i} = \mathrm{DBTS}, \text{where } \mathrm{RESV_0} = \mathrm{PRINC}$$

(5) $\mathrm{INVS_{i,n}} = \mathrm{Max}(\mathrm{INVS_{i,0}} * (1 + \mathrm{INT_i})^{n} + \sum_{j=1}^{n}[\mathrm{p_{i,n}} * (\mathrm{RESV_{j-1}} + \mathrm{CONT}) * (1 + \mathrm{INT_i})^{n-j}], 0)$

$$\forall \mathrm{i} = \mathrm{INVS}, \text{where } \mathrm{RESV_0} = \mathrm{PRINC}$$

For convenience, a summary of the variables and their definitions is included in the table below.

Variable	**Description**	**Equation (if applicable)**
PRINC	The initial amount of money that is available to spend on the given options at n=0	
CONT	The amount of money that is contributed each period as additional funding for the given options	
$RESV_n$	At the end of period n, the amount of money left over that was not allocated to a specific project	(2)
$ASTS_{i,n}$	The value of asset i in period n	(3)
$DBTS_{i,n}$	The value of debt i in period n	(4)
$INVS_{i,n}$	The value of investment i in period n	(5)
L	The sum of all assets, debts and investments	
INT_j	The interest rate associated with project j in L	
n	The period in N (all periods)	
ROI_n	The value of ROI in period n. This is also the objective function value	(1)
$p_{i,n}$	The proportion of available funds that are allocated to project i, in period n.	
MAX_{INVSa}	A made up value that is used to limit the proportion of available funds that can be allocated to INVSa in all periods.	
M	Objective function value at current iteration of algorithm	
ε	A small increment holder that is used to check if the objective function intersects the state space anywhere higher than M	

Table 2: Variables and their descriptions for capital budgeting problem

Using this structure, the problem was evaluated using the proposed search method. The specific problem used to evaluate the modified Tabu search consisted of 3 assets all with current values and depreciation rates, 3 debts with current amounts and interest rates and 3 different investments with current values and interest rates. Each of these 9 items has a variable attached to it for each period, and the problem has a horizon of 19 periods. This results in a total of 171 variables with each variable of a future period relying on the values of earlier variable values. The value of PRINC was set to be $50,000 and CONT was set to $5,000.

Implementation Difficulties

Notice that this problem is quite a bit more intricate than the B2 equation explored earlier. Perhaps the biggest and most complicating difference is the number of variables (171 versus 2). In practicality, this is still a small problem as typical real world problems can have variables in the count of tens of thousands if not more (Kirkpatrick and Vecchi 1983). Therefore, it is important that this algorithm be translatable through problems of multiple dimensions.

However, as problems increase in size, the explicit definition of the objective function becomes harder and harder (if not impossible to define). This complicates the algorithm, as both the gradient and the contour lines are calculated using the objective function.

7. Techniques for Complex Problems

There are two main issues in the algorithm that don't work for problems that aren't written explicitly. First, the algorithm requires the gradient in order to find appropriate local move directions which will lead to the local optima. Secondly, the algorithm requires a method of locating the contour points along which the objective function is equal to the value M+ ε. Both of these items are impossible to calculate without an explicit definition of the objective function.

Gradient Approximation

The first issue is determining the gradient of according to each variable at any given point. In the B2 example, an equation for the gradient was easily identified using basic calculus techniques. However, a similar equation cannot be written for the capital budgeting problem. Fortunately, we can avoid this issue by specifically testing the effects of small increments of the variables on the OF-value. When calculating the gradient, we add a small value (again we use ε) to the variable at our current point and record the change in the objective function (ΔOF).

Since we are not measuring the gradient at a specific point but approximating the gradient by the slope across minor increments, we must also subtract ε from our current point and record ΔOF (If we are currently at a local optima, and only one direction is considered, then our gradient approximation would be incorrect). From this point we can simply compare the respective increases or decreases incurred by

moving a variable by a small increment to identify which direction is the most beneficial to travel along.

A point that yields a decreasing variable effect from both directions would be a local maxima with respect to that variable, and two positives would be a local minima. However, a point that has both a positive and a negative effect resulting from the increments would be improvable by moving along the direction that caused the positive effect on the OF-value. Using this technique, we obtain a pseudo-gradient that can be used to identify a "good" direction in which to travel. Moving in this manner for every variable until all points are at local optima yields a local optima for the optimization problem.

Contour Approximation

The more complicated issue is calculating the contour points (intersection between M+ ε and the objective function). In the B2 problem, the contour points were easily calculated using basic calculus. However, for the capital budgeting problem, this calculation is much more complicated. Consequently, there is no simple way to obtain the contour lines or points necessary for the algorithm to execute. Luckily, there is a bit of leeway that we can use to our advantage.

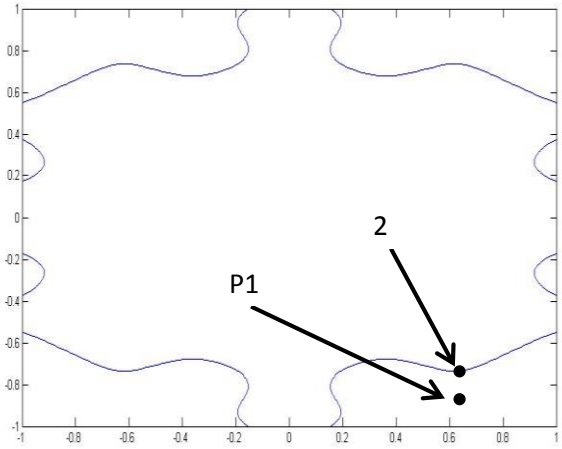


Figure 6: Vicinity of Contour Points

To visualize, first examine the contour line of the first iteration of the B2 example. From the gradient analysis in iteration 1, the algorithm reached the point P1 in the figure. From there, the contour line was drawn where the objective function was equal to M+ε. According to the algorithm, the next step is to move to the closest contour point which would be P2 in the figure.

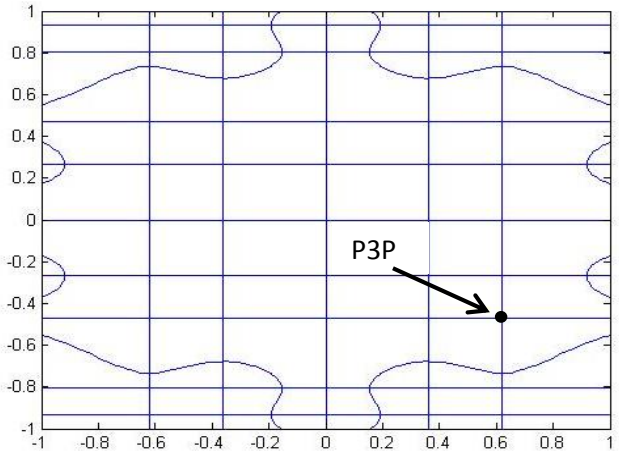


Figure 7: Gradient Lines and Contour Points

However, the next step of the algorithm is to repeat the gradient search which will move the point to the local optima of the point P2. For this problem, we overlaid a map of the lines where the gradient is equal to zero on top of the contour map. This gives a grid like graph where each intersection of the grid is a local maxima or minima.

As the graph of the objective function shows, the point P3 is a local maximum in the vicinity of the point P2. In fact, a gradient analysis at P2 will result in a move that will relocate the point to P3. However, this is not true only for the point P2. In fact any point within the local grid to the local maxima at P3 will also have move directions (based on the gradient) that will relocate the points to P3.

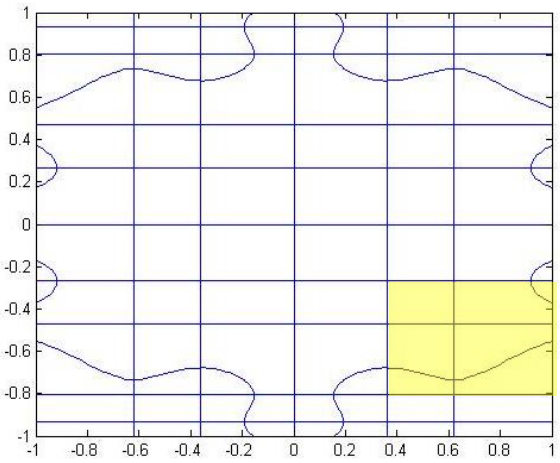


Figure 8: Critical Region

Based on this observation, the job of finding the exact contour line of M+ε becomes trivial. Instead, the algorithm only needs to find the general area where the contour line passes through. In this way, same local maxima will still be reached.

In order to approximate the region which contains the contour line, the structure of the objective function must be exploited. However, similarly to the original issue, the structure of the function is difficult to determine without having the function itself explicitly defined.

Therefore, it is necessary to approximate the structure of the objective function with a function that can be more easily manipulated. To make an approximation, we must make some assumptions as to the behavior of the functions structure. We can use polynomial approximation as the skeleton of our multivariate approximation.

The general statement of our approximation is that we are assuming that the structure of the objective function is such that it can be described by a polynomial equation up to degree *m* with respect to each variable. (i.e. $Y = a_1X + a_2X^2 + a_3X^3 \ldots + a_mX^m$). The a_i's in the equation can be found by solving the system of equations:

(6) $$\mathbf{Y} = \mathbf{XA}$$

Where **Y** is the matrix containing the objective function values Y_i when the variable in question X=x_i:

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \end{bmatrix}$$

Y will be a *p* x 1 matrix where *p* is the number of points in the set that are used to approximate the surface. The number *p* will get larger as the algorithm progresses and more points are added to the approximation set.

A is the *m* x 1 matrix containing the coefficients for each of the polynomial terms:

$$\mathbf{A} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \end{bmatrix}$$

Finally, **X** is the *p* x *m* matrix holding the polynomial variable descriptions for the variable at each of the *m* degree levels. Its structure is as follows:

$$\mathbf{X} = \begin{bmatrix} x_1 & x_1^2 & \cdots \\ x_2 & x_2^2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

Since **X** is not necessarily a square matrix, equation (6) above cannot be multiplied by $\mathbf{X^{-1}}$ on both sides. Therefore, the matrix **A** must be solved for using the following equation:

$$\mathbf{A} = (\mathbf{X^T X})^{-1}\mathbf{X^T Y}$$

Or similarly, **A** can be solved for by solving the system of equations:

$$\mathbf{X^T X A = X^T Y}$$

With an approximation with polynomials of degree *m*, the inverse matrix in this equation will be of size *m* x *m*. Therefore, the smaller degree approximation will be less precise but will be easier (faster) to compute. The degree of polynomials used for the approximation is left up to the user as a higher degree of polynomial approximation will be harder for the computer to compute and will therefore take longer.

This matrix **A** needs to be computed for each of the variables in the problem. In this way, we will obtain a list of *m* coefficients representing how the objective function value will vary with respect to each variable. Therefore, the total number of coefficients will be:

$$m * \mathrm{N}$$

Let $\widetilde{\mathbf{X}_\mathbf{i}}$ be the *m* x 1 matrix containing all the polynomials corresponding the variable, X_i, such that:

$$\widetilde{\mathbf{X}_\mathbf{i}} = \begin{bmatrix} x_i \\ x_i^2 \\ \vdots \end{bmatrix}$$

For our approximation, it is sufficient to say that the function that represents the objective function structure is:

$$\tilde{Y} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{A^T}\widetilde{\mathbf{X}_\boldsymbol{i}}, where\ N = total\ number\ of\ variables$$

This can be written out as:

$$\tilde{Y} = \frac{1}{N}(a_{1,1}X_1 + a_{1,2}X_1^2 + \cdots + a_{1,m}X_1^m +$$

$$a_{2,1}X_2 + a_{2,2}X_2^2 + \cdots + a_{2,m}X_2^m +$$

$$\vdots$$

$$a_{N,1}X_2 + a_{N,2}X_2^2 + \cdots + a_{N,m}X_N^m)$$

While this function has many terms, the gradient of this function is easy to compute. From this, we can choose to either set the derivative of $\tilde{Y}$ equal to zero hoping that the approximation is good enough that it will lead to a jump that is near the global optima, or we can set the derivative equal to the value M+ε to find a jump point that is similar to the actual contour point of the intersection of the actual objective function and M+ε.

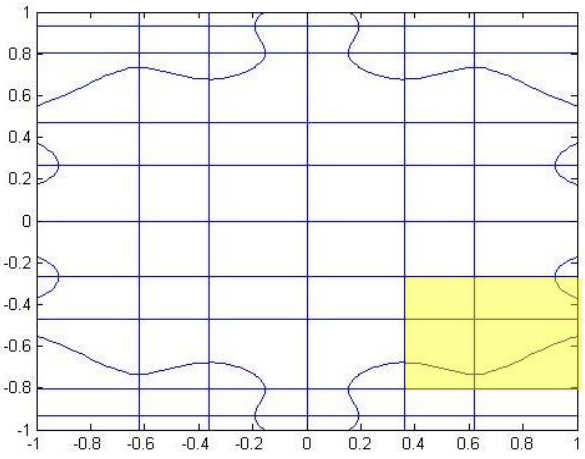


Figure 9: Critical Region for Tabu List

Whichever we choose, the success of the jump will depend on how well the objective function is approximated by $\tilde{Y}$. A poor approximation could just as easily lead to points that will not lead to improving local optima.

The nature of this approximation method is that the approximation becomes better with the degree of approximation *m,* and the number of points used to approximate, *p*. The degree *m* is limited only by what the user is willing to sacrifice in computing time (a larger *m* will result in a larger inverse matrix to be calculated). However, the number of points used to approximate the objective function does not significantly increase the computation time.

Therefore, for large problems, the addition of the "Tabu list" is incorporated into the algorithm. The Tabu list is very similar to the contemporary Tabu list in the sense that it stores information describing moves made and prevents the algorithm from returning to that area. This list will be referred to from here on as TABU. TABU differs from contemporary lists in the sense that once a point is on TABU, it never drops off the list.

The reasoning for this is that this algorithm uses a gradient search. Again, refer to the graph of the B2 contour with the gradient lines. Notice that once an area has been explored, there is no reason to return to any point local to that area as any of the local points will return the point to a maxima that has already been visited.

The second purpose of the list is it keeps track of all the jump points and optima points that the algorithm has visited along with their OF-values. This list is what will be used as input for the approximation function. This list is constantly updated as the algorithm progresses. Therefore, at the beginning of the procedure, the approximation will be poor. This initial approximation is made from a small sample of points that are collected at the very beginning of the algorithm. However,

as time progresses, the approximation will continually improve, driving the function closer and closer to the actual contour points/optimas.

8. Results

With the presented approximation methods the algorithm was applied to the presented capital budgeting problem. The parameters used to solve this problem were as follows:

m-degree approximation = 5

Max Iterations = 1000

OF-value at starting point= .5028

The hand solved value of the problem with the set of assets, debts, and investments resulted in an optimized value of ROI_n = .97 or 97%. This is the global optimal value identified for the problem by using an opportunity cost analysis of the assets, debts, and investment opportunities. The problem is solved by identifying which of all the options will result in the most positive gain and comparing it to the item which will result in the most negative loss in each period. In addition to any other constraints, most of the monetary contribution should be aimed at these projects.

Using the proposed modified Tabu search algorithm, the resulting maximized OF-value of the ROI_n = .9375 or 93.75%. This result is very close to the global

optimal value found by the heuristic above. The percent improvement is calculated as follows:

$$\%\text{Improvement} = \frac{.9375 - .5028}{.97 - .5028} \sim .9304$$

In other words, around 93% of the total possible improvement was achieved by the modified Tabu search heuristic. The error (97%-93.75%) was caused by inefficiencies in our approximation methods, specifically the OF-function approximation, which prevented the algorithm to effectively reach the global optimal value. A better approximation method or a higher degree approximation will help to close the gap between these two values.

9. Conclusions and Remarks

In this paper, a method for handling large scale problems with continuous random variables is presented using the gradient to reach local optima and then a contour analysis to "jump" to the base of other, improving optima. In simple problems, it was shown that the algorithm can reach a good local optima in a very short amount of iterations. However, this concept is harder to implement in higher dimensions. The success of this algorithm lies with how well the surface of the objective function can be approximated by a general, explicit function.

Further, improving research could include a more efficient way to calculate an approximation for the gradient at a specific point for a problem with n-variables where n is large. However, the more beneficial research would be to develop a

method of identifying or approximating the structure of an objective function so that a set of contour points can be more easily developed. This could include either a way to explicitly define the existing objective function or to develop a method of approximation that accurately describes the tendencies of the actual objective function.

The method of approximation proposed in this paper has some key flaws. Firstly, the assumption that the actual objective function follows structure similar to the polynomial function may or may not be appropriate for a given problem. First and foremost, this structure ignores all interaction effects between variables and observes only the main effects that they have on the objective function. However, even if interaction effects do exist, the polynomial approximation should still provide an adequate method of jumping as it still is bound by points that are on the actual objective function curve.

Another noteworthy potential research area is the integration of this continuous random variable technique with the well-defined methods of handling integer or Boolean variables. These types of problems have been studied extensively and therefore were ignored for the purposes of this paper. However, a user could easily create a problem that has both continuous variables and Boolean/integer variables. Moving forward an appropriate method of combining these techniques will need to be determined so that this method can be used to handle optimization problems of all structures.

Lastly, the effectiveness of this algorithm was limited by a lack of programming experience in implementing the algorithm. For this research, VBA was

used as a platform of development. However, inherent limitations of the VBA language cause unnecessary hurdles to overcome when coding this type of problem as well as having a noticeable lack in ability to handle matrices and perform matrix operations. A user wishing to implement this algorithm may have improved results by using a coding language that is more appropriate for mathematical search and manipulation.

10. References

Chelouah, R. and P. Siarry (2000). "Tabu search applied to global optimization." European Journal of Operational Research **123**(2): 256-270.

Gendreau, M. and J.-Y. Potvin (2010). Handbook of metaheuristics, Springer.

Glover, F. and M. Laguna (1999). Tabu search, Springer.

Hu, N. (1992). "Tabu search method with random moves for globally optimal design." International Journal for Numerical Methods in Engineering **35**(5): 1055-1070.

Karimi, A., et al. (2010). "Continuous ant colony system and tabu search algorithms hybridized for global minimization of continuous multi-minima functions." Computational Optimization and Applications **45**(3): 639-661.

Kirkpatrick, S. and M. Vecchi (1983). "Optimization by simmulated annealing." science **220**(4598): 671-680.

Siarry, P. and G. Berthiau (1997). "Fitting of tabu search to optimize functions of continuous variables." International Journal for Numerical Methods in Engineering **40**(13): 2449-2457.

Yang, X.-S. (2011). "Review of meta-heuristics and generalised evolutionary walk algorithm." International Journal of Bio-Inspired Computation **3**(2): 77-84.