Referee report on "Mixed Integer Programming Based Merge Search for Open Pit Block Scheduling"

This paper proposes a heuristic algorithm for solving the mine-planning problem of open pit block scheduling, which is an extension of the classical precedence constrained production scheduling problem.

Authors propose the use of a meta-heuristic called *Merge Search*, which is a kind of local-search heuristic that exploits parallelism and the efficient MIP solving of aggregated forms of the original problem. The method is similar to the CMSA heuristic [Blum et al, 2016], but its differences with this algorithm make it appropriate for large-scale PCPSP problem as the one appearing in mine planning problems.

Authors present a deep and thorough analysis using some of the Minelib's instances to test the algorithm, including a comparison with others state-of-art algorithms, sensitivity analysis on the parameters of the algorithms, among others. The proposed heuristic performs well, obtaining better results than other methods. However, it appears to be only suitable for the standard problem (positive knapsack constraints, no lower bounds) and for medium-sized problem (hundreds of thousands of blocks). Nevertheless, it is a contribution to the current literature on this problem.

Comments:

- Page 6: As described, the neighborhood search (Step 3 of the algorithm) is obtained by aggregating variables with value 0 and 1, and splitting the remaining randomly (a *merge search* with *m*=0). This is a key step of the algorithm. Have you tried different neighborhoods? There are a few classical ideas for open pit scheduling, specially on the local-search literature for this problem, that can be useful to improve the results of the algorithm.

- Page 11, Section 4.2: The relationship between Maximum Closure and a network flow problem is well known, and it appears in classical textbooks like Ahuja, Magnanti & Orlin, "Network Flows". Hence, I suggest to omit Appendix A and replace the cite from Hochbaum & Chen.

- Page 11: Maximum closure solver: The state-of-art algorithm for this problem is Hochbaum, D.: *The pseudoflow algorithm: a new algorithm for the maximum-flow problem*. Oper. Res. 56, 992–1009 (2008), which -in my experience- outperforms Boost implementations, at least on large instances of the problem. If not implemented, then at least it should be mentioned as a non-evalauted alternative.

- Page 12: Preprocessing: This idea appears in Munoz, et al : *A study of the Bienstock-Zuckerberg algorithm: applications in mining and resource constrained project scheduling*. Comput. Optim. Appl., 69(2), 501–534 (2018) under the name of *Path Contraction*.
- Page 13, Rounding Heuristic. The presented heuristic is a kind of TopoSort heuristics (Gershon, 93; Chicoisne et al, 2012). Please compare with current improvements of this heuristic, for example: Samavati et al, *A methodology for the large-scale multi-period precedence-constrained knapsack problem: an application in the mining industry*, International Journal of Production Economics 193 (2017) 12-20 or Samavati et al, *A new methodology for the open-pit mine production scheduling problem*, Omega 81 (2018) 169-182.
- Page 14: Could you include more details about B&B algorithm? Do you solve each node using BZ? How do you choose which variable to branch?
- Page 15: Computational results: I think that **BZ** and **B&B** are misleading names. The first one is a rounding heuristic using the LP relaxation of the problem (which can be obtained by BZ, Simplex or any other LP algorithm). The second one is a DFS over the B&B tree. May be **LP-Round** and **DFS-B&B**?
- Page 15: Why `mclaughlin` instances are too large for your proposed methodology? Which step is the one that require too much memory?
- Page 16, line 7: Can you explain the *very high communication overheads* of P-MS for `Newman1`? It is very counterintuitive that P-MS cannot solve the simpler problem of the dataset.
- Page 16, Table 3: I think that there are some errors with the computed Gap. For example, on `zuck_large` instance and Minelib, the lower bound is 5.73E7 and upper bound is 5.79E7, with a gap of 1.03%, not 0%. Same happens for B&B method.
- Page 16, Table 3: Why the LB of P-MS over `Zuck_large` is much worse than MS? Shouldn't P-MS always dominate MS?
- Page 17, Section 5.2.1: I disagree with comparing "average" gaps. This doesn't make sense at all. Please provide the results individually (Figure 6 but with all four instances individually) in order to validate your conclusion.
- Page 17. Why to exclude the other instances? In particular, for `zuck_large`, MS shows a better result than P-MS. It would be interesting to see how this varies with the time.
- Page 17, Section 5.2.2. Again, why you only consider these three instances? I assume that you need small instances in order to analyze the level 10000 of splitting. If that is the reason, it should be mentioned.

- Page 17: I'm confused on why do you use P-MS instead of MS for this analysis. I think that the effect of Random Splitting could be better shown running just MS.

- Page 20: Your conclusion on the number of levels is that there is a tradeoff between increasing diversity and memory requirements. Can you show this conclusion on a graph? For example, plotting the RSS memory of a run over time?

- Idea: Have you considered to use only the binary *x* variables of the problem? Fixing the value of *x* variables, it's easy to run one LP problem for each period optimizing only the *y* variables.

Typos and minor comments

- Page 2, Line -18: Typo: *hyrbid*
- Page 8: The formal wording for "cones" is *closure*.