Exercise: Solving an IP formulation for Open Pit Mine Production Scheduling

In this exercise, we will implement an IP formulation for the *Open Pit Mine Production Scheduling Problem* (OPMPSP) using the SCIP callable library. We are given the following data:

- a number of time periods T with uniform discount factor δ per period,
- a block model with block indices in $\mathcal{N} = \{1, \ldots, N\}$,
- for each block $i \in \mathcal{N}$:
  - a set $\mathcal{P}(i)$ of immediate predecessors,
  - the amount of rock R_i contained in block i,
  - the cost m_i of mining block i,
  - the profit p_i from processing block i,
- and mining and processing capacities M and P per period.

To model mining (i.e. excavation) and processing decisions, we use variables

$x_{i,t} \in \{0,1\}$	equal to 1 iff block i is mined within periods $1, \ldots, t$, and
$z_{i,t} \in [0,1]$	as the fraction of block i sent for processing in period t.

Then a simple IP formulation for the OPMPSP reads

$$\max \sum_{t=1}^{T} \delta^{t-1} \sum_{i=1}^{N} \big[-m_i(x_{i,t} - x_{i,t-1}) + p_i z_{i,t} \big]$$

$$\begin{aligned}
s.t. \quad x_{i,t-1} &\leqslant x_{i,t} && \text{for } i \in \mathcal{N}, t = 2, \ldots, T, && (1)\\
x_{i,t} &\leqslant x_{k,t} && \text{for } i \in \mathcal{N}, k \in \mathcal{P}(i), t = 1, \ldots, T, && (2)\\
z_{i,t} &\leqslant x_{i,t} - x_{i,t-1} && \text{for } i \in \mathcal{N}, t = 1, \ldots, T, && (3)\\
\sum_{i=1}^{N} R_i(x_{i,t} - x_{i,t-1}) &\leqslant M && \text{for } t = 1, \ldots, T, && (4)\\
\sum_{i=1}^{N} R_i z_{i,t} &\leqslant P && \text{for } t = 1, \ldots, T. && (5)\\
x_{i,t} &\in \{0,1\} && \text{for } i \in \mathcal{N}, t = 1, \ldots, T, && (6)\\
z_{i,t} &\in [0,1] && \text{for } i \in \mathcal{N}, t = 1, \ldots, T, && (7)\\
x_{i,0} &= 0 && \text{for } i \in \mathcal{N}. && (8)
\end{aligned}$$

Getting started

(a) Extract the OPMPSP project (`tar xzfv COatWork-OPMPSP.tgz`). Amongst others, it contains:

`src/main.c` Main file, which reads the data files from the command line, initializes SCIP, calls a function to build the above IP formulation, calls SCIP to solve it, and displays the solution. *(nothing to do here)*

`src/opmpsp_data.h/c` Provides methods for reading data. *(nothing to do here, but look at the data structures `BLOCK` and `PIT`)*

`src/opmpsp_model.h/c` Contains method `createOpmpspModel`, which needs to be extended to build the above IP formulation within SCIP.

`src/opmpsp_presol.h/c` Contains an OPMPSP-specific presolver which may be extended to apply single block variable fixing and extend the clique table of SCIP.

`opmpsp.set` A SCIP settings file, which is automatically read by the command `SCIPreadParams(scip, "opmpsp.set")` in file `main.c`. This may be extended or changed, e.g. to set a time limit when dealing with larger instances.

`data/` A folder with small, randomly generated block model data of different sizes. Each instance is specified by two files: `randXXX.blocks` (containing the amount of rock and ore of each block) and `randXXX.arcs` (containing the immediate predecessors of each block). `XXX` indicates the number of blocks.

(b) In the folder `COatWork-OPMPSP/lib`, create a softlink to your SCIP directory, e.g. `ln -s /home/coatwork/software/ziboptsuite-1.2.0/scip-1.2.0 scip`

(c) In the folder `COatWork-OPMPSP`, try to compile the provided code: `make depend` and `make`.

1. Building the OPMPSP formulation

(a) Open the file `src/opmpsp_model.c`: The function `createOpmpspModel` is already partially provided. You need to fill in the gaps marked by `BEGIN_TODO...END_TODO`: create the variables with their upper and lower bounds and objective coefficients and add the constraints (1)–(8).
(*Note* that in C we are using indices starting from 0, so time periods are numbered 0 to $T-1$ and blocks are numbered 0 to $N-1$.)

(b) In the folder `COatWork-OPMPSP`, try to compile the code and run it on the smallest example: `bin/opmpsp data/rand035.blocks data/rand035.arcs`. The solving process can be interrupted by pressing `CTRL-C`. The optimal value is 7.300242.

2. Implementing a problem-specific presolver

The precedence graph $(\mathcal{N}, \{(i,k) \mid k \in \mathcal{P}(i)\})$ is usually given transitively reduced to save constraints of type (2). Suppose we additionally compute $\mathcal{C}(i)$, the full predecessor cone, i.e. the set containing i itself and all predecessors of block i in the transitive closure of the precedence graph. If the amount of rock in $\mathcal{C}(i)$ exceeds the mining capacity M, then block i cannot feasibly be excavated in the first time period, i.e. $x_{i,1}$ may be fixed to 0. In general, we may fix

$$x_{i,1}, \ldots, x_{i,t} = 0 \tag{9}$$

if $\sum_{k \in \mathcal{C}(i)} R_k > tM$ holds.

(a) Open the file `src/opmpsp_presol.c`, which already contains a partial implementation of the presolver. You need to extend the execution method of the presolver `presolExecOpmpsp`. The transitive closure of the precedence graph is already available in the presolver data: `presoldata->transpreds[i][k] == TRUE` iff $k \in \mathcal{C}(i)$. The variable $x_{i,t}$ can be accessed by `presoldata->xvars[i][t]`.

(b) Fill in the gaps marked by `BEGIN_TODO_1...END_TODO_1`: compute the value of $\sum_{k \in \mathcal{C}(i)} R_k$ and store it in `predrock[i]`.

(c) Fill in the gap marked by `BEGIN_TODO_2...END_TODO_2`: implement the single block variable fixing according to (9).

The idea of single block variable fixing can be extended to multiple blocks. As a special case, consider two blocks i, $j \in \mathcal{N}$, which are incomparable, i.e. $i \notin \mathcal{C}(j)$ and $j \notin \mathcal{C}(i)$. Again, if the amount of rock in $\mathcal{C}(i) \cup \mathcal{C}(j)$ exceeds the mining capacity M, then at most one of i and j can feasibly be excavated during the first time period. In general, if

$$\sum_{k \in \mathcal{C}(i) \cup \mathcal{C}(j)} R_k > tM$$

holds for time period t, then the inequality

$$x_{i,t} + x_{j,t} \leqslant 1 \tag{10}$$

is valid. One way of using this information in SCIP is to add the edge $(x_{i,t}, x_{j,t})$ to the so-called *clique graph*. The clique separator will then dynamically try to separate these inequalities if violated.

(d) Fill in the gap marked by `BEGIN_TODO_3...END_TODO_3` by generating the two block conflicts (10): For each pair of blocks $i < j$, check whether they are incomparable using `presoldata->transpreds[i][j]` and `presoldata->transpreds[j][i]`. If so, compute $\sum_{k \in \mathcal{C}(i) \cup \mathcal{C}(j)} R_k$, and for all t with (), add the edge $(x_{i,t}, x_{j,t})$ to the clique graph. Use the command `SCIPaddClique` as explained in the code.

3. Experiments

(a) Compile your project and test it on the provided OPMPSP instances. In the file opmpsp.set you can turn your presolving techniques off by adjusting the lines `presolving/opmpsp/varfix = ...` and `presolving/opmpsp/pckcliques = ...`. You may also have to choose the time and node limits suitably, because the primal-dual gap can typically not be closed completely.

(b) Pick one of the test instances in folder `data/` and analyse the computational behaviour with and without your presolving techniques:

- Do they help to increase the overall number of fixings and implications which SCIP is able to generate in presolving?
- How many clique cuts are generated? (Search the SCIP statistics.)
- How do they effect the dual bound at the root node and during solving?
- Experiment with the clique separator settings in `opmpsp.set`: By default, it is only called at the root node, but using `separating/clique/freq = ...` you can call it additionally during the branch-and-bound tree. Does this help to decrease the dual bound faster? Does it affect the primal bound?
- In general, which primal heuristics seem to be especially effective in finding or improving feasible solutions? (to decypher the abbreviations on the left of the SCIP output, type `disp heur` in a separate SCIP shell.)
- Are some of the constraints (1)–(8) redundant? Does leaving leaving them out improve or hinder the performance of SCIP?

A note of caution: Results on randomly generated data are not always conclusive for real-world instances.

Good luck!