MIXED INTEGER-DISCRETE-CONTINUOUS OPTIMIZATION BY DIFFERENTIAL EVOLUTION

Part 1: the optimization method

Jouni Lampinen

University of Vaasa
Department of Information Technology
and Production Economics
P. O. Box 700, FIN-65101 Vaasa, Finland
phone: +358-6-3248 111, fax: +358-6-3248 467
E-mail: Jouni.Lampinen@UWasa.fi

Ivan Zelinka

Technical University of Brno
Faculty of Technology (Zlín)
Department of Automatic Control
nám. T.G.M. 275, 762 72 Zlín, Czech Republic
phone: +420-67-721 15 21, fax: +420-67-721 15 21
E-mail: Zelinka@zlin.vutbr.cz

Abstract: *This article discusses solving non-linear programming problems containing integer, discrete and continuous variables. The Part 1 of the article describes a novel optimization method based on Differential Evolution algorithm. The required handling techniques for integer, discrete and continuous variables are described including the techniques needed to handle boundary constraints as well as those needed to simultaneously deal with several non-linear and non-trivial constraint functions. In Part 2 of the article a mechanical engineering design related numerical example, design of a coil spring, is given to illustrate the capabilities and the practical use of the method. It is demonstrated that the described approach is capable of obtaining high quality solutions. The novel method is relatively easy to implement and use, effective, efficient and robust, which makes it as an attractive and widely applicable approach for solving practical engineering design problems.*

Keywords: evolutionary algorithms, differential evolution, non-linear optimization, engineering design

1 Introduction

In general, when discussing non-linear programming, the variables of the object function are usually assumed to be continuous. However, in practical engineering design work the problems in which some or all of the design variables have discrete or integer values are very common. They are commonly discrete because the available values are limited to a set of standard sizes. For example, the thickness of a steel plate, the diameter of a copper pipe, the size of a screw, the module (or diametral pitch) of a gear tooth, the size of a roller bearing, etc., are often limited to a set of commercially available standard sizes. Respectively integer variables are commonly used to express a number of identical elements used in the design. Examples of integer variables are the number of teeth of a gear, the number of bolts or rivets needed to fix a structure, the number of heat exchanger tubes, the number of cooling fins of a heat sink, the number of parallel V-belts used for transmission, the number of coils of a spring, etc.

It is clear that a large fraction of engineering design optimization problems fall into the category of *mixed integer-discrete-continuous, non-linear programming* problems. Despite of that, mostly solving continuous problems are discussed in the literature. In practice, the problems containing integer or discrete variables are usually solved as a continuous problem, and the nearest available discrete value is then selected. In this case the result is often quite far from optimum. The reason for this approach is, that it is still commonly considered that no really satisfactory non-linear programming method appear to be available, which is able to handle all integer, discrete and continuous variables and which is also efficient, effective, robust and easy to use. Generally, continuous problems are considered to be easier to solve than discrete ones, suggesting that the presence of any non-continuous variables considerably increases the difficulty of finding a solution.

Another source of difficulties in practical engineering design optimization is handling of constraints. Often multiple constraints are involved, and often the nature of the constraints is non-linear and non-trivial, too. The feasible solutions may be only a small subset of the design space.

Many different methods have been proposed for solving mixed integer-discrete-continuous, non-linear programming problems. Some engineering design related works were collected in Table 1. Recently, many soft-computing based methods have been under investigations, and were found to be highly potential. However, no single approach seems to be totally satisfying when all the involved aspects are considered. Without pointing to any single method, it can be stated, that all of them suffer from at least one of the following shortcomings: complexity of implementation and usage, lack of flexibility, high computational cost, poor robustness, limited constraint handling capabilities, poor capabilities in finding feasible high quality solutions.

In order to response these demands of practical engineering design optimization, a novel approach for solving mixed integer-discrete-continuous, non-linear engineering design optimization problems has been developed based on the recently introduced *Differential Evolution* (DE) algorithm [SP95a]. This investigation describes the techniques needed to handle boundary constraints as well as those needed to simultaneously deal with several

non-linear and non-trivial constraint functions. After introducing these techniques, an illustrative and practical numerical example is given in Part 2 of this article, in which the design of a coil spring is optimized by DE. The mixed-variable methods used to solve the example problem are discussed in detail and compared with published results obtained with other optimization methods for the same problem.

Mixed Integer-Discrete-Continuous, Non-Linear Optimization		
Reported by	*Proposed solution technique*	*Reference*
Sandgren	Branch & Bound using Sequential Quadratic programming	[San90]
Fu, Fenton & Gleghorn	Integer-Discrete-Continuous Non-Linear Programming	[FFG91]
Loh & Papalambros	Sequential Linearization Algorithm	[LP91a, LP91b]
Zhang & Wang	Simulated Annealing	[ZW93]
Chen & Tsao	Genetic Algorithm	[CT93]
Li & Chow	Non-Linear Mixed-discrete Programming	[LC94]
Wu & Chow	Meta-Genetic Algorithm	[WC95]
Lin, Zhang & Wang	Modified Genetic Algorithm	[LZW95]
Thierauf & Cai	Two-level parallel Evolution Strategy	[TC97]
Cao & Wu	Evolutionary Programming	[CW97]
Lampinen & Zelinka	Differential Evolution	This article

Table 1: *Some proposed methods for solving engineering design related mixed integer-discrete-continuous, non-linear optimization problems.*

2 Mixed Integer-Discrete-Continuous Non-Linear Programming

A mixed integer-discrete-continuous, non-linear programming problem can be expressed as follows:

(1)

$$\begin{aligned}
&\text{Find} \\
&\mathrm{X} = \{\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \ldots, \mathrm{x}_n\} = \left[X^{(i)}, X^{(d)}, X^{(c)}\right]^T \\
&\text{to minimize} \\
&f(X) \\
&\text{subject to constraints} \\
&g_j(X) \le 0 \qquad j = 1,\ldots,m \\
&\text{and subject to boundary constraints} \\
&x_i^{(L)} \le x_i \le x_i^{(U)} \quad i = 1,\ldots,n \\
&\text{where} \\
&X^{(i)} \in \mathbb{R}^i \;, \quad X^{(d)} \in \mathbb{R}^d \;, \quad X^{(c)} \in \mathbb{R}^c
\end{aligned}$$

$X^{(i)}$, $X^{(d)}$ and $X^{(c)}$ denote feasible subsets of integer, discrete and continuous variables respectively. The above formulation is general and basically the same for all types of variables. Only the structure of the design domain distinguishes one problem from another. However, it is worth to notice here the principal differences between integer and discrete variables. While both integer and discrete variables have a discrete nature, only discrete variables can assume floating-point values. In practice the discrete values of the feasible set are often unevenly spaced. These are the main reasons why integer and discrete variables requires different handling.

3 Differential Evolution

Price and Storn first introduced the Differential Evolution (DE) algorithm a few years ago [SP95a]. DE can be categorized into a class of *floating-point encoded, evolutionary optimization algorithms*. Currently, there are several variants of DE. The particular variant used throughout this investigation is the *DE/rand/1/bin* scheme. This scheme will be discussed here only briefly, since more detailed descriptions are provided in [SP95a, SP97a]. Since the DE algorithm was originally designed to work with continuous variables, the optimization of continuous problems is discussed first. Handling of integer and discrete variables are to be explained later.

Generally, the function to be optimized, *f*, is of the form:

(2)

$$f(X):\ \mathbb{R}^n \rightarrow \mathbb{R}$$

The optimization target is to minimize the value of this *objective function f(X),*

$$\min\left(f(X)\right) \tag{3}$$

by optimizing the values of its parameters:

$$X = \left(x_1, \ldots, x_{n_{param}}\right) \qquad x \in \mathbb{R} \tag{4}$$

where X denotes a vector composed of n_{param} objective function parameters. Usually, the parameters of the objective function are also subject to lower and upper boundary constraints, $x^{(L)}$ and $x^{(U)}$, respectively:

$$x_j^{(L)} \le x_j \le x_j^{(U)} \qquad j = 1, \ldots, n_{param} \tag{5}$$

As with all evolutionary optimization algorithms, DE works with a population of solutions, not with a single solution for the optimization problem. Population P of generation G contains n_{pop} solution vectors called individuals of the population. Each vector represents potential solution for the optimization problem.

$$P^{(G)} = X_i^{(G)} \qquad i = 1, \ldots, n_{pop}\ ,\ G = 1, \ldots, G_{\max} \tag{6}$$

So, the population P of generation G contains n_{pop} individuals each containing n_{param} parameters (*chromosomes* of individuals):

$$P^{(G)} = X_i^{(G)} = x_{i,j}^{(G)} \qquad i = 1, \ldots, n_{pop}\ ,\ j = 1, \ldots, n_{param} \tag{7}$$

In order to establish a starting point for optimum seeking, the population must be initialized. Often there is no more knowledge available about the location of a global optimum than the boundaries of the problem variables. In this case, a natural way to initialize the population $P^{(0)}$ (initial population) is to seed it with random values within the given boundary constraints:

$$P^{(0)} = x_{i,j}^{(0)} = r_{i,j}(x_j^{(U)} - x_j^{(L)}) + x_j^L \qquad i = 1, \ldots, n_{pop} \quad , \quad j = 1, \ldots, n_{param} \tag{8}$$

where r denotes to a uniformly distributed random value within range [0.0,1.0[.

The population reproduction scheme of DE is different from the other evolutionary algorithms. From the 1st generation forward, the population of the following generation $P^{(G+1)}$ is created in the following way on basis of the current population $P^{(G)}$. First a temporary (trial) population for the subsequent generation, $P'^{(G+1)}$, is generated as follows:

$$x_{i,j}'^{(G+1)} = \begin{cases} x_{C_i,j}^{(G)} + F \cdot \left(x_{A_i,j}^{(G)} - x_{B_i,j}^{(G)}\right) & \text{if} \quad r_{i,j} \le C_r \vee j = D_i \\ x_{i,j}^{(G)} & \text{otherwise} \end{cases} \tag{9}$$

where,

$i = 1, \ldots, n_{pop}\ ,\ j = 1, \ldots, n_{param}$

$D = 1, \ldots, n_{param}$

$A = 1, \ldots, n_{pop}\ ,\ B = 1, \ldots, n_{pop}\ ,\ C = 1, \ldots, n_{pop}\ ,\ A_i \ne B_i \ne C_i \ne i$

$C_r \in [0,1]\ ,\ F \in [0,2]\ ,\ r \in [0,1[$

A, *B* and *C* are three randomly chosen indexes referring to three randomly chosen individuals of population. They are mutually different from each other and also different from the running index *i*. New, random, values for *A*, *B* and *C* are assigned for each value of index *i* (for each individual). A new value for random number *r* is assigned for each value of index *j* (for each chromosome).

The index *D* refers to a randomly chosen chromosome and it is used to ensure that at least one chromosome of each individual vector $X'^{(G+1)}$ differs from its counterpart in the previous generation $X^{(G)}$. A new random (integer) value is assigned to *D* for each value of index *i* (for each individual).

F and C_r are DE control parameters. Both values remain constant during the search process. As well the third control parameter, n_{pop} (population size), remain constant, too. *F* is a real-valued factor in range [0.0,2.0] that controls the amplification of differential variations and C_r is a real-valued crossover factor in range [0.0,1.0] controlling the probability to choose mutated value for *x* instead of its current value. Generally, both *F* and C_r

affect the convergence velocity and robustness of the search process. Their optimal values are dependent both on objective function, *f(X)*, characteristics and on the population size n_{pop}. Usually, suitable values for F, C_r and n_{pop} can be found by trial-and-error after a few tests using different values. Practical advice on how to select control parameters n_{pop} , F and C_r can be found in [SP95a, Sto96a, SP97a].

The selection scheme of DE also differs from the other evolutionary algorithms. On basis of the current population $P^{(G)}$ and the temporary population $P'^{(G+1)}$, the population of the next generation $P^{(G+1)}$ is created as follows:

(10)

$$X_i^{(G+1)} = \begin{cases} X'^{(G+1)}_i & \text{if} \quad f_{cost}\left(X'^{(G+1)}_i\right) \le f_{cost}\left(X_i^{(G)}\right) \\ X_i^{(G)} & \text{otherwise} \end{cases}$$

Thus, each individual of the temporary (trial) population is compared with its counterpart in the current population. The one with the lower value of cost function $f_{cost}(X)$ (to be minimized) will survive to the population of the next generation. As a result, all the individuals of the next generation are as good or better than their counterparts in the current generation. The interesting point concerning DE's selection scheme is that a trial vector is not compared against all the individuals in the current population, but only against one individual, against its counterpart in the current population.

4 Constraint Handling

4.1 Boundary constraints

It is important to notice that the reproduction operation of DE is able to extend the search outside of the initialized range of the search space (Equation 8 and 9). It is also worthwhile to notice that sometimes this is beneficial property in problems with no boundary constraints because it is possible to find the optimum that is located outside of the initialized range. However, in boundary constrained problems, it is essential to ensure that parameter values lie inside their allowed ranges after reproduction. A simple way to guarantee this is to replace parameter values that violate boundary constraints with random values generated within the feasible range:

(11)

$$x'^{(G+1)}_{i,j} = \begin{cases} r_{i,j}(x_j^{(U)} - x_j^{(L)}) + x_j^{(L)} & \text{if} \quad x'^{(G+1)}_{i,j} < x_j^{(L)} \vee x'^{(G+1)}_{i,j} > x_j^{(U)} \\ x'^{(G+1)}_{i,j} & \text{otherwise} \end{cases}$$

where,

$i = 1, \ldots, n_{pop}$, $j = 1, \ldots, n_{param}$

This is the method that was used for this work. Another simple method is to reproduce the boundary constraint violating values according Equation 9 as many times as is necessary to satisfy the boundary constraints.

4.2 Constraint functions

In this investigation, a soft-constraint (penalty) approach was applied for handling of the constraint functions. The constraint function introduces a distance measure from the feasible region, but is not used to reject unfeasible solutions, as it is in the case of hard-constraints. One possible soft-constraint approach is to formulate the cost-function as follows:

(12)

$$f_{cost}(X) = \left(f(X) + a\right) \cdot \prod_{i=1}^{m} c_i^{b_i}$$

where

$$c_i = \begin{cases} 1.0 + s_i \cdot g_i(X) & \text{if} \quad g_i(X) > 0 \\ 1 & \text{otherwise} \end{cases}$$

$s_i \ge 1$

$b_i \ge 1$

$\min\left(f(X)\right) + a > 0$

The constant, *a*, is used to ensure that only non-negative values will be assigned to f_{cost}. When the value of *a* is set high enough, it does not otherwise affect the search process. Constant, *s*, is used for appropriate scaling of the constraint function value. The exponent, *b*, modifies the shape of the optimization surface. Generally, higher values of *s* and *b* are used when the range of the constraint function, *g(X)*, is expected to be low. Often setting

s=1 and *b=1* works satisfactorily and only if one of the constraint functions, $g_i(X)$, remains violated after the optimization run, it will be necessary to use higher values for s_i or/and b_i.

In many real-world engineering design optimization problems, the number of constraint functions is relatively high and the constraints are often non-trivial. It is possible that the feasible solutions are only a small subset of the search space. Feasible solutions may also be divided into separated islands around the search space. Furthermore, the user may easily define totally conflicting constraints so that no feasible solutions exist at all. For example, if two or more constraints conflict, so that no feasible solution exists, DE is still able to find the nearest feasible solution. In the case of non-trivial constraints, the user is often able to judge which of the constraints are conflicting on the basis of the nearest feasible solution. It is then possible to reformulate the cost-function or reconsider the problem setting itself to resolve the conflict.

Another benefit of the soft-constraint approach is that the search space remains continuous. Multiple hard constraints often split the search space into many separated islands of feasible solutions. This discontinuity introduces stalling points for some genetic searches and also raises the possibility of new, locally optimal areas near the island borders. For these reasons a soft-constraint approach is considered essential. It should be mentioned that many traditional optimization methods are only able to handle hard-constraints. For evolutionary optimization, the soft-constraint approach was found to be a natural approach.

5 Handling of Integer and Discrete Variables

In its canonical form, the Differential Evolution algorithm is only capable of handling continuous variables. Extending it for optimization of integer variables, however, is rather easy. Only a couple of simple modifications are required. First, for evaluation of cost-function, integer values should be used. Despite of that the DE itself may still work internally with continuous floating-point values. Thus,

(13)

$$f_{\text{cost}}(y_i) \qquad i = 1,\ldots,n_{param}$$

where

$$y_i = \begin{cases} x_i & \text{for continuous variables} \\ INT(x_i) & \text{for integer variables} \end{cases}$$

$$x_i \in X$$

INT() is a function for converting a real value to an integer value by truncation. Truncation is performed here only for purposes of cost-function value evaluation. Truncated values are not elsewhere assigned. Thus, DE works with a population of continuous variables regardless of the corresponding object variable type. This is essential for maintaining the diversity of the population and the robustness of the algorithm. Second, in case of integer variable, instead of Equation 8, the population should be initialized as follows:

(14)

$$P^{(0)} = x_{i,j}^{(0)} = r_{i,j}(x_j^{(U)} - x_j^{(L)} + 1) + x_j^{(L)} \qquad i = 1,\ldots,n_{pop} \quad , \quad j = 1,\ldots,n_{param}$$

Additionally, instead of Equation 11, the boundary constraint handling for integer variables should be performed as follows:

(15)

$$x'^{(G+1)}_{i,j} = \begin{cases} r_{i,j}(x_j^{(U)} - x_j^{(L)} + 1) + x_j^{(L)} & \text{if} \quad INT\left(x'^{(G+1)}_{i,j}\right) < x_j^{(L)} \quad \vee \quad INT\left(x'^{(G+1)}_{i,j}\right) > x_j^{(U)} \\ x'^{(G+1)}_{i,j} & \text{otherwise} \end{cases}$$

where,

$$i = 1,\ldots,n_{pop} \quad , \quad j = 1,\ldots,n_{param}$$

Discrete values can also be handled in a straightforward manner. Suppose that the subset of discrete variables, $X^{(d)}$, contains l elements that can be assigned to variable x:

(16)

$$X^{(d)} = x_i^{(d)} \qquad i = 1,\ldots,l$$

where

$$x_i^{(d)} < x_{i+1}^{(d)}$$

Instead of the discrete value x_i itself, we may assign its index, i, to x. Now the discrete variable can be handled as an integer variable that is boundary constrained to range 1,…,l. To evaluate of the objective function, the discrete value, x_i , is used instead of its index i. In other words, instead of optimizing the value of the discrete variable directly, we optimize the value of its index i. Only during evaluation is the indicated discrete value used. Once

the discrete problem has been converted into an integer one, the previously described methods for handling integer variables can be applied (Equations 13–15).

6. Conclusions

The described method is relatively simple, easy to implement and easy to use. Despite of that, it is capable of optimizing all integer, discrete and continuous variables and capable of handling non-linear objective functions with multiple non-trivial constraints.

A soft-constraint (penalty) approach is applied for handling of constraint functions. Some optimization methods require a feasible initial solution as a starting point for a search. Preferably, this solution should be rather close to a global optimum to ensure convergence to it instead of a local optimum. If non-trivial constraints are imposed, it may be difficult or impossible to provide a feasible initial solution. The efficiency, effectiveness and robustness of many methods are often highly dependent on the quality of the starting point. The combination of DE with the soft-constraint approach does not require any initial solution, but it can still take advantage of a high quality initial solution if one is available. For example, this initial solution can be used for initialization of the population in order to establish an initial population that is biased towards the feasible region of the search space.

If there are no feasible solutions in the search space, as is the case for totally conflicting constraints, DE with the soft-constraint approach is still able to find the nearest feasible solution. This is important in practical engineering design work because often many non-trivial constraints are involved.

The described approach was targeted to fill the gap in the field of mixed discrete-integer-continuous optimization, where no really satisfactory methods appeared to be available. The Part 2 of this article will demonstrate, with a practical example, that the attempts were at least fairly successful. Despite being in its infancy, the described approach have great potential to become a widely used, multipurpose optimization tool for solving a broad range of practical engineering design problems.

References

[CW97] Cao, Y. J. and Wu, Q. H. (1997). *Mechanical design optimization by mixed-variable evolutionary programming*. Proceedings of the *1997 IEEE Conference on Evolutionary Computation*, pp. 443–446.

[CT93] Chen, J. L. and Tsao, Y. C. (1993). *Optimal design of machine elements using genetic algorithms.* Journal of the Chinese Society of Mechanical Engineers, 14(2):193–199, 1993.

[FFG91] Fu, J.-F., Fenton, R. G. and Cleghorn, W. L. (1991). *A mixed integer-discrete-continuous programming method and its application to engineering design optimization.* Engineering Optimization, 17(4):263–280, 1991. ISSN 0305-2154

[LC94] Li, H.-L. and Chou, C.-T. (1994). *A global approach for nonlinear mixed discrete programming in design optimization.* Engineering Optimization, 22():109–122, 1994.

[LZW95] Lin, Shui-Shun, Zhang, Chun and Wang, Hsu-Pin (1993). *On mixed-discrete nonlinear optimization problems: A comparative study.* Engineering Optimization, 23(4):287–300, 1995. ISSN 0305-215X

[LP91a] Loh, Han Tong and Papalambros, P. Y. (1991). *A sequential linearization approach for solving mixed-discrete nonlinear design optimization problems.* Transactions of the ASME, Journal of Mechanical Design, 113(3):325–334, September 1991.

[LP91b] Loh, Han Tong and Papalambros, P. Y. (1991). *Computational implementation and tests of a sequential linearization algorithm for mixed-discrete nonlinear design optimization.* Transactions of the ASME, Journal of Mechanical Design, 113(3):335–345, September 1991.

[Sto96a] Storn, Rainer (1996). *On the usage of differential evolution for function optimization*. NAFIPS 1996, Berkeley, pp. 519 - 523.

[SP95a] Storn, Rainer and Price, Kenneth (1995). *Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces*. Technical Report TR-95-012, ICSI, March 1995. (Available via `ftp` from `ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012.ps.Z`)

[SP97a] Storn, Rainer and Price, Kenneth (1997). *Differential Evolution – A simple evolution strategy for fast optimization*. Dr. Dobb's Journal, April 97, pp. 18–24 and p. 78.

[San90] Sandgren, E. (1990). *Nonlinear integer and discrete programming in mechanical design optimization.* Transactions of the ASME, Journal of Mechanical Design, 112(2):223–229, June 1990. ISSN 0738-0666

[TC97] Thierauf, G. and Cai, J. (1997*). Evolution strategies – parallelisation and application in engineering optimization*. In B.H.V. Topping (ed.) (1997). *Parallel and distributed processing for computational mechanics.* Saxe-Coburg Publications, Edinburgh (Scotland). ISBN 1-874672-03-2

[WC95] Wu, S.-J. and Chow, P.-T. (1995). *Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization.* Engineering Optimization, 24(2):137–159, 1995. ISSN 0305-215X

[ZW93] Zhang, Chun and Wang, Hsu-Pin (1993). *Mixed-discrete nonlinear optimization with simulated annealing.* Engineering Optimization, 21(4):277–291, 1993. ISSN 0305-215X

Reference data for this document:

[LZ99b] Jouni Lampinen – Ivan Zelinka (1999). *Mixed Integer-Discrete-Continuous Optimization By Differential Evolution, Part 1: the optimization method.* In: Ošmera, Pavel (ed.) (1999). Proceedings of *MENDEL'99, 5th International Mendel Conference on Soft Computing, June 9.–12. 1999, Brno, Czech Republic*. Brno University of Technology, Faculty of Mechanical Engineering, Institute of Automation and Computer Science, Brno (Czech Republic), pp. 71–76. ISBN 80-214-1131-7.