Journal of Uncertain Systems
Vol.x, No.x, pp.xx-xx, 2009
Online at: www.jus.org.uk

A splitting technique for discrete search based on convex relaxation

Martin Fuchs[12*] Arnold Neumaier[2]
[1] *CERFACS, 31057 Toulouse, France*
[2] *University of Vienna, Faculty of Mathematics, 1090 Wien, Austria*
martin.fuchs81@gmail.com arnold.neumaier@univie.ac.at

Received 15 January 2009; Accepted 03 July 2009

Abstract

In mixed integer programming branching methods are a powerful and frequently employed tool. This paper presents a branching strategy for the case that the integer constraints are associated with a finite set of points in a possibly multidimensional space. We use the knowledge about this discrete set represented by its minimum spanning tree and find a splitting based on convex relaxation. Typical applications include design optimization problems where design points specifying several discrete choices can be considered as such discrete sets.

Keywords: branching strategies; mixed integer programming; convex relaxation; minimum spanning tree; design optimization

1 Introduction

In mixed integer programming problems one seeks an optimal solution among variables $\theta \in \mathbf{T} \subset \mathbb{R}^{n_0}$ where the components of $\mathbf{T} = T^1 \times \cdots \times T^{n_0}$ are intervals of continuous or integer variables. Depending on the objective function and the constraints one faces different 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 the splitting (also called branching) 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 an exhaustive survey on discrete optimization, including branch and bound, see [13].

In many problem formulations, integer variables arise from reformulation of discrete multidimensional sets in terms of subsets of $\mathbb{Z}$. That means the original problem contains constraints like $z \in \mathcal{Z} := \{z_1, \ldots, z_N\}$, $z_k \in \mathbb{R}^n$. This constraint can be reformulated to a mixed 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 $\sum_{i=1}^N b_i = 1$.

One real-life application, where this kind of discrete search spaces shows up frequently is design optimization, cf., e.g., [2, 4, 12]. In design optimization a design choice can be either a continuous variable, e.g., the diameter of a car tire, or an integer variable, e.g., the choice between different motor types, determining a multidimensional discrete space $\mathcal{Z}$ of design points (in our example the specifications like mass and performance of different motors). 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 $\mathcal{Z}$. Using this knowledge, however, may have significant advantages since functional constraints (i.e., constraints that

*Corresponding author. Email: martin.fuchs81@gmail.com , Web: www.martin-fuchs.net

model the functional relationships between different components of the design, often given as black boxes) typically depend on the values of $z \in \mathcal{Z}$ rather than on the values of the integer choices.

In this paper we present a branching strategy for discrete search spaces such as $\mathcal{Z}$ 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.

We use the knowledge about the structure of the space $\mathcal{Z}$ 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 $\mathcal{Z}$. 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 optimization problem formulation and notation. In Section 3 we discuss in detail how to determine a splitting based on convex relaxation of the discrete constraints. Section 4 summarizes the corresponding implementation in MATLAB. In Section 5 we describe a simple solver strategy that makes use of our splitting technique. An illustrative example in 3 dimensions shows how to use the method, and a more complicated real-life example in 10 dimensions is given in Section 6.

2 Problem statement

Assume that we are given a mixed-integer optimization problem with selection constraints of the following form:

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

where $c, x \in \mathbb{R}^m$, ^{T} means transposed, $A \in \mathbb{R}^{m \times m}$, $F : \mathbb{R}^{n_z} \to \mathbb{R}^m$, $Z : \mathbb{R}^{n_0} \to \mathbb{R}^{n_z}$, $\theta = (\theta^1, \theta^2, \ldots, \theta^{n_0})^T$ is a vector of **choice variables**, $\mathbf{T}$ the selection space for θ. We assume that the choices can be discrete or continuous. Let I_d be the index set of choice variables that are discrete and I_c be the index set of choice variables that are continuous, $I_d \cup I_c = \{1, 2, \ldots, n_0\}$, $I_d \cap I_c = \emptyset$. 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 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 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 (see, e.g., Table 1 with $N_i = 10$, $n_i = 3$).

Table 1: Tabulated design specifications (taken from [4]): Each row corresponds to the specifications of the thruster choice θ.

θ	**Thruster**	F/N	I_{sp}/s	m_{thrust}/kg
1	Aerojet MR-111C	0.27	210.0	200
2	EADS CHT 0.5	0.50	227.3	200
3	MBB Erno CHT 0.5	0.75	227.0	190
4	TRW MRE 0.1	0.80	216.0	500
5	Kaiser-Marquardt KMHS Model 10	1.0	226.0	330
6	EADS CHT 1	1.1	223.0	290
7	MBB Erno CHT 2.0	2.0	227.0	200
8	EADS CHT 2	2.0	227.0	200
9	EADS S4	4.0	284.9	290
10	Kaiser-Marquardt KMHS Model 17	4.5	230.0	380

That means $Z^i(\theta^i)$ is the $\theta^{i^{\text{th}}}$ row of τ^i. The mapping Z^i, $i \in I_d$, can be regarded as a reformulation of a multivariate 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.,

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

We 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.

Remark 1: The problem statement (1) is typical in design optimization where the choice θ^i may be the choice of a design component (as mentioned in Section 1 and shown in Table 1), and F models the functional relationship between different components. In this case F is typically 1-dimensional and $c = 1$, $A = 1$. Thus we can reformulate (1) as

$$\left.\begin{array}{ll} \min\limits_{\theta,z} & F(z) \\ \text{s.t.} & z = Z(\theta), \\ & \theta \in \mathbf{T}, \end{array}\right\} \tag{4}$$

by eliminating x and introducing a new intermediate variable z.

3 Convex relaxation based splitting

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 (1) 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 difference between the contribution of each branch to the relaxed solution of (1) is minimal (also see Section 6.1 for an illustrative example). 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

$$\widehat{v}^i = \sum_{j=1}^{N_i} \widehat{\lambda}^i_j Z^i(j), \text{ for } i \in I_d, \tag{5}$$

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 that 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 (1) we reformulate the problem as follows. Assume that we have an initial set of N_0 starting points $z_1, \ldots, z_{N_0}$. The starting points come 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 $\ell = \min_{x \in Z^i(T^i)} x^k$, $u = \max_{x \in Z^i(T^i)} x^k$, $x = (x^1, \ldots, x^{n_i})$. Let $F_1 = F(z_1), \ldots, F_{N_0} = F(z_{N_0})$ be the function evaluations of F at the starting points, N_0 be the number of points where F is known. We solve the following problem:

$$\left.\begin{aligned}
\min_{z,x,\mu,v,\lambda}\ & c^T x + \varepsilon\|\mu\|_p \\
\text{s.t.}\ & \sum_{j=1}^{N_0} \mu_j F_j \le Ax, \\
& z = \sum_{j=1}^{N_0} \mu_j z_j, \\
& \sum_{j=1}^{N_0} \mu_j = 1, \\
& z = (v^1, \ldots, v^{n_0}), \\
& v^i = \sum_{j=1}^{N_i} \lambda_j^i Z^i(j) \text{ for } i \in I_d, \\
& \sum_{j=1}^{N_i} \lambda_j^i = 1 \text{ for } i \in I_d, \\
& \lambda_j^i \ge 0 \text{ for } i \in I_d, 1 \le j \le N_i, \\
& v^i \in [\underline{\theta^i}, \overline{\theta^i}] \text{ for } i \in I_c.
\end{aligned}\right\} \tag{6}$$

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., $z = (v^1, \ldots, v^{n_0})$, $v^i = \sum_{j=1}^{N_i} \lambda_j^i Z^i(j)$, $\sum_{j=1}^{N_i} \lambda_j^i = 1$, $\lambda_j^i \ge 0$, $1 \le j \le N_i$. 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 (6) is convex, the constraints are linear.

Remark 2: In the implementation we choose $p = 1$ if F is multidimensional (i.e., $m \ge 2$). In case of a 1-dimensional F the problem (6) with $p = 1$ is typically unbounded for low ε, and after increasing ε it typically changes towards a binary solution $\mu_i = 1$ for some i, and $\mu_k = 0$ for $k \ne i$. 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 (6) 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 $\widehat{z} \ne \widehat{z}_{\text{start}}$. Hence in case that $m = 1$ we solve (6) twice, with $p = 1$ and $p = 2$. Thus we get two solutions $\widehat{z}_1$ and $\widehat{z}_2$, respectively. Then we wish to compare the two solutions. For $m = 1$, and assuming that $\text{sign}(c \cdot A) = 1$, $A \ne 0$, (1) can be rewritten as

$$\left.\begin{aligned}
\min_z\ & F(z) \\
\text{s.t.}\ & z \in Z(\mathbf{T}),
\end{aligned}\right\} \tag{7}$$

similar to (4). Hence to compare $\widehat{z}_1$ and $\widehat{z}_2$ we just have to evaluate 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 (6) with $p = 2$, otherwise we use the convex combination λ found by (6) with $p = 1$.

Remark 3: One could also approximate F nonlinearly in (6). Hence (6) becomes a nonlinear programming problem which requires a different implementation than the one described in Section 4.

The solution of (6) gives the values of the coefficients $\widehat{\lambda}^i = (\widehat{\lambda}^i_1, \ldots, \widehat{\lambda}^i_{N_i})$, $i \in I_d$, of the convex combinations. 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., Figure 1.

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

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

$$w^i_{k1} = \sum_{\{j|Z^i(j)\in Z^i_{k1}\}} \widehat{\lambda}^i_j, \tag{8}$$

$$w^i_{k2} = \sum_{\{j|Z^i(j)\in Z^i_{k2}\}} \widehat{\lambda}^i_j. \tag{9}$$

We split T^i into T^i_1 and T^i_2 across the edge $\widehat{k} = \arg\min_k |w^i_{k1} - \frac{1}{2}|$, i.e., the edge where the weight on the one side and the weight on the other side are closest to 50%.

Remark 4: Thus in total we find up to $2^{|I_d|}$ possible branches. In this study we give no answer to the question on which variable to split. However, this decision – namely whether or how to join the branches in an optimization algorithm in order to find a division of the search space after computing T^i_1, T^i_2 – may seriously affect the performance of the algorithm and depends on how the algorithm handles the resulting subproblems. Section 5 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 from the function evaluations $F_1, \ldots, F_{N_0}$, so black box functions F can be handled which is often occurring in real-life applications.

4 Implementation

We have implemented our approach in MATLAB. The implementation is available online at `www.martin-fuchs.net/downloads.php`. To solve (6), we use the public domain package CVX [6]. The following functions realize the approach presented in Section 3.

- `CONVRELAX.m`

  ```
  function [zrel,lambda,mue,solverstatus]=convrelax(z,f,Z,isdiscrete,e,c,A,p)
  ```

  This function solves problem (6) using CVX [6].
  `z` corresponds to $\begin{pmatrix} z_1^1 & \ldots & z_1^{n_z} \\ \vdots & \ddots & \vdots \\ z_{N_0}^1 & \ldots & z_{N_0}^{n_z} \end{pmatrix}$.
  `f` corresponds to $\begin{pmatrix} F_1^1 & \ldots & F_1^m \\ \vdots & \ddots & \vdots \\ F_{N_0}^1 & \ldots & F_{N_0}^m \end{pmatrix}$.
  `Z{i}` corresponds to τ^i.
  `isdiscrete` is an $1 \times n_0$ boolean vector with ith entry 1 iff $i \in I_d$.
  `e`, `c`, `A`, `p` correspond to ε, c, A, p.
  `zrel` corresponds to the solution of (6).
  `lambda{i}` corresponds to $\widehat{\lambda}^i$.
  `mue` corresponds to μ.
  `solverstatus` is the status of CVX. It helps adjusting `e`.

- `CONVRELSPLIT.m`

  ```
  function [Zsplit,splitperm,mstsplit]=convrelsplit(lambda,Z,isdiscrete,mst)
  ```

This function performs the splitting as described in the last section.
`lambda`, `Z`, `isdiscrete` as above,
`mst{i}` contains the minimum spanning tree of $Z^i(T^i)$.
`Zsplit{i,1}`, `Zsplit{i,2}` are the splits of τ^i.
`splitperm{i,1}`, `splitperm{i,2}` are the splits T_1^i, T_2^i.
`mstsplit{i,1}`, `mstsplit{i,2}` are the minimum spanning trees belonging to the splits T_1^i, T_2^i.

5 A simple solver strategy

To demonstrate our splitting method in examples (cf. Section 6) we have also implemented it in a simple solver with the following branching strategy: We round the relaxed solution $\widehat{z}$ of (6) to the next feasible point of (1) $\widehat{z}_{\text{round}} := \arg\min_{\{z\in Z(\mathbf{T})\}} \|z-\widehat{z}\|_2$ and start from $\widehat{z}_{\text{round}}$ a linesearch on $\mathbf{T}$. The function evaluations during linesearch 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 linesearch. 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., $T_1 = T^1 \times \cdots \times T_1^i \times \cdots \times T^{n_0}$, $T_2 = T^1 \times \cdots \times T_2^i \times \cdots \times T^{n_0}$, where T_1^i, T_2^i 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 linesearch 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 linesearch until satisfaction. As a stopping criterion one may choose, e.g., that the optimal solution found by the linesearch has not been improved for N_{iter} times.

This simple strategy is already suited to demonstrate the usefulness of our splitting routine in Section 6. It can probably be improved by a more intelligent branch selection for the next iteration step. Implementing a more sophisticated solver that makes use of our splitting is a current research topic out of the scope of this paper.

6 Illustrative examples

In this section we present two examples of application of our method. The first example is an illustration of the method, and we include this example under the filename `example.m` in the package containing our method downloadable at `www.martin-fuchs.net/downloads.php`. The second example is more complicated and arises in real-life spacecraft system design, cf. [12].

6.1 A simple illustration

Let $\mathbf{T} = T^1 \times T^2 \times T^3$, $T^1 = \{1,\ldots,7\}$, $T^2 = [0,2]$, $T^3 = \{1,\ldots,10\}$, $I_d = \{1,3\}$, $I_c = \{2\}$, let the associated table τ^1 be as shown in Table 2 and $\tau^3 = (1,2,\ldots,10)^T$. The function F from our problem statement is given by

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

The optimal solution of (1) with these specifications is obviously given as $\theta = (5,0,10)$. Relaxing the discrete search space to a continuous space, the solution of (1) is obtained at $\widehat{z} = (-\frac{1}{2}, \frac{3}{4}, 0, 10)$.

We solve problem (6) with $N_0 = 20$, $c = 1$, $A = 1$, $\varepsilon = 10^2$ two times for $p = 1$ and $p = 2$, as described in Remark 3 since $F(z) \in \mathbb{R}^1$.

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 Figure 1.

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

$$\sum_{j=1}^{7} \widehat{\lambda}_j^1 Z^1(j) = (-0.29, 0.83), \tag{11}$$

which is close to $(\widehat{z}^1, \widehat{z}^2)$, so our implemented 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 $\sum_{j\in\{1,2,3,4\}} \widehat{\lambda}_j^1 = \sum_{j\in\{5,6,7\}} \widehat{\lambda}_j^1 =$

Table 2: Tabulated data $Z^1(T^1)$

θ^1	$Z^i(\theta^i)$	
1	4	0
2	4	1
3	6	0
4	5	3
5	-4	0
6	-8	0
7	-4	2

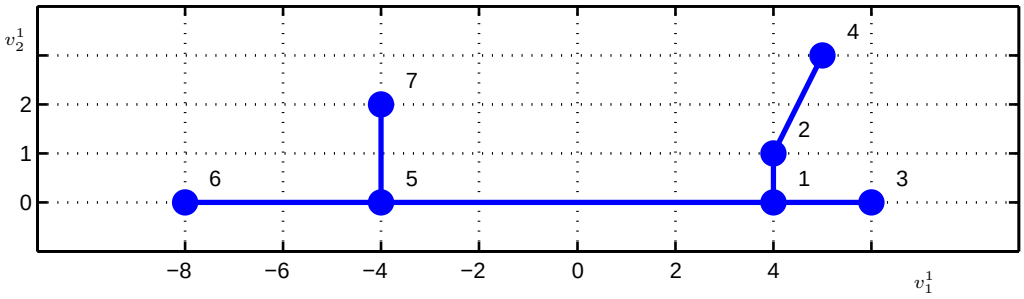

Figure 1: Graph of the minimum spanning tree of $Z^1(T^1)$.

0.5. Thus it is not surprising that our implemented method actually splits T^1 into $T_1^1 = \{1,2,3,4\}$ and $T_2^1 = \{5,6,7\}$ in most cases. Sometimes it splits off the leaf 7, i.e., $T_1^1 = \{1,2,3,4,5,6\}$ and $T_2^1 = \{7\}$, or it finds $T_1^1 = \{1,3,5,6,7\}$, $T_2^1 = \{2,4\}$, depending on the approximation and the convex combination found from (6), 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 value of $\widehat{\lambda}_{10}^3$ close to 1. Except from few cases where we find $T_1^3 = \{1,2,3,4,5,6,7,8\}$, $T_2^3 = \{9,10\}$, we find the expected splitting of T^3 into $T_1^3 = \{1,2,3,4,5,6,7,8,9\}$ and $T_2^3 = \{10\}$ with our implemented method.

We have also used the solver strategy described in Section 5 to find the optimum $\theta = (5,0,10)$ (this is not contained in `example.m`). In the first iteration we split T^3 and find the branch $T_{\text{new}} = T^1 \times T^2 \times T_{\text{new}}^3$, $T_{\text{new}}^3 = \{10\}$ 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 $\theta = (5,0,10)$.

6.2 A real-life application

This example is a problem of optimization under uncertainty in spacecraft system design, described in [12]. After reasonable simplification, the problem can be formulated as in (4), 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.

We have applied the solver strategy described in Section 5 to this example and 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 it did not have a single 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] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] N. Alexandrov and M. Hussaini. Multidisciplinary design optimization: State of the art. In *Proceedings of the ICASE/NASA Langley Workshop on Multidisciplinary Design Optimization*, Hampton, Virginia, USA, 1997.

[3] C. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. Oxford University Press, 1995.

[4] M. Fuchs, D. Girimonte, D. Izzo, and A. Neumaier. *Robust Intelligent Systems*, chapter Robust and automated space system design, pages 251–272. Springer, 2008.

[5] M. Fuchs and A. Neumaier. Autonomous robust design optimization with potential clouds. *International Journal of Reliability and Safety*, 3(1/2/3):23–34, 2009.

[6] M. Grant and S. Boyd. CVX: A system for disciplined convex programming. `http://www.stanford.edu/~boyd/cvx/cvx_usrguide.pdf`, `http://www.stanford.edu/~boyd/cvx/`, 2009.

[7] W. Huyer and A. Neumaier. Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14(4):331–355, 1999.

[8] W. Huyer and A. Neumaier. SNOBFIT – Stable Noisy Optimization by Branch and Fit. *ACM Transactions on Mathematical Software*, 35(2), 2008. Article 9, 25 pages.

[9] D. Jones, C. Perttunen, and B. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

[10] S. Leyffer. *Deterministic Methods for Mixed Integer Nonlinear Programming*. PhD thesis, University of Dundee, Department of Mathematics & Computer Science, 1993.

[11] G. Nemhauser and L. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, 1988.

[12] A. Neumaier, M. Fuchs, E. Dolejsi, T. Csendes, J. Dombi, B. Banhelyi, and Z. Gera. Application of clouds for modeling uncertainties in robust space system design. ACT Ariadna Research ACT-RPT-05-5201, European Space Agency, 2007.

[13] R. Parker and R. Rardin. *Discrete optimization*. Academic Press, 1988.

[14] M. Tawarmalani and N. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591, 2004.

[15] E. Weisstein. Minimum spanning tree. MathWorld – A Wolfram Web Resource, 2008. Available on-line at: `http://mathworld.wolfram.com/MinimumSpanningTree.html`.