Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

ANDREAS BLEY
THORSTEN KOCH
LINGFENG NIU

Experiments with nonlinear extensions to SCIP

supported by the DGF Research Center Matheon

ZIB-Report 08-28 (July 2008)

Experiments with nonlinear extensions to SCIP*

Andreas Bley Thorsten Koch Lingfeng Niu†

revised May 8, 2009

Abstract

This paper describes several experiments to explore the options for solving a class of mixed integer nonlinear programming problems that stem from a real-world mine production planning project. The only type of nonlinear constraints in these problems are bilinear equalities involving continuous variables, which enforce the ratios between elements in mixed material streams.

A branch-and-bound algorithm to handle the integer variables has been tried in another project. However, this branch-and-bound algorithm is not effective for handling the nonlinear constraints. Therefore state-of-the-art nonlinear solvers are utilized to solve the resulting nonlinear subproblems in this work. The experiments were carried out using the NEOS server for optimization. After finding that current nonlinear programming solvers seem to lack suitable preprocessing capabilities, we preprocess the instances beforehand and use a heuristic approach to solve the nonlinear subproblems.

In the appendix, we explain how to add a polynomial constraint handler that uses IPOPT as embedded nonlinear programming solver for the constraint programming framework SCIP. This is one of the crucial steps for implementing our algorithm in SCIP. We briefly describe our approach and give an idea of the work involved.

1 Introduction

Mixed Integer Nonlinear Programming (MINLP) is an optimization problem with continuous and discrete variables and nonlinearities in the objective function or at least one of the constraint functions. Many real world applications are suitable to be modeled as MINLP, because it can simultaneously optimize the system structure (discrete) and parameters (continuous). One particular type of nonlinear constraints that is often encountered are mixing or blending constraints, which enforce that the mixing ratio of materials is

*Project supported by the DGF Research Center Matheon

†State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, AMSS, CAS, Beijing

the same in certain material streams. Mathematically, these mixing constraints can be expressed as bilinear equations. In this paper, we consider solving a special mixed integer nonlinear programming problem arising from the problem of scheduling the production of an open-pit mine, whose nonlinear constraints are just bilinear equations.

1.1 Problem Description

The mine can be considered as a set of *panels*, which are identified by the numbers $1, \ldots, N$. Each panel represents a volume of underground material. In particular, there are a number of mineral *attributes* which are of interest in the mining operation. We denote the set of all attributes of interest by A. For each panel, the quantity of attributes in A is assumed to be known. The value $\alpha_i^a \in \mathbb{Z}_+$ denotes the quantity of attributes $a \in A$ in panel $i \in \{1, \ldots, N\}$. A special attribute is *rock*, which describes the total tonnes of underground material in each panel.

In the operation of the mine, *rock* is *extracted* from the mine and then can be sent to one of three destinations: *waste*, *processing*, or into a *stockpile*. Material that is sent to waste can be ignored in scheduling production. Processing is performed at a processing plant that extracts the valuable material from the rock. It is assumed that any material processed is immediately sold. All material put into the stockpile is immediately mixed, thus becoming homogeneous. Nothings happens in the stockpiles, so their attributes are simply the sum of those of their constituent ingredients. At any stage, the material on the stockpile can be sent to processing.

An important feature of an open-pit mining operation is the *precedence* structure. For each panel $i = 1, \ldots, N$, mining engineering software is able to calculate the set of other panels, denoted by $\mathrm{Pred}(i) \subseteq \{1, \ldots, N\}$, that must be completely extracted before the extraction of panel i can safely begin.

The profitability of a mine is calculated as its net present value (NPV), which depends crucially on *when* the valuable material extracted is sold. Thus the life of the mine is divided into T periods. The mine *schedule* specifies the mining activities, i.e., the material to be extracted, processed and stockpiled in each period. The profit that these activities yield must be multiplied by a period discount factor, which we denote by δ_t for each $t = 1, \ldots, T$, and the sum of the profits, weighted by the discount factor for the period in which they occur, gives the NPV. We seek a schedule that maximizes this NPV.

In this specific application, the nonlinearities arise from the use of stockpiles. In fact, the open pit mine production planning problem without the use of stockpiles can be formulated and solved very efficiently as mixed integer linear programming problems. Several variants of such problems are described in [13, 12, 6]. We will introduce a MINLP formulation for a mine with a single infinite-capacity stockpile briefly in the next subsection.

1.2 MINLP formulation

In this work, we only consider three attributes: the total *rock* tonnage, the *ore* content, and the *metal* content, i.e., $A = \{rock, ore, met\}$. We make the simplifying assumption that the stockpile is empty at the start of the first time period and at the end of the planning horizon. Furthermore, we assume that material taken from the stockpile is taken off instantaneously at the start of the respective period. Material added to the stockpile is added at the end of a period, i.e., after the removal of the material taken off for processing in the same period, but before the removal of the material taken off for processing in the following period.

The following parameters and variables are used for our model description.

Parameters (all non-negative):

p_t^{met}	metal price per unit in period t
c_t^m	extraction cost per ton of rock in period t
c_t^p	processing cost per ton of ore in period t
b_t^m	extraction capacity in tons of ore in period t
b_t^p	processing capacity in tons of rock in period t

Note that the processing capacity and cost depend only on the ore tonnage of the material sent for processing, while the mining capacity and cost depend on the rock tonnage of the material extracted.

Variables

For each panel $i = 1, .., N$ and each period $t = 1, ..., T$:

$f_{i,t}^m \in [0, 1]$	fraction of panel i extraced in period t
$f_{i,t}^p \in [0, 1]$	fraction of panel i sent directly for processing in period t
$f_{i,t}^s \in [0, 1]$	fraction of panel i sent to stockpile in period t
$f_{i,t}^o \in [0, 1]$	fraction of panel i sent from stockpile to stockpile in period t
$f_{i,t}^r \in [0, 1]$	fraction of panel i remaining in stockpile from period $t-1$ to period t

In order to ensure that the extraction respects the given precedence relations required for safe mining, we introduce binary decision variables:

$$x_{i,t} = \begin{cases} 1 & \text{panel } i \text{ is completely extracted by end of period } t \text{ or earlier,} \\ 0 & \text{otherwise.} \end{cases}$$

Then the **M**ine **P**roduction **S**cheduling problem with a single **S**tockpile (MPSS) can be formulated as follows:

$$\max \sum_{t=1}^{T} \delta_t \left(\sum_{i=1}^{N} \left(\left(p_t^{met} \alpha_i^{met} - c_t^p \alpha_i^{ore}\right)\left(f_{i,t}^p + f_{i,t}^o\right) - c_t^m \alpha_i^{rock} f_{i,t}^m\right)\right) \tag{1.1}$$

$$
\begin{array}{lll}
\text{s.t.} & f^p_{i,t} + f^s_{i,t} \le f^m_{i,t}, & \forall i = 1,...,N, t = 1,...,T. \\
& \sum_{t=1}^T f^m_{i,t} \le 1, & \forall i = 1,\cdots,N. \\
& \sum_{t'=1}^t f^m_{i,t'} \ge x_{i,t}, & \forall i = 1,...,N, t = 1,...,T. \\
& \sum_{t'=1}^t f^m_{i,t'} \le x_{j,t}, & \forall i = 1,...,N, t = 1,...,T, j \in \text{Pred}(i). \\
& x_{i,t} \le x_{i,t+1}, & \forall i = 1,...,N, t = 1,...,T-1. \\
& f^r_{i,t} + f^s_{i,t} = f^o_{i,t+1} + f^r_{i,t+1}, & \forall i = 1,\cdots,N, t = 1,\cdots,T-1. \\
& \sum_{i=1}^N \alpha_i^{rock} f^m_{i,t} \le b^m_t, & \forall t = 1,...,T. \\
& \sum_{i=1}^N \alpha_i^{ore}(f^p_{i,t} + f^o_{i,t}) \le b^p_t, & \forall t = 1,...,T. \\
& f^o_{i,t} f^r_{j,t} = f^o_{j,t} f^r_{i,t}, & \forall i,j = 1,\cdots,N, t = 1,\cdots,T. \\
& 0 \le f^m_{i,t}, f^s_{i,t}, f^p_{i,t}, f^o_{i,t}, f^r_{i,t} \le 1, & \forall i = 1,...,N, t = 1,...,T. \\
& f^o_{i,1} = f^r_{i,1} = f^r_{i,T} = f^s_{i,T} = 0, & \forall i = 1,\cdots,N. \\
& x_{i,t} \in \{0,1\}, & \forall i = 1,...,N, t = 1,...,T.
\end{array}
$$

where the bilinear constraint $f^o_{i,t} f^r_{j,t} = f^o_{j,t} f^r_{i,t}$ represents the requirement that all the material in the stockpile must be homogeneously mixed. This MINLP contains $5NT$ continuous variables and NT binary variables. The number of nonlinear constraints is NT. For a mine with 125 panels and 25 time periods (this is the scale of one of our experiment problems), the scale of the model is 3125 binary variables and 390625 bilinear constraints. Since this is considered a large scale instance regarding current MINLP solving techniques, we would prefer a more compact formulation.

Notice that variables $f^o_{\cdot,\cdot}$ and $f^r_{\cdot,\cdot}$ are only used to indicate the material flows in and out of the stockpile. Now consider using the following aggregated variables instead of reducing the number of variables and nonlinear constraints . We define for each period $t = 2, ..., T$ and each attribute $a \in \{ore, met\}$ the following variables:

q^a_t units of attribute a in the stockpile at the end of period t,
o^a_t units of attribute a removed from the stockpile at the start of period t (i.e., at the end of period $t-1$).

As all material put into the stockpile is mixed, we have to ensure that the material taken off the stockpile at the start of a period has the same metal–ore composition as the material contained in the stockpile at the end of the preceding period. Therefore, the following bilinear equalities are needed,

$$o^{ore}_t q^{met}_{t-1} = o^{met}_t q^{ore}_{t-1} \qquad \forall t = 2, \ldots, T.$$

which is also the only group of nonlinear constraints that are necessary to model the mixing property of the stockpile. Then we get another MINLP model:

$$\max \sum_{t=1}^T \delta_t \Big(\sum_{i=1}^N ((p^{met}_t \alpha^{met}_i - c^p_t \alpha^{ore}_i) f^p_{i,t} - c^m_t \alpha^{rock}_i f^m_{i,t}) + p^{met}_t o^{met}_t - c^p_t o^{ore}_t \Big) \tag{1.2}$$

$$
\begin{array}{lll}
\text{s.t.} & f^p_{i,t} + f^s_{i,t} \le f^m_{i,t}, & \forall i = 1, ..., N, t = 1, ..., T. \\
& \sum_{t=1}^{T} f^m_{i,t} \le 1, & \forall i = 1, \cdots, N. \\
& \sum_{t'=1}^{t} f^m_{i,t'} \ge x_{i,t}, & \forall i = 1, ..., N, t = 1, ..., T. \\
& \sum_{t'=1}^{t} f^m_{i,t'} \le x_{j,t}, & \forall i = 1, ..., N, t = 1, ..., T, j \in \text{Pred}(i). \\
& x_{i,t} \le x_{i,t+1}, & \forall i = 1, ..., N, t = 1, ..., T-1. \\
& \sum_{i=1}^{N} \alpha^a_i f^s_{i,1} = q^a_1, & \forall a \in \{ore, met\} \\
& q^a_{t-1} - o^a_t + \sum_{i=1}^{N} \alpha^a_i f^s_{i,t} = q^a_t, & \forall t = 2, ..., T-1, a \in \{ore, met\}. \\
& q^a_{T-1} - o^a_T + \sum_{i=1}^{N} \alpha^a_i f^s_{i,T} = 0, & \\
& \sum_{i=1}^{N} \alpha^{rock}_i f^m_{i,t} \le b^m_t, & \forall t = 1, ..., T. \\
& \sum_{i=1}^{N} \alpha^{ore}_i f^p_{i,1} \le b^p_1, & \\
& \sum_{i=1}^{N} \alpha^{ore}_i f^p_{i,t} + o^{ore}_t \le b^p_t, & \forall t = 2, ..., T. \\
& o^{ore}_t q^{met}_{t-1} = o^{met}_t q^{ore}_{t-1}, & \forall t = 2, ..., T-1. \\
& o^a_T = q^a_{T-1}, & \forall a \in \{ore, met\}. \\
& o^a_t, q^a_{t-1} \ge 0, & \forall t = 2, ..., T, a \in \{ore, met\}. \\
& 0 \le f^m_{i,t}, f^s_{i,t}, f^p_{i,t} \le 1, & \forall i = 1, ..., N, t = 1, ..., T. \\
& x_{i,t} \in \{0, 1\}, & \forall i = 1, ..., N, t = 1, ..., T.
\end{array}
$$

This MINLP model has $T-2$ nonlinear constraints and $4NT + 2T - 2$ variables, whose scale is smaller than (1.1). We call this MINLP the aggregated MPSS formulation due to the use of aggregated variables q^a_t and o^q_t. Model (1.1) is called the Basic Warehouse formulation.

2 The solution of the MPSS model

In this paper we will concentrate on the situation when a feasible schedule for mining is given (that is, all binary variables in the model are fixed), how to solve the resulting pure nonlinear programming (NLP) effectively and efficiently. Because both the number of variables and nonlinear constraints in (1.2) are much smaller than the number in (1.1), we only consider (1.2) in the following.

When binary variables $x_{\cdot,\cdot}$ are fixed, the formulation of the resulting pure NLP is the same as (1.2). The only difference is that x is parameter instead of variable this time. For completeness, we restate the pure continuous problem here:

$$
\max \sum_{t=1}^{T} \delta_t \Big(\sum_{i=1}^{N} ((p^{met}_t \alpha^{met}_i - c^p_t \alpha^{ore}_i) f^p_{i,t} - c^m_t \alpha^{rock}_i f^m_{i,t}) + p^{met}_t o^{met}_t - c^p_t o^{ore}_t\Big) \tag{2.1}
$$

$$
\begin{array}{lll}
\text{s.t.} & f^p_{i,t} + f^s_{i,t} \leq f^m_{i,t}, & \forall i = 1, ..., N, t = 1, ..., T. \\
& \sum_{t=1}^{T} f^m_{i,t} \leq 1, & \forall i = 1, \cdots, N. \\
& \sum_{t'=1}^{t} f^m_{i,t'} \geq x_{i,t}, & \forall i = 1, ..., N, t = 1, ..., T. \\
& \sum_{t'=1}^{t} f^m_{i,t'} \leq x_{j,t}, & \forall i = 1, ..., N, t = 1, ..., T, j \in \text{Pred}(i). \\
& x_{i,t} \leq x_{i,t+1}, & \forall i = 1, ..., N, t = 1, ..., T-1. \\
& \sum_{i=1}^{N} \alpha^a_i f^s_{i,1} = q^a_1, & \forall a \in \{ore, met\} \\
& q^a_{t-1} - o^a_t + \sum_{i=1}^{N} \alpha^a_i f^s_{i,t} = q^a_t, & \forall t = 2, ..., T-1, a \in \{ore, met\}. \\
& q^a_{T-1} - o^a_T + \sum_{i=1}^{N} \alpha^a_i f^s_{i,T} = 0, & \\
& \sum_{i=1}^{N} \alpha^{rock}_i f^m_{i,t} \leq b^m_t, & \forall t = 1, ..., T. \\
& \sum_{i=1}^{N} \alpha^{ore}_i f^p_{i,1} \leq b^p_1, & \\
& \sum_{i=1}^{N} \alpha^{ore}_i f^p_{i,t} + o^{ore}_t \leq b^p_t, & \forall t = 2, ..., T. \\
& o^{ore}_t q^{met}_{t-1} = o^{met}_t q^{ore}_{t-1}, & \forall t = 2, ..., T-1. \\
& o^a_T = q^a_{T-1}, & \forall a \in \{ore, met\}. \\
& o^a_t, q^a_{t-1} \geq 0, & \forall t = 2, ..., T, a \in \{ore, met\}. \\
& 0 \leq f^m_{i,t}, f^s_{i,t}, f^p_{i,t} \leq 1, & \forall i = 1, ..., N, t = 1, ..., T.
\end{array}
$$

2.1 Solving the continuous nonlinear subproblem

Two mines are used in our experiment: marvin (85 panels and 17 time periods) and ob25 (125 panels and 25 time periods). For the construction of our nonlinear subproblems, the values of the x variables are needed. Notice that the x variables have the same meaning in all the models mentioned above. So different sets of x values which have been fixed to the values of a global near-optimal solution computed by a branch-and-bound algorithm for the aggregated model (1.1), warehouse model (1.2) and extended warehouse model are used. In the following, we will use "agg", "wh" and "ewh" to represent the aggregated model, warehouse model and extended model, respectively. All together, 9 different datasets are used: marvin (agg, wh, ewh); marvin2 (agg, wh, ewh) and ob25 (agg, wh, ewh).

We formulated our nonlinear optimization problem in the AMPL [14] modeling language and tried to solve the resulting NLP with several state-of-the-art nonlinear programming solvers available through the NEOS server [9, 8, 4]: filterQP version 20020316 [5], IPOPT version 3.3.3 [16], KNITRO version 5.2.0 [2], Lancelot release A [3], LOQO version 6.06 [15], MINOS version 5.51 [11], PENNON version 2.2 [10], and SNOPT version 7.2-8 [7].

When applying these solvers off-the-shelf, we experienced substantial problems. Only a few solvers were able to find a local solution to at least some datasets. Most solvers were not able to find a feasible solution at all. Computational results for marvin, marvin2 and ob25 are listed in Table 1, 2, and 3, respectively. Column "Solver" gives the name of different solvers. The optimal objective function values found are listed in the column "Solution info." if the corresponding solver can find a optimal solution successfully. Otherwise, the reason for failure is given.

Observing that several solvers complain about reaching the iteration limit, we doubled the maximum iteration number for these solvers. However, all of those solvers still failed, indicating that enlarging the iteration number or computing time alone has only very little impact.

There are also some solvers which complain that the models are badly scaled. We concluded that the NLP solvers were not capable of doing the necessary preprocessing automatically. Hence, the preprocessing for the continuous nonlinear subproblems was done by us in advance. The fixed integer variables are removed and the resulting implications were propagated on the other variables' domains in order to eliminate variables that implicitly have been fixed by these implications. If, for example, variable $x_{i,t}$ has been fixed to 1 (i.e., panel i is completely extracted by the end of period t), it follows that $x_{i,\hat{t}} = 1$ and $f^m_{i,\hat{t}} = 0$ for all $\hat{t} = t+1,\dots,T$ and, consequently, all variables $x_{i,t}$ and $f^m_{i,\hat{t}}$ can be removed from the subproblem. This simple preprocessing technique reduces the scale of the NLP considerably. Take the marvin dataset for example, preprocessing reduced the number of f-variables in the marvin test dataset from 4335 to 339 and the number of constraints accordingly.

Applying our preprocessing techniques prior to passing the subproblem to the nonlinear programming solvers resulted in a much better performance of the solvers. Most of them were now able to find feasible and locally optimal solutions within a few seconds. Table 4 gives an overview of the results, which indicate that careful preprocessing is important to successful NLP solving.

All the nonlinear solvers mentioned above only seek local solutions, i.e., a point at which the objective function is larger than at all other feasible points in the vicinity. They do not necessarily find a global optimal solution. One the other hand, the branch-and-bound algorithm needs to know the global optimum or at least a reasonable upper bound for each node. How to cope with the gap between the nonlinear solvers' ability of only finding the local optima and the requirement from branch-and-bound framework for global optima will be our next topic in this section.

We give the following heuristic algorithm which we think can "lead" the iteration to a local optimum with a larger objective value.

The intuitive explanation is that the objective value is reduced step by step and the previous solution provides a good starting point for the next step. Suppose there are n nonlinear constraints, in each step we add at least one of the remaining constraints to the model. We need to call the nonlinear solver at most n times. It can be expected that, with a good starting point

Algorithm 1 Heuristic Algorithm for Finding Global Solution

Step 1. Initialization. Drop the nonlinear constraints and solve the resulting linear programming to get an initial point.

Step 2. Construct New Subproblem. Choose one nonlinear constraint which is violated by the current solution. Add this constraint to the problem formulation and solve the new problem starting with the solution of the last step as initial point.

Step 3. Termination Test. Stop the iteration if all the nonlinear constraints are satisfied numerically. Otherwise, goto Step 2.

given, the expense for each iteration is not so high.

For our special problem, the nonlinear constraints represent the requirement of homogeneous materials in the stockpile. In other words, they forbid the free processing to happen in the stockpile. With this background knowledge, we can simplify Algorithm 1 as follows:

Algorithm 2 Heuristic Algorithm for Finding Global Solution

Step 1. Initialization. Drop the nonlinear constraints and solve the resulting linear programming. Let $t = 2$

Step 2. Construct New Subproblem. Choose the nonlinear constraint corresponding to the t-th time period and add it to the problem formulation. Add this constraint to the problem formulation and solve the new problem starting with the solution of the last step as initial point.

Step 3. Termination Test. Stop the iteration if $t = T - 1$. Otherwise, set $t := t + 1$ and goto Step 2.

The computational results for Algorithm 2 are listed in Table 5. The underlying NLP solver chosen in this set of experiments is filterQP. The reason is that we think Algorithm 2, which considers nonlinear constraints one by one, is very suitable to be used together with the active set method, which is just the underlying algorithm in filterQP.

Column “OneByOne” gives the solution found by Algorithm 2. Column ”AllInOne” shows the solution obtained by solving the preprocessed NLP model (2.1) with filterQP directly. Column “original solution” contains the objective function values for the solution found by a branch-and-bound algorithm from another project. From these results we can see that our solution has larger objective function values than the original solution. We experienced that the solutions obtained by NLP solvers usually fit the nonlinear constraints better numerically, i.e., the corresponding constraint violation value is smaller.

We also tried the interior point solver IPOPT, but the results obtained by Algorithm 2 using IPOPT are almost the same as the results obtained by considering all the nonlinear constraints together. On the other hand, we find most of the time, the solution obtained by filterQP using Algorithm 2 is as good as the solution found by using IPOPT directly. So we guess maybe these solution are already the global optimal. Therefore, now our problem will be to estimate the quality of our solution.

2.1.1 Piecewise linearization to estimate the quality of NLP solution

For the general non-convex NLP, global solutions are not only difficult to locate, but also difficult to identify. There are no easily computable criteria to verify the global optimality of a given solution. One of the common approaches to estimate the quality of the solution is computing an upper bound (for the maximization problems) by piecewisely linearizing the nonlinear constraints and solving the resulting MIP. In this work, we also utilize this technique.

Firstly, a set of threshold values for the metal-to-ore grade of the mixed material in the stockpile is introduced. For $t \in \{1, \cdots, T-2\}$, we define values $r_{t,1} < \cdots < r_{t,L}$, where $r_{t,1} = r_t^{min}$, $r_{t,L} = r_t^{max}$ are estimated lower and upper bounds for q_t^{met}/q_t^{ore}. Since the material in the stockpile is homogeneously mixed, the grade of the material in the stockpile at the end of period of t should be the same as the grade of the material taken off the stockpile at the start of the next period. Hence, there exists a number $l(t)$ between 1 and L for each t, such that

$$r_{t,l(t)} \leq \frac{q_t^{met}}{q_t^{ore}} = \frac{o_{t+1}^{met}}{o_{t+1}^{ore}} < r_{t,l(t)+1}$$

i.e.,

$$\begin{aligned} r_{t,l(t)} q_t^{ore} \leq q_t^{met} < r_{t,l(t)+1} q_t^{ore} \\ r_{t,l(t)} o_{t+1}^{ore} \leq o_{t+1}^{met} < r_{t,l(t)+1} o_{t+1}^{ore} \end{aligned} .$$

We can relax this group of constraints by introducing binary variable

$$y_{t,l} = \begin{cases} 1 & \text{if } q_t^{met}/q_t^{ore} \geq r_{t,l} , \\ 0 & \text{otherwise} \end{cases}$$

for all $t \in \{1, \cdots, T-2\}$ and $l \in \{1, \ldots, L\}$ such that

$$\begin{aligned} r_{t,l} q_t^{ore} - q_t^{met} + r_{t,l} B_t^{ore} y_{t,l} &\leq r_{t,l} B_t^{ore} && (2.2a) \\ q_t^{met} - r_{t,l} q_t^{ore} - B_t^{met} y_{t,l} &< 0 && (2.2b) \\ r_{t,l} o_{t+1}^{ore} - o_{t+1}^{met} + r_{t,l} B_t^{ore} y_{t,l} &\leq r_{t,l} B_t^{ore} && (2.2c) \\ o_{t+1}^{met} - r_{t,l} o_{t+1}^{ore} - B_t^{met} y_{t,l} &< 0 && (2.2d) \\ y_{t,l} &\geq y_{t,l+1} && (2.2e) \end{aligned}$$

where B_t^{ore} and B_t^{met} are upper bounds on the amount of ore and metal that can be contained in the stockpile at the beginning of period t, respectively.

Replacing the nonlinear mixing equalities (1.3) with the linear constraints (2.2) and the additional binary variables $y_{t,l}$, we obtain the following MIP relaxation of the original formulation (2.1):

$$\max \sum_{t=1}^{T} \delta_t (\sum_{i=1}^{N} ((p_t^{met} \alpha_i^{met} - c_t^p \alpha_i^{ore}) f_{i,t}^p - c_t^m \alpha_i^{rock} f_{i,t}^m) + p_t^{met} o_t^{met} - c_t^p o_t^{ore}) \tag{2.3}$$

$$\begin{aligned}
\text{s.t.} \quad & f_{i,t}^p + f_{i,t}^s \le f_{i,t}^m, && \forall i = 1, ..., N, t = 1, ..., T. \\
& \textstyle\sum_{t=1}^{T} f_{i,t}^m \le 1, && \forall i = 1, \cdots, N. \\
& \textstyle\sum_{t'=1}^{t} f_{i,t'}^m \ge x_{i,t}, && \forall i = 1, ..., N, t = 1, ..., T. \\
& \textstyle\sum_{t'=1}^{t} f_{i,t'}^m \le x_{j,t}, && \forall i = 1, ..., N, t = 1, ..., T, j \in \text{Pred}(i). \\
& x_{i,t} \le x_{i,t+1}, && \forall i = 1, ..., N, t = 1, ..., T-1. \\
& \textstyle\sum_{i=1}^{N} \alpha_i^a f_{i,1}^s = q_1^a, && \forall a \in \{ore, met\} \\
& q_{t-1}^a - o_t^a + \textstyle\sum_{i=1}^{N} \alpha_i^a f_{i,t}^s = q_t^a, && \forall t = 2, ..., T-1, a \in \{ore, met\}. \\
& q_{T-1}^a - o_T^a + \textstyle\sum_{i=1}^{N} \alpha_i^a f_{i,T}^s = 0, && \\
& \textstyle\sum_{i=1}^{N} \alpha_i^{rock} f_{i,t}^m \le b_t^m, && \forall t = 1, ..., T. \\
& \textstyle\sum_{i=1}^{N} \alpha_i^{ore} f_{i,1}^p \le b_1^p, && \\
& \textstyle\sum_{i=1}^{N} \alpha_i^{ore} f_{i,t}^p + o_t^{ore} \le b_t^p, && \forall t = 2, ..., T. \\
& r_{t,l} q_t^{ore} - q_t^{met} + r_{t,l} B_t^{ore} y_{t,l} \le r_{t,l} B_t^{ore}, && \forall t = 1, ..., T-2, l = 1, \cdots, L. \\
& q_t^{met} - r_{t,l} q_t^{ore} - B_t^{met} y_{t,l} < 0, && \forall t = 1, ..., T-2, l = 1, \cdots, L. \\
& r_{t,l} o_{t+1}^{ore} - o_{t+1}^{met} + r_{t,l} B_t^{ore} y_{t,l} \le r_{t,l} B_t^{ore}, && \forall t = 1, ..., T-2, l = 1, \cdots, L. \\
& o_{t+1}^{met} - r_{t,l} o_{t+1}^{ore} - B_t^{met} y_{t,l} < 0, && \forall t = 1, ..., T-2, l = 1, \cdots, L. \\
& y_{t,l} \ge y_{t,l+1}, && \forall t = 1, ..., T-2, l = 1, \cdots, L. \\
& o_T^a = q_{T-1}^a, && \forall a \in \{ore, met\}. \\
& o_t^a, q_{t-1}^a \ge 0, && \forall t = 2, ..., T, a \in \{ore, met\}. \\
& 0 \le f_{i,t}^m, f_{i,t}^s, f_{i,t}^p \le 1, && \forall i = 1, ..., N, t = 1, ..., T. \\
& y_{t,l} \in \{0, 1\}, && \forall t = 1, ..., T-2, l = 1, ..., L.
\end{aligned}$$

Clearly, the optimal objective function value for this MIP is an upper bound for the global optimal solution of nonlinear programming (2.1). The accuracy of this approximation depends on the number and the values of the threshold grades $r_{t,l}$. Generally speaking, the more threshold grades we use, the finer is the approximation we can get. But increasing the number of grades also increases the time needed to solve MIP (2.3). In our following tests, we set $L = 10$ and evenly divide the possible grade range. We compute the values of $y_{t,l}$ according to the NLP solution and take them as the starting point for the MIP. So if our solution is the global optimum, the MIP usually terminates very quickly.

Results are given in Table 6. For comparison, we list the objective function values corresponding to the solution found by algorithm 2 again in column "NLP sol.". Column "upper bound" and "gap" give the upper bound we computed from MIP relaxation (2.3) (SCIP is used as the underlying MIP solver) and the difference percentage between upper bound and NLP solutions. Results show that from point of view of the objective function value our solution is very close to the optimal solution.

The computing time is also given in the same table. We find most of the time is spent on solving MIP (2.3) to get the upper bound of the solution. Although algorithm 2 needs to solve $T - 2$ NLP and 1 LP, the computing time still can be controlled in a few seconds, and relatively less than the phase of estimating the upper bound. And the total time of finding the NLP solution and estimating the upper bound is less than finding the same accurate solution by using the branch-and-bound algorithm. So combining the branch-and-bound algorithm with the NLP solver and our upper bound MIP relaxation will be a better choice for the open-pit mine production scheduling problem.

3 Conclusions

We discussed in detail how to solve the NLP raised when solving the multi-period single stockpile open-pit mine production scheduling problem. Several state-of-the-art NLP solvers are used. A simple iterative scheme is proposed as a heuristic algorithm for finding the global solution. A piecewise linearization technique to estimate the quality of the solutions is derived at the same time. Numerical results show that the solutions produced are very close to the proven global upper bounds.

References

[1] T. Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007.

[2] R. Byrd, J. Nocedal, and R. Waltz. KNITRO: An integrated package for nonlinear optimization. In *Large-Scale Nonlinear Optimization*, pages 35–59. Springer-Verlag, 2006.

[3] A. R. Conn, N. I. M. Gould, and P. L. Toint. *LANCELOT: a Fortran Package for Large-Scale Nonlinear Optimization (Release A)*, volume 17 of *Springer Series in Computational Mathematics.* Springer Verlag, Heidelberg, New York, 1992.

[4] E. Dolan. The NEOS server 4.0 administrative guide. Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, May 2001.

[5] R. Fletcher and S. Leyffer. *User manual for filterSQP*. University of Dundee, March 1999.

[6] C. Fricke. *Applications of Integer Programming in Open Pit Mining.* PhD thesis, University of Melbourne, Aug. 2006.

[7] P. E. Gill, W. Murray, and M. Saunders. User's guide for SNOPT (version 5.3). Technical report, Department of Mathematics, University of California, 1997.

[8] W. Gropp and J. Moré. Optimization environments and the NEOS server. In M. D. Buhmann and A. Iserles, editors, *Approximation Theory and Optimization*, pages 168–172. Cambridge University Press, 1997.

[9] M. M. J. Czyzyk and J. Moré. The NEOS server. *IEEE Journal on Computational Science and Engineering*, 5:68–75, 1998.

[10] M. Kocvara and M. Stingl. PENNON: A code for convex nonlinear and semidefinite programming. *Optimization Methods and Software*, 18(3):317–333, 2003.

[11] B. A. Murtagh and M. Saunders. MINOS 5.5 user's guide. Technical report, Department of Operations Research, Stanford University, Stanford, CA, USA, 1998.

[12] C. F. Natashia Boland and G. Froyland. A strengthened formulation for the open pit mine production scheduling problem. Technical Report 1624, Optimization Online, Mar. 2007. http://www.optimization-online.org/DB_FILE/2007/03/1624.pdf.

[13] G. F. Natashia Boland, Irina Dumitrescu and A. M. Gleixner. LP-based disaggregation approaches to solving the open pit mining production scheduling problem with block processing selectivity. *Computers & Operations Research*, 36:1064–1089, 2009.

[14] D. M. G. R, Fourer and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press/Wadsworth Publishing Company, Belmont, CA, 1993.

[15] R. J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 11:451–484, 1999.

[16] A. Wächter. *Introduction to* IPOPT*: A tutorial for downloading, installing, and using IPOPT*, November 2006.

Solver	Solution info.
Aggregated Model	
filterQP	Nonlinear constraints locally infeasible
IPOPT	Maximum number of iteration exceeded
Lancelot	Too many iterations
LOQO	Iteration limit
MINOS	Unbounded(or badly scaled) problem
PENNON	No progress
SNOPT	Cannot satisfy nonlinear constraints
KNITRO	Iteration limit reached
Extended Warehouse Model	
filterQP	Nonlinear constraints locally infeasible
IPOPT	Restoration Phase Failed.
Lancelot	Too many iterations
LOQO	Iteration limit
MINOS	693879181.5
PENNON	No progress
SNOPT	Requested accuracy could not be achieved.
KNITRO	691868010.2
Warehouse Model	
filterQP	Nonlinear constraints locally infeasible
IPOPT	Restoration Phase Failed
Lancelot	Too many iterations
LOQO	Iteration limit
MINOS	The current point cannot be improved
PENNON	No progress
SNOPT	Cannot satisfy nonlinear constraints
KNITRO	Iteration limit reached

Table 1: Results of different solvers for marvin data without preprocessing

Solver	Solution info.
Aggregated Model	
filterQP	Nonlinear constraints locally infeasible
IPOPT	Restoration Phase Failed.
Lancelot	Too many iterations
LOQO	Iteration limit
MINOS	Objective has not changed
PENNON	No progress
SNOPT	Cannot satisfy nonlinear constraints
KNITRO	Iteration limit reached
Extended Warehouse Model	
filterQP	Nonlinear constraints locally infeasible
IPOPT	Maximum Number of Iterations Exceeded.
Lancelot	Too many iterations
LOQO	Iteration limit
MINOS	693290432.9
PENNON	No progress
SNOPT	693290291.6
KNITRO	Iteration limit reached
Warehouse Model	
filterQP	Nonlinear constraints locally infeasible
IPOPT	687684346.8
Lancelot	Too many iterations
LOQO	Iteration limit
MINOS	671547237.4
PENNON	No progress
SNOPT	Cannot satisfy nonlinear constraints
KNITRO	Iteration limit reached

Table 2: Results of different solvers for marvin2 data without preprocessing

Solver	Solution info.
Aggregated Model	
filterQP	Nonlinear constraints locally infeasible
IPOPT	46325056.7 (solved to acceptable level)
Lancelot	Too many iterations
LOQO	Iteration limit
MINOS	Too many major iterations
PENNON	Iteration limit
SNOPT	Cannot satisfy nonlinear constraints
KNITRO	Iteration limit reached
Extended Warehouse Model	
filterQP	48834876.98
IPOPT	48835025.54
Lancelot	Too many iterations
LOQO	Iteration limit
MINOS	48834869.22
PENNON	Iteration limit
SNOPT	48834867.45
KNITRO	48833251.19
Warehouse Model	
filterQP	48790471.32
IPOPT	48790471.76
Lancelot	Too many iterations
LOQO	Iteration limit
MINOS	48706876.16
PENNON	No progress
SNOPT	48696006.39
KNITRO	48702059.05

Table 3: Results of different solvers for ob25 data without preprocessing

Solver	Solution info.		
	marvin	marvin2	ob25
		Aggregated Model	
filterQP	671261921.2	675214818.9	46325056.1
IPOPT	671261927	675214827.3	46325055.51
Lancelot	671261966.4	Step too small	46042626.54
LOQO	Iteration limit	Iteration limit	Iteration limit
MINOS	671261921.2	675214819.3	46325056.1
PENNON	671261921.3	Line search fail.	46325056.08
SNOPT	671261923.7	675215259.7	46325056.1
KNITRO	671261892	675214788.1	46325050.19
		Extended Warehouse Model	
filterQP	694254443.5	694117682.5	48834998.46
IPOPT	694472431.7	694117688.2	48835024.33
Lancelot	Step too small	694151532.3	Too many iter.
LOQO	694459569.5	694096130.3	Iteration limit
MINOS	694254455.5	694117682.5	Too many iter.
PENNON	694472426	694117682.3	No progress
SNOPT	694255806.1	694117682.5	48842209.5
KNITRO	694472397.2	694117653.9	48835017.81
		Warehouse Model	
filterQP	689050448.5	687678500.4	48790471.5
IPOPT	689050453.1	687684345.6	48790470.63
Lancelot	689055911.9	687685030.2	Too many iter.
LOQO	688952262.8	Iteration limit	Iteration limit
MINOS	689050450.9	687678500.7	48790472.03
PENNON	689050448.5	687684339.9	48790471.34
SNOPT	689050475.8	687678502.2	48790487.03
KNITRO	689050418.4	687684310.9	48790465.46

Table 4: Results of different solvers with preprocessing

Data	OneByOne	AllInOne	Original
marvin data			
agg	671261921.1	671261921.2	5.53839e+08
ewh	694472426	694254443.5	6.93879e+08
wh	689050448.4	689050448.5	6.72883e+08
marvin2 data			
agg	675214818.7	675214818.9	5.47343e+08
ewh	694117682.5	694117682.5	6.93291e+08
wh	687678500.2	687678500.4	6.71547e+08
ob25 data			
agg	46042584.48	46325056.1	4.36723e+07
ewh	48835025.2	48834998.46	4.88349e+07
wh	48790471.32	48790471.5	4.86963e+07

Table 5: Comparison of different algorithms

	Alg.A		estimated bound		
Problem	objective	t[s]	upper bound	t[s]	gap [%]
marvin data					
agg	6.71262e+08	32	6.73807e+08	1017	0.4
ewh	6.94472e+08	22	6.96649e+08	325	0.3
wh	6.89050e+08	23	6.91382e+08	454	0.3
marvin2 data					
agg	6.75215e+08	21	6.76789e+08	1367	0.2
ewh	6.94118e+08	22	6.96080e+08	412	0.3
wh	6.87679e+08	19	6.90073e+08	377	0.3
ob25					
agg	4.60426e+07	42	4.65771e+07	3770	1.1
ewh	4.88350e+07	58	4.89722e+07	181	0.3
wh	4.87905e+07	23	4.89737e+07	758	0.4

Table 6: Computational results for estimating the upper bound

A Implementation

In this appendix, we explain how to handle polynomial constraints in SCIP, which is the crucial step for implementing the above discussed algorithms.

SCIP, which is implemented in the C programming languages, is currently one of the fastest non-commercial MIP solvers available. It is also a framework for Constraint Integer Programming (CIP) and branch-cut-and-price. It allows total control of the solution process and the access to detailed information down to the guts of the solver. A detailed description of SCIP can be found in [1] and web site `http://scip.zib.de/`.

The standard distribution of SCIP provides all functionalities necessary to solve constraint and integer linear programs. Via its programming interface, however, SCIP can be easily extended by specialized handlers do deal with other constraint types as well. In the following, we will explain how to add a polynomial constraint to SCIP, which is the first nonlinear constraint handler in SCIP.

Generally speaking, all the information regarding the polynomial constraints is included in the polynomial constraint handler. We start by defining the data structures necessary to represent the polynomials. All the constraint handlers communicate with SCIP through standard callback functions. Therefore, we need to specify these callback functions in the polynomial constraint handler. The main task of the polynomial constraint handler is to improve the current solution by making the polynomial constraints feasible. To do this a nonlinear programming solver is needed. We choose the open source software IPOPT as the underlying NLP solver and explain how to interface SCIP with IPOPT.

A.1 Data Structures for Polynomial Constraints

A polynomial is the sum of several monomials. A monomial is composed of the coefficient, the variables and their corresponding powers. We use the following code for the data structure of monomials and polynomials:

```
typedef struct MonomialTag {
    int nuses;
    int nvars;
    SCIP_VAR ** vars;
    SCIP_Real * power;
    SCIP_Real coefficient;
} Monomial;

typedef struct PolynomialTag {
    int nuses;
    int nMonomials;
    Monomial ** monomials;
```

```
} Polynomial;
```

These two data structures provide us a convenient way to represent polynomials. For example, an instance of monomial structure with member variables `nvars` $= n$, `coefficient` $= c$ and `power` $= \{a_1, \cdots, a_n\}$ is $cx_1^{a_1} \cdots x_n^{a_n}$.

Since the constraint handlers in SCIP encapsulate all the data needed to represent the constraint, SCIP itself does not need to have any knowledge about the particular data structures used. Only the constraint handler has the information needed to describe the nonlinear constraints. Tasks that need detailed knowledge about the constraints, such as feasibility checking, are invoked by SCIP via callback functions through the constraint handler itself. These callback functions have to be implemented for the polynomial constraint handler. Details can be found in the SCIP documentation.

A.2 Functions for Polynomial Constraints

Functions for Monomials

The following six functions are implemented for monomials, to create, free, capture, release and evaluate the given monomial.

```
SCIP_RETCODE monomialCreate( SCIP*       scip,
                             Monomial**  monomials,
                             int         nvars,
                             SCIP_VAR**  vars,
                             SCIP_Real*  power,
                             SCIP_Real   coefficient )
```

Creates a monomial when the coefficient and power for each variable are given.

```
SCIP_RETCODE monomialFree( SCIP*      scip,
                           Monomial** monomials )
```

Frees the given monomial and sets the corresponding monomial pointer to `NULL`.

```
SCIP_Bool is_monomial_valid( Monomial *monomial )
```

Validates that the power of each variable in the monomial is non-zero. Returns `TRUE` if the given monomial is valid and `FALSE` otherwise.

```
void captureMonomial( SCIP* scip, Monomial*  monomial )
```

```
void releaseMonomial( SCIP* scip, Monomial** monomial )
```

Captures and releases the monomial by increasing and decreasing the reference counter.

```
SCIP_Real evaluateMonomial( SCIP*     scip,
                            SCIP_SOL* sol,
                            Monomial* monomial )
```

Evaluates `monomial` at the point `sol`.

Functions for Polynomials

Analogous to the operations for monomials, we implemented the following six functions for polynomials: `polynomialCreate`, `polynomialFree`, `is_polynomial_valid`, `capturePolynomial`, `releasePolynomial`, and `evaluatePolynomial`.

A.3 Constructing and Solving the Nonlinear Subproblem

The polynomial constraint handler does not contain any information about the objective function and any linear constraints. When creating the nonlinear subproblem, we access these information via the SCIP functions `SCIPgetLPRowsData`, `SCIPgetLPColsData`, `SCIPcolGetObj`, `SCIProwGetRhs`, `SCIProwGetLhs` and `SCIProwGetNLPNonz`. To make sure that the data we get via these functions is correct and complete, our polynomial constraint handler should be the last one to be checked by SCIP. For this, the macro property `CONSHDLR_CHECKPRIORITY` and `CONSHDLR_ENFOPRIORITY` should be set to the smallest value.

The data for the nonlinear polynomial constraints are incrementally obtained by the SCIP function `SCIPconsGetData`.

Creation and solution of the nonlinear subproblem are completely encapsulated in the function

```
SCIP_RETCODE ipoptSolve( SCIP*       scip,
                         SCIP_CONS** conss,
                         int         nconss,
                         SCIP_SOL*   sol,
                         SCIP_Real * contVarsVals ) .
```

This function first creates the nonlinear subproblem, then solves it (by calling the function `Callipopt` of IPOPT), and finally returns the solution computed by IPOPT and releases all temporary data structures. On the other hand, function `ipoptSolve` is called by function

```
SCIP_RETCODE improveSolByIpopt( SCIP *          scip,
                                SCIP_CONSHDLR * conshdlr,
                                SCIP_CONS **    conss,
                                int             nconss,
                                SCIP_SOL*       sol ) ,
```

which tries to improve the given solution `sol` by applying the nonlinear solver IPOPT and, if successful, returns the improved solution back to SCIP via the parameter `sol`.

To interface with IPOPT, the data of the created nonlinear subproblem is encapsulated in the internal data structure `NLP`:

```
typedef struct NLPTag {
   int nvars;
   int nbinvars;
   int nintvars;
   int nimplvars;
   int ncontvars;
   int nactivevars;
   int nnonactivevars;
   int nfixed;
   int naggr;
   int nmultaggr;
   int nnegation;

   int m_LP;
   int m_NLP;

   int * nnonz;
   int ** jCols;
   SCIP_Real ** values;

   PolynomialIpopt** polynomials;

   SCIP_Real * lhs;
   SCIP_Real * rhs;
} NLP;
```

The data type `PolynomialIpopt` used in this data structure is similar to the structure `Polynomial` discussed in SectionA.1 and will be explained in Section A.4.

A.4 The IPOPT Interface

To pass the nonlinear subproblem to IPOPT, we implemented all call-back functions that are necessary for the problem representation in the C–language programming interface of IPOPT. These functions are `eval_f`, `eval_grad_f`, `eval_g`, `eval_jac_g` and `eval_h`, which are the evaluation of the objective function, objective function gradient, constraint itself, Jacobi matrix, and Hessian matrix values separately.

Then SCIP calls IPOPT via the function `Callipopt` to solve the current nonlinear subproblem.

The data of the nonlinear subproblem is passed to IPOPT using two extra data structures:

```
 typedef struct MonomialIpoptTag {
   int nvars;
   int * indicies;
   SCIP_Real * power;

   SCIP_Real coefficient;
} MonomialIpopt;

typedef struct PolynomialIpoptTag {
   int nMonomials;
   MonomialIpopt ** monomials;
} PolynomialIpopt;
```

These data structures are similar to the structures `Monomial` and `Polynomial` in SCIP. They explicitly contain the indices of the variables in the monomials, which are hidden by the member `SCIP_VAR` in the corresponding SCIP data structures.

Generally speaking, it is not an easy task to implement the functions to evaluate the gradient and Hessian (IPOPT callback functions for problem representation). However, in our case, except for the polynomial constraints, all the other parts are linear. So this information can be formalized analytically and evaluated efficiently. The gradient of a linear function can be obtained directly from its coefficient vector and the Hessian is the zero matrix. Therefore, for the Jacobian of the constraints and the Hessian of the Lagrangian function, we only need to consider the polynomial constraints (`computePolynomialGradientElement`, `computeHessPolynomial` in our code). As a polynomial is the sum of several monomials, the main job is to compute the gradient and Hessian of the monomials. These operations are implemented in through functions like `computeHessMonomial`, `computeMonomialGradientElement`, etc.

IPOPT was designed for optimizing large sparse nonlinear programs. Because of problem sparsity, the required matrices (like the constraints Jacobian or Lagrangian Hessian) are not stored as dense matrices, but rather in a sparse matrix format. IPOPT can be customized for a variety of matrix formats, the triplet format was chosen in our implementation. In triplet format only the nonzero entries are stored. The matrix is encoded in two integer arrays and one double array, all of which have length equal to the number of non-zeros in the matrix. (In the case of a symmetric matrix, only the lower triangle of the matrix is stored.) By defining `nnz` to be the number of stored non-zero, we define the three arrays as follows,

irow[nnz]
jcol[nnz]
A[nnz]
meaning that for any k in the range $0, \cdots$,nnz-1 the elements at row irow[k] and column jcol[k] have value A[k].

B AMPL files

B.1 Aggregated Model

```
# Model file for the mining problem - reduced variables
# parameter section begin
param N := 85;
param T := 17;
param BP := 20;
param BM := 60;
param CM := 900000;
param CP := 4000000;
param Price := 10380000;
param ratio  := 1.1;

# code section begin
param g_underline ;
param g_overline ;
param g_overline2;

let g_underline := -1;
let g_overline := -1;
let g_overline2 := -1;

set SI := 1 .. N;
set ST := 1 .. T;
set ST_2 := 2 .. T;
set ST_T := 1 .. T-1;

param threshold := 0.00000001;
param alpha_rock  {SI} >= 0;
param alpha_ore   {SI} >= 0;
param alpha_metal {SI} >= 0;
param first {SI} >= 0;
param last {SI} >= 0;
param delta {SI} ;

param init_f_m{SI,ST} >= 0;
param init_f_p{SI,ST} >= 0;
param init_f_s{SI,ST} >= 0;

param init_ore_o{ ST } >= 0;
param init_met_o{ ST } >= 0;
param init_ore_q{ ST } >= 0;
param init_met_q{ ST } >= 0;
```

```
param EPSILON ;
param LOWER_IND ;
let LOWER_IND := T-1;

# aux set define
set St{ k in ST } := { i in ST: i <= k };
set SPred{SI}; # Pred set initialized in data file

# window set
set SReduce_x_arc{ k in SI } := { i in ST: first[k] <= i and i <= last[k]};
set SReduce_x_arc_2{ k in SI } := { i in ST_2: first[k] <= i and i <= last[k]};

# index set for variable
set IND = {i in SI, t in SReduce_x_arc[i]};

var f_p{ i in SI, t in SReduce_x_arc[i] } >= 0, <= 1, := init_f_p[i,t];
var f_m{ i in SI, t in SReduce_x_arc[i] } >= 0, <= 1, := init_f_m[i,t];
var f_s{ i in SI, t in SReduce_x_arc[i] } >= 0, <= 1, := init_f_s[i,t];

var met_o{t in ST_2} >= 0, := init_met_o[t];
var ore_o{t in ST_2} >= 0, := init_ore_o[t];

var met_q{t in ST_T} >= 0, := init_met_q[t];
var ore_q{t in ST_T} >= 0, := init_ore_q[t];

param x{SI,ST} binary;

# objective
maximize NPV:
sum { i in SI, t in SReduce_x_arc[i] } ratio**(-t) * ( ( Price * alpha_metal[i] - CP * alpha_ore[i]) * f_p[i,t] - CM * alpha_rock[i] * f_m[i,t] ) +
sum { t in ST_2 } ratio**(-t) * ( Price * met_o[t] - CP * ore_o[t] );

# constraints
# constraint on conservation of material
subject to C1{ i in SI, t in SReduce_x_arc[i] : delta[i] = 1 }:
   f_p[i,t] + f_s[i,t] <= f_m[i,t];
# constraint on safe mining 1-2
subject to C2{ i in SI}: sum{ t in ST: t in SReduce_x_arc[i] } f_m[i,t] = 1;

# constraint on mining capacity respected
subject to C4{ t in ST }:
   sum{ i in SI: t in SReduce_x_arc[i] } alpha_rock[i] * f_m[i,t] <= BM;
# constraint on ore in stockpile calculated in period 1
subject to C5: sum{ i in SI: 1 >= first[i]} alpha_ore[i] * f_s[i,1] = ore_q[1];
# constraint on metal in stockpile calculated in period 1
subject to C6: sum{ i in SI: 1 >= first[i]} alpha_metal[i] * f_s[i,1] = met_q[1];
# constraint ore on in stockpile calculated in other periods
subject to C7{ t in ST_2 : t in ST_T }:
   ore_q[t - 1] - ore_o[t]
   + sum{ i in SI: t in SReduce_x_arc[i]} alpha_ore[i] * f_s[i,t] = ore_q[t];
# constraint on metal in stockpile calculated in other periods
subject to C8{ t in ST_2 : t in ST_T }:
```

```
    met_q[t - 1] - met_o[t]
    + sum{ i in SI: t in SReduce_x_arc[i]} alpha_metal[i] * f_s[i,t] = met_q[t];

# constraint on we should not borrow things from the next period
subject to c73{t in ST_T: t< T-1}: ore_q[t] >= ore_o[t+1];
subject to c83{t in ST_T: t< T-1}: met_q[t] >= met_o[t+1];
subject to c74: ore_q[T-1] = ore_o[T];
subject to c84: met_q[T-1] = met_o[T];

# constraint on processing capacity is respected in period 1
subject to C9: sum{ i in SI: 1 >= first[i] } alpha_ore[i] * f_p[i,1] <= BP;
# constraint on processing capacity is respected in other periods
subject to C10{t in ST_2}:
    sum{ i in SI: t in SReduce_x_arc[i]} alpha_ore[i] * f_p[i,t] + ore_o[t] <= BP;
# constraint on material taken off stockpile at start of period has the same
# composition as material in stockpile at end of preceding period
subject to C11{ t in ST_2 : t in ST_T and t >= LOWER_IND }:
    ore_o[t] * met_q[t - 1] = met_o[t] * ore_q[t - 1];
# constraint on upper and lower bounds 1-2
subject to C12{ t in ST_T }: g_underline * ore_q[t] <= met_q[t];
subject to C13{ t in ST_T }: g_overline2 * ore_q[t] >= met_q[t];
subject to C14{ t in ST_2 }: g_underline * ore_o[t] <= met_o[t];
subject to C15{ t in ST_2 }: g_overline2 * ore_o[t] >= met_o[t];

subject to C16 { i in SI, t in SReduce_x_arc[i] : delta[i] = 0 }: f_p[i,t] = 0;
subject to C17 { i in SI, t in SReduce_x_arc[i] : delta[i] = 0 }: f_s[i,t] = 0;

# constraint on others
# specified in the declaration

data;

# calculate g_underline and g_overline
for { i in SI } {
    if alpha_ore[i] > 0 then {
        if g_underline = -1 or alpha_metal[i] / alpha_ore[i] < g_underline then {
            let g_underline := alpha_metal[i] / alpha_ore[i];
        }
        if alpha_metal[i] / alpha_ore[i] > g_overline then {
            let g_overline := alpha_metal[i] / alpha_ore[i];
        }
    }
    # initial indicator variable delta to 0
    let delta[i] := 0;
}

# tight the the under and over line
if CP / Price > g_underline then {
    let g_underline := CP / Price;
}

for { i in SI } {
    # re-assign indicator variable delta
    if alpha_ore[i] > 0 then {
```

```
        if alpha_metal[i] / alpha_ore[i] > g_underline then {
            let delta[i] := 1;
        }
    }

    # printf "delta[%3d]=%d \n",i, delta[i];
    # calculate the second large overline
    if alpha_ore[i] > 0 then {
        if alpha_metal[i] / alpha_ore[i]
            < g_overline and alpha_metal[i] / alpha_ore[i]
            > g_overline2 then {
            let g_overline2 := alpha_metal[i] / alpha_ore[i];
        }
    }
}

printf "g_underline = %8.6f , g_overline = %8.6f , g_overline2 = %8.6f \n",
    g_underline, g_overline, g_overline2 ;
```

B.2 Extended Warehouse Model

```
# Model file for the mining problem - reduced variables
# parameter section begin
param N := 85;
param T := 17;
param BP := 20;
param BM := 60;
param CM := 900000;
param CP := 4000000;
param Price := 10380000;
param ratio  := 1.1;

# code section begin
param g_underline ;
param g_overline ;
param g_overline2;

let g_underline := -1;
let g_overline := -1;
let g_overline2 := -1;

set SI := 1 .. N;
set ST := 1 .. T;
set ST_2 := 2 .. T;
set ST_T := 1 .. T-1;

param threshold := 0.00000001;
param alpha_rock  {SI} >= 0;
param alpha_ore   {SI} >= 0;
param alpha_metal {SI} >= 0;
param first {SI} >= 0;
param last {SI} >= 0;
param delta {SI} ;

```

```
param init_f_m{SI,ST} >= 0;
param init_f_p{SI,ST} >= 0;
param init_f_s{SI,ST} >= 0;

param init_ore_o{ ST } >= 0;
param init_met_o{ ST } >= 0;
param init_ore_q{ ST } >= 0;
param init_met_q{ ST } >= 0;

param EPSILON ;
param LOWER_IND ;
let LOWER_IND := T-1;

# aux set define
set St{ k in ST } := { i in ST: i <= k };
set SPred{SI}; # Pred set initialized in data file

# window set
set SReduce_x_arc{ k in SI } := { i in ST: first[k] <= i and i <= last[k]};
set SReduce_x_arc_2{ k in SI } := { i in ST_2: first[k] <= i and i <= last[k]};

# index set for variable
set IND = {i in SI, t in SReduce_x_arc[i]};

var f_p{ i in SI, t in SReduce_x_arc[i] } >= 0, <= 1, := init_f_p[i,t];
var f_m{ i in SI, t in SReduce_x_arc[i] } >= 0, <= 1, := init_f_m[i,t];
var f_s{ i in SI, t in SReduce_x_arc[i] } >= 0, <= 1, := init_f_s[i,t];

var met_o{t in ST_2} >= 0, := init_met_o[t];
var ore_o{t in ST_2} >= 0, := init_ore_o[t];

var met_q{t in ST_T} >= 0, := init_met_q[t];
var ore_q{t in ST_T} >= 0, := init_ore_q[t];

param x{SI,ST} binary;

# objective
maximize NPV:
sum { i in SI, t in SReduce_x_arc[i] } ratio**(-t) * ( ( Price * alpha_metal[i] -
CP * alpha_ore[i]) * f_p[i,t] - CM * alpha_rock[i] * f_m[i,t] ) +
sum { t in ST_2 } ratio**(-t) * ( Price * met_o[t] - CP * ore_o[t] );

# constraints
# constraint on conservation of material
subject to C1{ i in SI, t in SReduce_x_arc[i] :
   delta[i] = 1 }: f_p[i,t] + f_s[i,t] <= f_m[i,t];
# constraint on safe mining 1-2
subject to C2{ i in SI}: sum{ t in ST: t in SReduce_x_arc[i] } f_m[i,t] = 1;

# constraint on mining capacity respected
subject to C4{ t in ST }:
   sum{ i in SI: t in SReduce_x_arc[i] } alpha_rock[i] * f_m[i,t] <= BM;
# constraint on ore in stockpile calculated in period 1
subject to C5: sum{ i in SI: 1 >= first[i]} alpha_ore[i] * f_s[i,1] = ore_q[1];
```

```
# constraint on metal in stockpile calculated in period 1
subject to C6: sum{ i in SI: 1 >= first[i]} alpha_metal[i] * f_s[i,1] = met_q[1];
# constraint ore on in stockpile calculated in other periods
subject to C7{ t in ST_2 : t in ST_T }:
   ore_q[t - 1] - ore_o[t]
   + sum{ i in SI: t in SReduce_x_arc[i]} alpha_ore[i] * f_s[i,t] = ore_q[t];
# constraint on metal in stockpile calculated in other periods
subject to C8{ t in ST_2 : t in ST_T }:
   met_q[t - 1] - met_o[t]
   + sum{ i in SI: t in SReduce_x_arc[i]} alpha_metal[i] * f_s[i,t] = met_q[t];

# constraint on we should not borrow things from the next period
subject to c73{t in ST_T: t< T-1}: ore_q[t] >= ore_o[t+1];
subject to c83{t in ST_T: t< T-1}: met_q[t] >= met_o[t+1];
subject to c74: ore_q[T-1] = ore_o[T];
subject to c84: met_q[T-1] = met_o[T];

# constraint on processing capacity is respected in period 1
subject to C9: sum{ i in SI: 1 >= first[i] } alpha_ore[i] * f_p[i,1] <= BP;
# constraint on processing capacity is respected in other periods
subject to C10{t in ST_2}:
   sum{ i in SI: t in SReduce_x_arc[i]} alpha_ore[i] * f_p[i,t] + ore_o[t] <= BP;
# constraint on material taken off stockpile at start of period has the same
# composition as material in stockpile at end of preceding period
subject to C11{ t in ST_2 : t in ST_T and t >= LOWER_IND }:
   ore_o[t] * met_q[t - 1] = met_o[t] * ore_q[t - 1];
# constraint on upper and lower bounds 1-2
subject to C12{ t in ST_T }: g_underline * ore_q[t] <= met_q[t];
subject to C13{ t in ST_T }: g_overline2 * ore_q[t] >= met_q[t];
subject to C14{ t in ST_2 }: g_underline * ore_o[t] <= met_o[t];
subject to C15{ t in ST_2 }: g_overline2 * ore_o[t] >= met_o[t];

subject to C16 { i in SI, t in SReduce_x_arc[i] : delta[i] = 0 }: f_p[i,t] = 0;
subject to C17 { i in SI, t in SReduce_x_arc[i] : delta[i] = 0 }: f_s[i,t] = 0;

data;
```

B.3 Warehouse Model

```
# Model file for the mining problem - reduced variables
# parameter section begin
param N := 85;
param T := 17;
param BP := 20;
param BM := 60;
param CM := 900000;
param CP := 4000000;
param Price := 10380000;
param ratio  := 1.1;

# code section begin
param g_underline ;
param g_overline ;
param g_overline2;
```

```
let g_underline := -1;
let g_overline := -1;
let g_overline2 := -1;

set SI := 1 .. N;
set ST := 1 .. T;
set ST_2 := 2 .. T;
set ST_T := 1 .. T-1;

param threshold := 0.00000001;
param alpha_rock  {SI} >= 0;
param alpha_ore   {SI} >= 0;
param alpha_metal {SI} >= 0;
param first {SI} >= 0;
param last {SI} >= 0;
param delta {SI} ;

param init_f_m{SI,ST} >= 0;
param init_f_p{SI,ST} >= 0;
param init_f_s{SI,ST} >= 0;

param init_ore_o{ ST } >= 0;
param init_met_o{ ST } >= 0;
param init_ore_q{ ST } >= 0;
param init_met_q{ ST } >= 0;

param EPSILON ;
param LOWER_IND ;
let LOWER_IND := T-1;

# aux set define
set St{ k in ST } := { i in ST: i <= k };
set SPred{SI}; # Pred set initialized in data file

# window set
set SReduce_x_arc{ k in SI } := { i in ST: first[k] <= i and i <= last[k]};
set SReduce_x_arc_2{ k in SI } := { i in ST_2: first[k] <= i and i <= last[k]};

# index set for variable
set IND = {i in SI, t in SReduce_x_arc[i]};

var f_p{ i in SI, t in SReduce_x_arc[i] } >= 0, <= 1, := init_f_p[i,t];
var f_m{ i in SI, t in SReduce_x_arc[i] } >= 0, <= 1, := init_f_m[i,t];
var f_s{ i in SI, t in SReduce_x_arc[i] } >= 0, <= 1, := init_f_s[i,t];

var met_o{t in ST_2} >= 0, := init_met_o[t];
var ore_o{t in ST_2} >= 0, := init_ore_o[t];

var met_q{t in ST_T} >= 0, := init_met_q[t];
var ore_q{t in ST_T} >= 0, := init_ore_q[t];

param x{SI,ST} binary;
```

```
# objective
maximize NPV:
sum { i in SI, t in SReduce_x_arc[i] } ratio**(-t) * ( ( Price * alpha_metal[i] -
CP * alpha_ore[i]) * f_p[i,t] - CM * alpha_rock[i] * f_m[i,t] ) +
sum { t in ST_2 } ratio**(-t) * ( Price * met_o[t] - CP * ore_o[t] );

# constraints
# constraint on conservation of material
subject to C1{ i in SI, t in SReduce_x_arc[i] : delta[i] = 1 }:
   f_p[i,t] + f_s[i,t] <= f_m[i,t];
# constraint on safe mining 1-2
subject to C2{ i in SI}: sum{ t in ST: t in SReduce_x_arc[i] } f_m[i,t] = 1;

# constraint on mining capacity respected
subject to C4{ t in ST }:
   sum{ i in SI: t in SReduce_x_arc[i] } alpha_rock[i] * f_m[i,t] <= BM;
# constraint on ore in stockpile calculated in period 1
subject to C5: sum{ i in SI: 1 >= first[i]} alpha_ore[i] * f_s[i,1] = ore_q[1];
# constraint on metal in stockpile calculated in period 1
subject to C6: sum{ i in SI: 1 >= first[i]} alpha_metal[i] * f_s[i,1] = met_q[1];
# constraint ore on in stockpile calculated in other periods
subject to C7{ t in ST_2 : t in ST_T }:
   ore_q[t - 1] - ore_o[t]
   + sum{ i in SI: t in SReduce_x_arc[i]} alpha_ore[i] * f_s[i,t] = ore_q[t];
# constraint on metal in stockpile calculated in other periods
subject to C8{ t in ST_2 : t in ST_T }:
   met_q[t - 1] - met_o[t]
   + sum{ i in SI: t in SReduce_x_arc[i]} alpha_metal[i] * f_s[i,t] = met_q[t];

# constraint on we should not borrow things from the next period
subject to c73{t in ST_T: t< T-1}: ore_q[t] >= ore_o[t+1];
subject to c83{t in ST_T: t< T-1}: met_q[t] >= met_o[t+1];
subject to c74: ore_q[T-1] = ore_o[T];
subject to c84: met_q[T-1] = met_o[T];

# constraint on processing capacity is respected in period 1
subject to C9: sum{ i in SI: 1 >= first[i] } alpha_ore[i] * f_p[i,1] <= BP;
# constraint on processing capacity is respected in other periods
subject to C10{t in ST_2}:
   sum{ i in SI: t in SReduce_x_arc[i]} alpha_ore[i] * f_p[i,t] + ore_o[t] <= BP;
# constraint on material taken off stockpile at start of period has the same
# composition as material in stockpile at end of preceding period
subject to C11{ t in ST_2 : t in ST_T and t >= LOWER_IND }:
   ore_o[t] * met_q[t - 1] = met_o[t] * ore_q[t - 1];
# constraint on upper and lower bounds 1-2
subject to C12{ t in ST_T }: g_underline * ore_q[t] <= met_q[t];
subject to C13{ t in ST_T }: g_overline2 * ore_q[t] >= met_q[t];
subject to C14{ t in ST_2 }: g_underline * ore_o[t] <= met_o[t];
subject to C15{ t in ST_2 }: g_overline2 * ore_o[t] >= met_o[t];

subject to C16 { i in SI, t in SReduce_x_arc[i] : delta[i] = 0 }: f_p[i,t] = 0;
subject to C17 { i in SI, t in SReduce_x_arc[i] : delta[i] = 0 }: f_s[i,t] = 0;

data;
```