On Solving the 2D Open-Pit Mining problem using Goldberg-Tarjan's Push-Relabel Max-Flow Algorithm

Garrenlus de Souza
Informatics Institute
Federal University Of Rio Grande do Sul - UFRGS
Porto Alegre, Brazil
gsouza@inf.ufrgs.br

I. Introduction

A. The Problem

This variation of the open pit mining problem is defined as follows. Suppose that the mining field can be represented as a 2D plot. This bi-dimensional plot is finite and can be fully captured by a square grid. In this grid each element represents the amount of return one could get by effectively exploring this supposed clump of dirt. Non negative values indicate a resulting surplus in resources after the exploration of such spot while values below zero indicate a loss. A clump can only be explored at once or not at all. The grid representation is used to capture the notion of depth, and in turn implies some constraints on what is involved in the task of exploring a clump of dirt. In this case, there is only one restriction, that subdivides itself into two. A clump with two neighbors horizontally can only be explored by first exploring its three neighbours in the adjacent level above.

Since our grid is finite, the other case applies to (horizontally) limitrophe clumps of dirt, for which the constraint differs by including only two nodes in the adjacent row above. Of course spots in the topmost row, do not have any constraint in this regard and can be explored directly.

B. Reduction

Instances of this problem can be mapped to max-flow/min-cut through a reduction. This reduction can be performed in linear time on the amount of lumps and is performed as follows. Take the number of clumps of dirt represented as cost/profit values in the grid as n. Now create an empty graph G and add 2 vertices to it. Take these two nodes and mark them as s and t. For every value c in the grid that is non-negative create a node and an arc with capacity c from s to this newly created node. For c values below zero, create a node and an arc with capacity $-c$(it becomes positive) from it to t. The "exploration dependencies" are captured by creating arcs with infinite capacity between nodes with an exploration constraint (from the dependant to the dependee).

This way we can bound the maximum amount of available resources by the capacities of the arcs that come from s and can use the max-flow from s to t to sever the arcs involved in explorations that would result in losses (by limiting the outcome to a break even situation, a trivial solution possible by leaving the entire "pit" unexplored). This approach would naturally present the solution for the partitioning problem through the min-cut that can be derived from the final state of the residual graph (after finding an optimal flow).

One can find the solution by performing a Breadth First Traversal in the residual graph from s, expanding the frontier only through arcs with remaining capacity greater than zero (including the synthetic arcs). When the traversal is completed, the set V of visited nodes can be used to find out which clumps of dirt were mined in the original grid. This can be done by identifying the incident arc that was created with each of the nodes in the set when the graph was built. One could then proceed to set as unexplored all the nodes that are not in V. This way the 2D open-pit mining problem can be solved by a reduction to the min-cut problem, solved by its dual instance on the realm of network max-flow.

II. Methodology

A. Environment

For the current study, the applied methodology consisted primarily into the analysis of running time statistics made over a reference implementation written in C++. The operating system is Ubuntu 20.04 LTS, running over a 32 GB RAM, 11th Gen Intel Core i7-1185G7 at 3.00GHz clock-speed system. All the code was compiled using g++ 9.4.0 (built for this very platform) through a CMake enabled build pipeline set to *"Release"* mode. The instances evaluated are available at [1] and the max-flow algorithm was previously tested and deemed correct in a previous assignment [1]. The timing fixtures are not native to C++ as it was observed that different instances presented representative differences even at process level as the grid size grown. All the data structures employed come from the *C++ STL* unless otherwise stated. The measurement was performed manually over each instance. Every presented solution consists in a "png" file, a conversion from the "pbm" file that can be generated (using solution and instance files) through the *opm* tool provided by the course instructor [2].

B. Implementation

The algorithm of choice was *Goldberg & Tarjan's*[1]. The graph was built as previously stated. After that a max-flow query was performed from s to t. Then the *BFT* procedure was performed returning a list of nodes reachable from the source (the explored nodes). As a means of validation an extra procedure was added to output (directly into the *stdout*) a DIMACS compliant instance of the max-flow reduction. The default behavior of the current implementation is to output a solution for the received instance as a line with two values, the width and depth of the pit and then the resulting grid in a tabular manner with the value 1 for explored clumps of dirt and 0 for the ones left aside. As by the definition of the assignment, the net-worth (revenue - losses) is reported at the standard error output.

1) Maximizing the active node selection: The employed version of *Goldberg & Tarjan's* algorithm used a *HL* data structure in order to optimize the selection of the next node to receive a discharge (e.g. the next node to have its excess flow exhausted in the terminology of push-relabel algorithms and such). One interesting aspect of this decision is the fact that some of the optimizations that make using an HL implementation faster [1] in some cases must be disabled as their use would render the current approach incorrect. One of these optimizations is the cap over d_v when it comes to building the pool of active nodes (eligible to undergo a discharge), that effectively keep active nodes out of the process once they reach $d_v = n$ as we can be sure they are not a part of the path between the source and the sink. Keeping this optimization activated would render the current approach ineffective at the final graph traversal stage.

C. Input

The instances of the problem were defined as follows by the course instructor and the value for each clump of dirt ranges from -128 to 127.

TABLE I

Name	Width	Depth
I1	10	10
I2	50	50
I3	200	100
I4	300	200
I5	600	200
I6	800	400

III. Results

In order to get a more straightforward understanding of the solution the "pbm" file generated using the *opm*[2] tool was fed into the *convert* utilitary as a mean to obtain ".png" files, shown below.

In this case the clumps painted in shades of blue and red were explored, while shades of green indicate that a region was left aside.

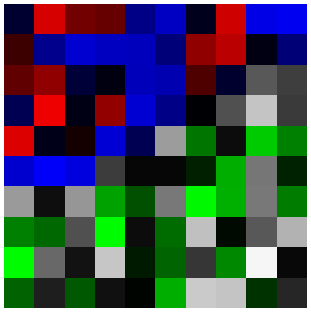

Fig. 1. The output of *opm* for the authors solution on I1.

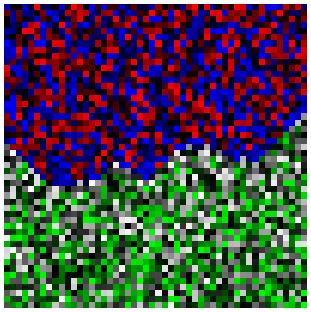

Fig. 2. The output of *opm* for the authors solution on I2.

IV. Discussion

One could point out by the previously presented pictures that the exploration of the pit was quite limited and never seemed to reach 50%. Though a value above the 50% mark was actually reached, it was just once can be seen in Table II.

TABLE II
GENERATED BY SPREAD-LATEX

Name	Time (s)	Area	Exploration ratio
I1	0.003	100	10.00%
I2	0.119	2500	51.64%
I3	1.469	20000	14.61%
I4	36.798	60000	32.98%
I5	55.589	120000	13.67%
I6	291.37	320000	6.25%

The time taken to compute the optimal "exploration cut" seemed to follow a monotonic (positive) growth pattern as

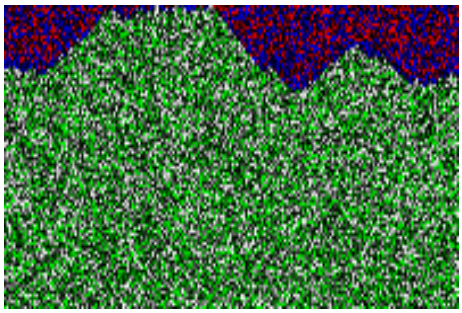

Fig. 3. The output of *opm* for the authors solution on I3.

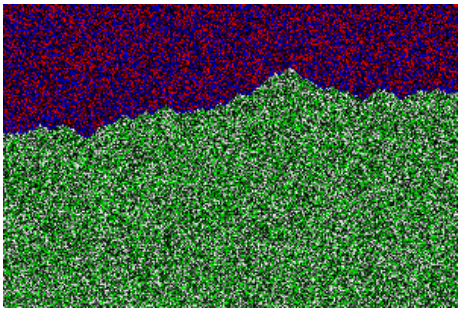

Fig. 4. The output of *opm* for the authors solution on I4.

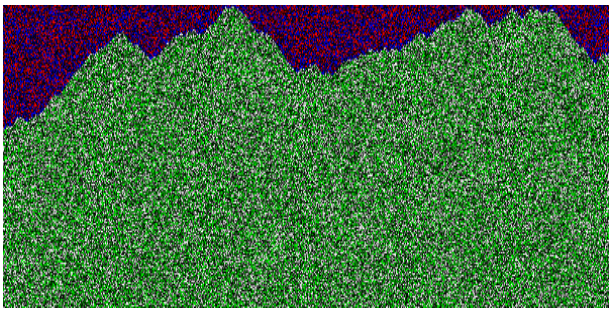

Fig. 5. The output of *opm* for the authors solution on I5.

the size of the instances grow. This goes along the rationale that the resulting graph that ends up undergoing the max-flow algorithm grows as the size of the grid grows (Fig 5).

The explored area coverage ranged from 6.25% to 51.64% while execution time ranged from 3ms up to 291.37s. All the implemented source code, as well as the results can be obtained directly at [1].

REFERENCES

[1] "Implementing goldberg & tarjan's push-relabel algorithm," https://github.com/GarrenSouza/inf05016-push_relabel, Accessed: 2023-2-2.

[2] "Inf 05016 - assignments," https://www.inf.ufrgs.br/~mrpritt/doku.php?id=inf05016:2022-2-trabalhos, Accessed: 2023-2-22.

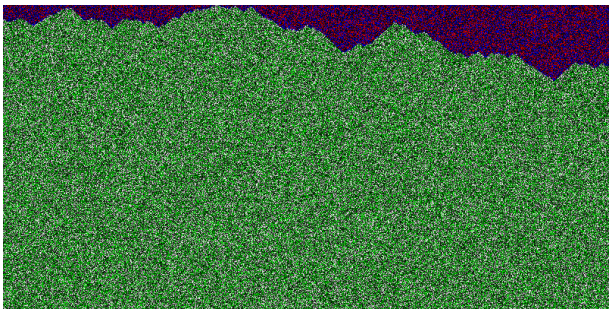

Fig. 6. The output of *opm* for the authors solution on I6.

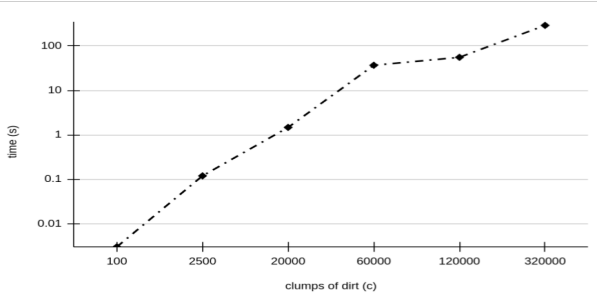

Fig. 7. The output of *opm* for the authors solution on I5.