Discrete search in design optimization

Martin Fuchs and Arnold Neumaier

Abstract It is a common feature of many real-life design optimization problems that some design components can only be selected from a finite set of choices. Each choice corresponds to a possibly multidimensional design point representing the specifications of the chosen design component. In this paper we present a method to explore the resulting discrete search space for design optimization. We use the knowledge about the discrete space represented by its minimum spanning tree and find a splitting based on convex relaxation.

1 Introduction

In design optimization the typical goal is to find a design with minimal cost while satisfying functionality constraints. In some cases there are further objectives relevant to assess the quality of the design, thus leading to multicriteria optimization. We focus on the case that the objective is 1-dimensional and we look for the optimal design $\theta \in \mathbf{T} \subset \mathbb{R}^{n_0}$ where the components of $\mathbf{T} = T^1 \times \cdots \times T^{n_0}$ are intervals of **continuous** variables (e.g., the thickness of a car's body) or **integer** variables (e.g., the choice between different motor types). The integer variables arise from reformulation of discrete multidimensional sets in terms of subsets of $\mathbb{Z}$. That means the original problem contains constraints like $z \in \mathscr{Z} := \{z_1, \ldots, z_N\}$, $z_k \in \mathbb{R}^n$, where z_i contains the specifications of the choice i, e.g., mass, performance, and cost of different motors. The constraint $z \in \mathscr{Z}$ can be reshaped to an integer formulation of the search space by using N binary variables $b_1, \ldots, b_N$, $b_i \in \{0,1\}$ and the constraints $z = \sum_{i=1}^N b_i z_i$ and $\sum_{i=1}^N b_i = 1$.

Martin Fuchs
CERFACS, 31057 Toulouse, France, e-mail: martin.fuchs81@gmail.com

Arnold Neumaier
University of Vienna, Faculty of Mathematics, 1090 Wien, Austria,
e-mail: arnold.neumaier@univie.ac.at

Depending on the objective function and the constraints design optimization belongs to one of the following problem classes of **mixed integer programming**: both objective function and constraints are all linear, i.e., mixed integer linear programming (MILP); at least one constraint or the objective function is nonlinear, i.e., mixed integer nonlinear programming (MINLP); at least one constraint or the objective function is given as a black box, i.e., black box optimization. For all these types of problems there exist algorithms that employ a **splitting** of the search space $\mathbf{T}$ as a crucial part of their solution technique.

A splitting technique finds a subdivision of the original problem in two or more subproblems such that the associated optimization algorithm can decide how to proceed solving these subproblems which may possibly include further splitting. The subdivision must ensure that the optimal solution of the original problem can be found as one of the solutions of the subproblems.

For a study of efficient methods using splitting in branch and bound for MILP see, e.g., [3, 11]. Additionally, methods for MINLP also employing branch and bound can be found, e.g., in [10, 14]. Branching in black box problems with continuous variables only is studied, e.g., in [7, 9]. Branching rules in mixed integer programming (mainly MILP) are presented, e.g., in [1]. For a survey on discrete optimization, including branch and bound, see [13]. The special class of design optimization problems can be studied, e.g., in [2, 4, 12].

If the design optimization problem is formulated using integer choice variables, it can be tackled heuristically without branching, e.g., by separable underestimation [5]. An optimization algorithm using a branching method on the integers does not exploit the knowledge about the structure of $\mathscr{Z}$. Using this knowledge, however, may have significant advantages since the constraints that model the functional relationships between different components of the design depend on the values of $z \in \mathscr{Z}$ rather than on the values of the integer choices.

In this paper we present a splitting strategy for discrete search spaces such as $\mathscr{Z}$ described above. We use only local information about functional constraints. Thus the method is applicable in all problem classes of mixed integer programming, even for black box optimization, which frequently occurs in design optimization.

We use the knowledge about the structure of the space $\mathscr{Z}$ represented by its minimum spanning tree [15]. We represent a solution of an auxiliary optimization problem as a convex combination of the points $z_1, \ldots, z_N$, and use the coefficients of the combination to determine a splitting across an edge of the minimum spanning tree of $\mathscr{Z}$, cf. [6]. An implementation of our method in MATLAB can be found at `www.martin-fuchs.net/downloads.php`.

This paper is organized as follows. In Section 2 we introduce the design optimization problem formulation and notation. In Section 3 we describe how to determine a splitting based on convex relaxation of the discrete constraints. A simple solver strategy that employs our splitting technique is sketched in Section 4. Results for a real-life example in 10 dimensions are given in Section 5.

2 Design optimization

Let $F : \mathbb{R}^{n_z} \to \mathbb{R}$ be a scalar objective function which typically is a **black box** model, e.g., for the cost of the design, containing the functional relationships between different design components. We assume that the design optimization problem is formulated in the following form:

$$\left.\begin{array}{ll} \min\limits_{\theta,z} & F(z) \\ \text{s.t.} & z = Z(\theta), \\ & \theta \in \mathbf{T}, \end{array}\right\} \tag{1}$$

where $Z : \mathbb{R}^{n_0} \to \mathbb{R}^{n_z}$, $\theta = (\theta^1, \theta^2, \ldots, \theta^{n_0})^T$ is a vector of **choice variables**, ^{T} means transposed, and $\mathbf{T}$ is the domain of possible choices of θ. We assume that the choices can be discrete or continuous. By I_d we denote the index set of choice variables that are discrete, and we denote the index set of choice variables that are continuous by I_c. We have $I_d \cup I_c = \{1, 2, \ldots, n_0\}$, $I_d \cap I_c = \emptyset$. The **selection constraints** $\theta \in \mathbf{T}$ specify which choices are allowed for each choice variable, i.e., $\mathbf{T} = T^1 \times \cdots \times T^{n_0}$, $T^i = \{1, 2, \ldots, N_i\}$ for $i \in I_d$, and $T^i = [\underline{\theta^i}, \overline{\theta^i}]$ for $i \in I_c$. The mapping Z assigns an input vector z to a given choice vector θ.

In the **discrete** case, $i \in I_d$, a choice variable θ^i determines the value of n_i components of the vector $z \in \mathbb{R}^{n_z}$ which is the input for the model F. Let $1, 2, \ldots, N_i$ be the possible choices for θ^i, $i \in I_d$, then the discrete choice variable θ^i corresponds to a finite set of N_i points in $\mathbb{R}^{n_i}$. Usually this set is provided in an $n_i \times N_i$ table τ^i, i.e., $Z^i(\theta^i)$ is the θ^ith column of τ^i (see, e.g., Table 1). The mapping Z^i, $i \in I_d$, can be regarded as a reformulation of a multidimensional discrete sub search space consisting of $Z^i(1), \ldots, Z^i(N_i)$ into the integer choices $1, \ldots, N_i$. In the **continuous** case, $i \in I_c$, the choice variable θ^i belongs to an interval $[\underline{\theta^i}, \overline{\theta^i}]$, i.e.,

$$Z^i(\theta^i) := \theta^i \text{ for } \theta^i \in [\underline{\theta^i}, \overline{\theta^i}]. \tag{2}$$

Via the concatenation of $Z^i(\theta^i)$ we now define

$$Z(\theta) := z := (Z^1(\theta^1), \ldots, Z^{n_0}(\theta^{n_0})). \tag{3}$$

Note that any vector $z = Z(\theta)$ has the length $n_z = \sum_{i \in I_d} n_i + |I_c|$. We call Z a table mapping as the nontrivial parts of Z consist of the tables τ^i.

Toy example. Let $\mathbf{T} = T^1 \times T^2 \times T^3$ be choices for the design of a car, let $T^1 = \{1, \ldots, 7\}$ be seven choices for the motor of the car, let $T^2 = [0, 2]$ be the continuous choice of the thickness of the car's body, let $T^3 = \{1, \ldots, 10\}$ be ten choices for the length of the car's axes that are only available in integer units. Thus we have $I_d = \{1, 3\}$, $I_c = \{2\}$. Let the associated table τ^1 be as shown in Table 1 with $N_1 = 7$, $n_1 = 2$, describing two-dimensional characteristics of each motor. The table τ^3 is simply given by $\tau^3 = (1, 2, \ldots, 10)^T$. Let the objective function F be given by

$$F(z) = F(v_1^1, v_2^1, v_1^2, v_1^3) = \left(v_1^1 + \frac{1}{2}\right)^2 + \left(v_2^1 - \frac{3}{4}\right)^2 + \exp(v_1^2) + (v_1^3 - 10)^2. \quad (4)$$

modeling the cost of the designed car, where $v^i := Z^i(\theta^i)$. The optimal solution of (1) with these specifications is given as $\theta = (5,0,10)$ since $Z^1(5) = (-4,0)^T$ is closest to$(-\frac{1}{2}, \frac{3}{4})^T$ and the two choices θ^2 and θ^3 are obvious.

Note that the objective in fact depends on $Z(\mathbf{T})$, and not on the integer values in T^1 or T^3. This is the crucial message of this example: in design optimization discrete choices are typically associated with multidimensional discrete spaces. Subdividing the search space should thus subdivide $Z(\mathbf{T})$ instead of $\mathbf{T}$. The rest of the example is constructed to be sufficiently simple to illustrate the method presented in the next sections.

Table 1 Tabulated data $Z^1(T^1)$

θ^1	1	2	3	4	5	6	7
v_1^1	4	4	6	5	-4	-8	-4
v_2^1	0	1	0	3	0	0	2

The method that we use to solve (1) is a splitting strategy based on convex relaxation of the discrete search spaces, cf. [6]. We solve (1) as a special case of the following optimization problem.

$$\left.\begin{aligned} &\min_{\theta,x} \; c^T x \\ &\text{s.t. } \; F(Z(\theta)) \le Ax, \\ &\qquad \theta \in \mathbf{T} := T^1 \times \cdots \times T^{n_0}, \end{aligned}\right\} \quad (5)$$

where $F : \mathbb{R}^{n_z} \to \mathbb{R}^{m_F}$, possibly $m_F > 1$, $c, x \in \mathbb{R}^{m_x}$, $A \in \mathbb{R}^{m_F \times m_x}$. With $c = A = 1$, and by eliminating x and introducing a new intermediate variable z in (5) we see that (1) is a special case of (5).

3 Convex relaxation based splitting strategy

The idea behind the method presented in this section is that a representation of a continuous relaxed solution $\widehat{z} = (\widehat{z}^1, \ldots, \widehat{z}^{n_z})$ of (5) by **convex combinations** of the points $Z^i(\theta^i)$, $\theta^i \in T^i$, $i \in I_d$, gives an insight about the relationship between the solution $\widehat{z}$ and the structure of the discrete search space in each component $i \in I_d$. The split divides this space into two branches, where the contribution of each branch to the relaxed solution of (5) is as balanced as possible. Thus we can exploit

the knowledge of the structure of $Z^i(T^i)$ – represented by its **minimum spanning tree** – towards finding a natural subdivision of $\mathbf{T}$. A convex combination for $\widehat{z} = (\widehat{v}^1, \ldots, \widehat{v}^{n_0})$ is given by

$$\widehat{v}^i = \sum_{\theta^i=1}^{N_i} \widehat{\lambda}^i_{\theta^i} Z^i(\theta^i), \text{ for } i \in I_d, \tag{6}$$

with $\sum_{j=1}^{N_i} \lambda^i_j = 1$, $\lambda^i_j \geq 0$, i.e., a convex combination of the finitely many tabulated vectors in $\mathbb{R}^{n_i}$ given in a table τ^i. We will see how the coefficients $\widehat{\lambda}^i = (\widehat{\lambda}^i_1, \ldots, \widehat{\lambda}^i_{N_i})$, $i \in I_d$, of the convex combination in the ith coordinate impose a splitting of the search space components T^i, $i \in I_d$.

To compute a convex relaxation of (5) we reformulate the problem as follows. Assume that we have an initial set of N_0 starting points $z_1, \ldots, z_{N_0}$ coming from a relaxation $Z^1_{\text{rel}} \times \cdots \times Z^{n_0}_{\text{rel}}$ of the discrete constraints. That means the discrete sets $Z^i(T^i) \subset \mathbb{R}^{n_i}$, $i \in I_d$, are relaxed to interval bounds $Z^i_{\text{rel}} = Z^i_{\text{rel},1} \times \cdots \times Z^i_{\text{rel},n_i}$, where $Z^i_{\text{rel},k} = [\ell, u]$ with $\ell = \min_{v^i \in Z^i(T^i)} v^i_k$, $u = \max_{v^i \in Z^i(T^i)} v^i_k$. Let $F_1 = F(z_1), \ldots, F_{N_0} = F(z_{N_0})$ be the function evaluations of the model F at the starting points which are used to approximate F. We solve the following problem:

$$\left.\begin{aligned}
\min_{z,x,\mu,v,\lambda} \quad & c^T x + \varepsilon \|\mu\|_p \\
\text{s.t.} \quad & \sum_{j=1}^{N_0} \mu_j F_j \leq Ax, \\
& z = \sum_{j=1}^{N_0} \mu_j z_j, \\
& \sum_{j=1}^{N_0} \mu_j = 1, \\
& z = (v^1, \ldots, v^{n_0}), \\
& v^i = \sum_{j=1}^{N_i} \lambda^i_j Z^i(j) \text{ for } i \in I_d, \\
& \sum_{j=1}^{N_i} \lambda^i_j = 1 \text{ for } i \in I_d, \\
& \lambda^i_j \geq 0 \text{ for } i \in I_d, 1 \leq j \leq N_i, \\
& v^i \in [\underline{\theta^i}, \overline{\theta^i}] \text{ for } i \in I_c.
\end{aligned}\right\} \tag{7}$$

Here we approximate F at the given evaluation points, i.e., $F(z) \approx \sum_{j=1}^{N_0} \mu_j F_j$, for $z = \sum_{j=1}^{N_0} \mu_j z_j$, $\sum_{j=1}^{N_0} \mu_j = 1$. We require the solution to be a convex combination of the tabulated points $Z^i(j)$ in the discrete case $i \in I_d$, i.e., $z = (v^1, \ldots, v^{n_0})$, $v^i =$

$\sum_{j=1}^{N_i} \lambda_j^i Z^i(j)$, $\sum_{j=1}^{N_i} \lambda_j^i = 1$, $\lambda_j^i \geq 0$, $1 \leq j \leq N_i$. And we require the solution to respect the bound constraints on the continuous choices, i.e., $v^i \in [\underline{\theta^i}, \overline{\theta^i}]$. The constant ε can be considered as a regularization parameter, adjusted externally. The objective function in (7) is convex, the constraints are linear.

Remark 1. In case of design optimization with 1-dimensional F the problem (7) is typically unbounded for $p = 1$ and low ε. After increasing ε it typically changes towards a binary solution $\mu_i = 1$ for some i, and $\mu_k = 0$ for $k \neq i$. A binary μ would simply mean that if there are starting points among the z_j in the Cartesian product of the convex hulls of $Z^i(T^i)$, then the solution $\widehat{z} =: \widehat{z}_{\text{start}}$ is the best of these points. Choosing $p = 2$ and increasing ε from 0 towards ∞ in numerical experiments, the solution of (7) typically changes from unbounded to a binary solution and then converges to $\mu = (\frac{1}{N_0}, \ldots, \frac{1}{N_0})$ which may produce an alternative solution $\widehat{z} \neq \widehat{z}_{\text{start}}$. Hence in case that $m_F = 1$ we solve (7) twice, with $p = 1$ and $p = 2$. Thus we get two solutions $\widehat{z}_1$ and $\widehat{z}_2$, respectively. Then we compare the two solutions by evaluating F, i.e., $\widehat{F}_1 := F(\widehat{z}_1)$ and $\widehat{F}_2 := F(\widehat{z}_2)$. If $\widehat{F}_2 < \widehat{F}_1$ we use in the remainder of our method the convex combination λ found by (7) with $p = 2$, otherwise we use the convex combination λ found by (7) with $p = 1$.

Remark 2. One could also approximate F nonlinearly in (7). Hence (7) becomes a nonlinear programming problem which requires a different formulation.

The solution of (7) gives the values of the coefficients $\widehat{\lambda}^i = (\widehat{\lambda}_1^i, \ldots, \widehat{\lambda}_{N_i}^i)$, $i \in I_d$, of the convex combinations for $\widehat{v}^i$. These values are now used to determine a **splitting** of T^i.

Consider the ith coordinate $\theta^i \in T^i = \{1, 2, \ldots, N_i\}$, $i \in I_d$. One computes the **minimum spanning tree** for the points $Z^i(T^i) \subset \mathbb{R}^{n_i}$, see, e.g., Fig. 1.

For a fixed edge k in the graph belonging to the minimum spanning tree of $Z^i(T^i)$ we denote by Z_{k1}^i the set of all points on the right side of k, and we denote by Z_{k2}^i the set of points on the left side of k (e.g., in Fig. 1 let k be the edge $(1\text{—}5)$, then $Z_{k1}^1 = \{1,2,3,4\}$, and $Z_{k2}^1 = \{5,6,7\}$).

For every edge k in the minimum spanning tree of $Z^i(T^i)$ one computes the weight w_{k1}^i of all points in Z_{k1}^i and the weight w_{k2}^i of all points in Z_{k2}^i by

$$w_{k1}^i = \sum_{\{j | Z^i(j) \in Z_{k1}^i\}} \widehat{\lambda}_j^i, \tag{8}$$

$$w_{k2}^i = \sum_{\{j | Z^i(j) \in Z_{k2}^i\}} \widehat{\lambda}_j^i. \tag{9}$$

We split T^i into T_1^i and T_2^i across the edge $\widehat{k} = \arg\min_k |w_{k1}^i - \frac{1}{2}|$, i.e., the edge where the weight on the one side and the weight on the other side are closest to 50%.

Remark 3. Interpreting the weight on one side as the contribution to the relaxed solution we thus split T^i as balanced as possible. Though a balanced splitting is not necessarily the best strategy in general, it is motivated quite naturally. If there is one

leaf of the minimum spanning tree close to the relaxed solution it has a weight close to 1, and we split off just this leaf, thus strongly reducing the search space. If the weights are rather uniform we do not have sufficiently precise information to split off just a small subset of the minimum spanning tree, so we split in a rather uniform way, i.e., in two partitions of similar weight.

Remark 4. In total we find up to $2^{|I_d|}$ possible branches. The decision on which variable to split – and whether or how to join the branches in an optimization algorithm in order to find a division of the search space after computing T_1^i, T_2^i – may seriously affect the performance of the algorithm and depends on how the algorithm handles the resulting subproblems. Section 4 presents one possibility of a branching strategy.

Remark 5. Using the minimum spanning tree is not scaling invariant as the results depend on distances between the discrete points $Z^i(T^i)$. Hence the user of the method should use a scaling of the variables where distances between the discrete points have a reasonable meaning.

A particular strength of our approach is that we do not require information about F except for the function evaluations $F_1, \ldots, F_{N_0}$, hence also black box functions F can be handled which is often occurring in real-life applications.

An implementation of the method can be found online at `www.martin-fuchs.net/downloads.php`.

Toy example. To solve our example problem we apply the method with $N_0 = 20$, $c = A = 1$, and $\varepsilon = 10^2$.

We look for splittings of T^i, $i \in I_d = \{1,3\}$. The graph of the minimum spanning tree of $Z^1(T^1)$ is shown in Fig. 1.

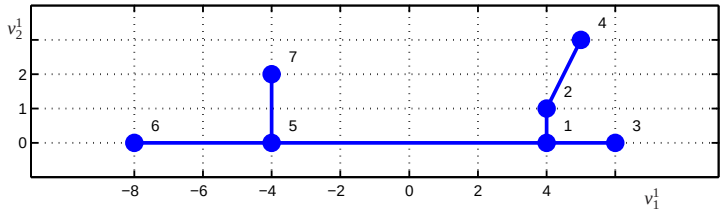

Fig. 1 Graph of the minimum spanning tree of $Z^1(T^1)$

Relaxing the discrete search space to a continuous space, the solution of (5) is obtained at $\widehat{z} = (-\frac{1}{2}, \frac{3}{4}, 0, 10)$.

The convex combination with fixed $\widehat{\lambda}^1 = (\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6})$ would give

$$\sum_{j=1}^{7} \widehat{\lambda}_j^1 Z^1(j) = (-0.29, 0.83), \tag{10}$$

which is close to $(\widehat{z}^1,\widehat{z}^2)$, so our method should find a weighting similar to $\widehat{\lambda}^1$. This weighting would apparently lead to a split of $Z^1(T^1)$ across the edge $k=(1\text{—}5)$ since $\sum_{j\in\{1,2,3,4\}}\widehat{\lambda}^1_j=\sum_{j\in\{5,6,7\}}\widehat{\lambda}^1_j=0.5$. Thus it is not surprising that our implemented method actually splits T^1 into $T^1_1=\{1,2,3,4\}$ and $T^1_2=\{5,6,7\}$ in most experiments. Sometimes it splits off the leaf 7, i.e., $T^1_1=\{1,2,3,4,5,6\}$ and $T^1_2=\{7\}$, or it finds $T^1_1=\{1,3,5,6,7\}$, $T^1_2=\{2,4\}$, depending on the approximation and the convex combination found from (7), determined by the function evaluations $F_1,\ldots,F_{N_0}$.

The split of T^3 is expected to split off the leaf 10 since F is strictly monotone decreasing in $v^3=z^4\in[1,10]$ resulting in a weight $\widehat{\lambda}^3_{10}$ close to 1. Except for few cases where we find $T^3_1=\{1,2,3,4,5,6,7,8\}$, $T^3_2=\{9,10\}$, we find the expected splitting of T^3 into $T^3_1=\{1,2,3,4,5,6,7,8,9\}$ and $T^3_2=\{10\}$ with our implemented method.

4 A simple solver

We have implemented the method in a simple solver with the following branching strategy: We round the relaxed solution $\widehat{z}$ of (7) to the next feasible point of (5) $\widehat{z}_{\text{round}}:=\arg\min_{\{z\in Z(\mathbf{T})\}}\|z-\widehat{z}\|_2$ and start from $\widehat{z}_{\text{round}}$ a local search in $\mathbf{T}$, i.e., an integer line search for the discrete choice variables, afterwards multilevel coordinate search (MCS) [7], for the continuous choice variables and an iteration of this procedure until satisfaction. The function evaluations during the local search are used in two ways.

First, we use them to determine the coordinate i of $\theta=(\theta^1,\theta^2,\ldots,\theta^{n_0})$ for which $\text{dev}_{\max}(i)$ is maximal, where $\text{dev}_{\max}(i)$ is the maximum deviation of the function values while varying θ^i in the local search. Then we split the original $\mathbf{T}=T^1\times\cdots\times T^{n_0}$ only in this coordinate and get two branches T_1, T_2, i.e., $T_1=T^1\times\cdots\times T^i_1\times\cdots\times T^{n_0}$, $T_2=T^1\times\cdots\times T^i_2\times\cdots\times T^{n_0}$, where T^i_1, T^i_2 are the results from our splitting method in Section 3.

Second, we select T_1 for the next iteration step if the best point found during local search comes from T_1, otherwise we select T_2 for the next iteration step. Having selected a branch for the next step we iterate branching and local search until satisfaction. As a stopping criterion one may choose, e.g., that the optimal solution found by the local search has not been improved for N_{iter} times, or a maximum total number of iterations.

This simple strategy is already suited to demonstrate the usefulness of our splitting routine in Section 5. As soon as our method has been implemented into an enhanced version of the solver we will also provide a comparison with further different solvers on more test cases.

Toy example. We have used the solver strategy described to find the optimum $\theta=(5,0,10)$ of our example problem. In the first iteration we split T^3 and find the branch $T_{\text{new}}=T^1\times T^2\times T^3_{\text{new}}$, $T^3_{\text{new}}=\{10\}$ for the next iteration step. In the

following iterations T^1 is reduced to $\{5,6,7\}$, then to $\{5,6\}$, and finally to $\{5\}$, and local search confirms the optimum $\theta = (5,0,10)$.

5 A real-life application

We have applied our method to a problem of optimization under uncertainty in spacecraft system design, described in the study [12], and we compare the results with the existing method used in that study. After reasonable simplification, the problem can be formulated as in (1), where $\theta \in \mathbb{R}^{10}$ is a 10-dimensional design point, $F(Z(\theta))$ is a MATLAB routine computing the worst case for the total mass of the spacecraft at the design point θ under all admissible uncertainties. That means one looks for the design with the minimal total mass, taking into account possible uncertainties.

In [12] heuristics based on SNOBFIT [8] was used which suggested a candidate for the optimal solution in each iteration step. From this candidate one performs a local search and iterates afterwards until no improvement of the optimal solution has been found 4 times in a row. This SNOBFIT based search is done 20 times independently with 20 different random starting ensembles to check the reliability of the putative global optimum found. However, SNOBFIT is not developed to deal with integers, so integer variables θ^i, $i \in I_d$ are treated as continuous variables and rounded to the next integer values. Hence the optimum candidates suggested by SNOBFIT are suboptimal, and the local search gives the most significant improvement towards the optimal solution. The global optimum was found in 3 out of 20 runs. On an average one run required about 2500 evaluations of F.

With the solver strategy described in Section 4 we can confirm the resulting global optimum. The reliability of our approach, however, is significantly better. In 5 independent runs we have found the optimum 4 times. One run failed because there was no feasible point in the set of initial function evaluations. One run also required about 2500 function evaluations on an average. Hence at the same level of reliability we have found the solution with much less effort.

References

1. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Operations Research Letters **33**(1), 42–54 (2005)
2. Alexandrov, N., Hussaini, M.: Multidisciplinary design optimization: State of the art. In: Proceedings of the ICASE/NASA Langley Workshop on Multidisciplinary Design Optimization. Hampton, Virginia, USA (1997)
3. Floudas, C.: Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications. Oxford University Press (1995)
4. Fuchs, M., Girimonte, D., Izzo, D., Neumaier, A.: Robust Intelligent Systems, chap. Robust and automated space system design, pp. 251–272. Springer (2008)

5. Fuchs, M., Neumaier, A.: Autonomous robust design optimization with potential clouds. International Journal of Reliability and Safety **3**(1/2/3), 23–34 (2009)
6. Fuchs, M., Neumaier, A.: A splitting technique for discrete search based on convex relaxation. Journal of Uncertain Systems, Special Issue on Global Optimization and Intelligent Algorithm (2009). Accepted, preprint available on-line at: `http://www.martin-fuchs.net/publications.php`
7. Huyer, W., Neumaier, A.: Global optimization by multilevel coordinate search. Journal of Global Optimization **14**(4), 331–355 (1999)
8. Huyer, W., Neumaier, A.: SNOBFIT – Stable Noisy Optimization by Branch and Fit. ACM Transactions on Mathematical Software **35**(2) (2008). Article 9, 25 pages
9. Jones, D., Perttunen, C., Stuckman, B.: Lipschitzian optimization without the Lipschitz constant. Journal of Optimization Theory and Applications **79**(1), 157–181 (1993)
10. Leyffer, S.: Deterministic methods for mixed integer nonlinear programming. Ph.D. thesis, University of Dundee, Department of Mathematics & Computer Science (1993)
11. Nemhauser, G., Wolsey, L.: Integer and combinatorial optimization. Wiley-Interscience (1988)
12. Neumaier, A., Fuchs, M., Dolejsi, E., Csendes, T., Dombi, J., Banhelyi, B., Gera, Z.: Application of clouds for modeling uncertainties in robust space system design. ACT Ariadna Research ACT-RPT-05-5201, European Space Agency (2007)
13. Parker, R., Rardin, R.: Discrete optimization. Academic Press (1988)
14. Tawarmalani, M., Sahinidis, N.: Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. Mathematical Programming **99**(3), 563–591 (2004)
15. Weisstein, E.: Minimum spanning tree. MathWorld – A Wolfram Web Resource (2008). Available on-line at: `http://mathworld.wolfram.com/MinimumSpanningTree.html`