Facility Location with BDDs: Status update 2

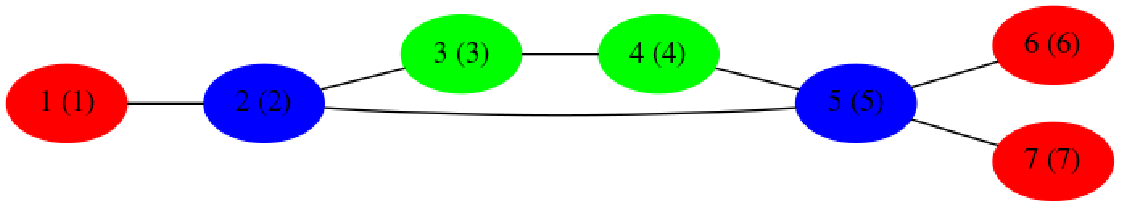
A. Bochkarev

1 Status

- we are dealing with the Facility location problem in the following edition: every facility can "cover" all its neighbors, we have to cover every customer at least once, no overlap costs; we have "colored" facilities + budget for number of locations per color.
- I have implemented constructing BDDs for (1) **cover** constraints, and (2) **color** constraints.
- with this problem formulation, intersection BDD is still very large, but now we have at least something to discuss, as the process depends on many factors.
- runtimes, to me, still do not look competitive with MIP – we might want to look into some specific instance types?

2 The problem.

First, let me re-introduce my "model" problem. Assume I have the following seven nodes (costs are in the parentheses) of graph G:



Color limits are as follows: red (5), blue (1), green (3). (So, the point is: I can't have two "central" blue nodes, ② and ⑤, at the same time – other limits are not binding.)

3 Plan of attack

- build **cover** BDD (encode the condition "every facility has to be covered at least once");
- build **color** BDD (encode the condition "respect color budgets");
- make them order-associated;
- build an intersection DD.
- formulate and solve the network flow out of the intersection DD.

All of these steps are implemented now.

4 Building the cover DD

- I keep track of a "state" – a boolean vector of length N (number of nodes in G) , denoting whether each node was covered or not. So, I have a "covering" *state* associated with every BDD node.
- so, "incorporating" node (of G) to the BDD is straightforward: I just update the states for all nodes adjacent to this "new" node, with the following caveat.
- if from some state there is no way I could cover at least one node (of G) further, I connect the respective BDD node to the “trash pipe” (a set of nodes with no path to **True** terminal).
- to keep track of such "opportunities further in the BDD", I basically keep the remaining *"freedoms"* of G nodes: node degree minus the number of its neighbors already present in the diagram above. When such number drops to zero, then there is no way the "state" of the node ("covered" vs. "not covered" can be changed further down the BDD).
- so, I "process" nodes one by one as follows. I just "incorporate" the node, and then all its neighbors, into the BDD. For example, "processing" node ① involves incorporating nodes ① and ②. Processing ② would involve incorporating ①, ②, ③, and ⑤, and so on.
- Again, after every node processing I update the node "freedoms".
- I take special care of the nodes which "freedoms" has just dropped to zero: the ones that are not covered will drive the respective BDD state (BDD node) to the “trash pipe”.
- The final piece of the puzzle is the order of nodes processing. I just pick the node with the smallest number of "freedoms" every time.
- edge weights are zeroes, except for "yes"-arcs for non-trash-sleeve nodes, where the weights correspond to location costs.

  The resulting diagram for the problem above presented in Figure 1.

5 Building the color DD

- building the **color** diagram is simple: I would need to pick an *order* for colors, and then, within each color the state is number of locations already "used". For example, if the budget for green is 3 locations, I might have states up to {`0`, `1`, `2`, `3`, and `(infeasible)`} in any “green” layer of the color diagram.
- of course, after every color block the layer width is reset to at most two (comprising state `0` and, perhaps, `(infeasible)`).
- now, ordering does matter, of course, and we have certain flexibility here: I can pick any order *within* each color, and I can arbitrarily shuffle *colors*. So, what I am doing is:
  - *within* each color order of G nodes correspond to their relative order in the cover diagram.

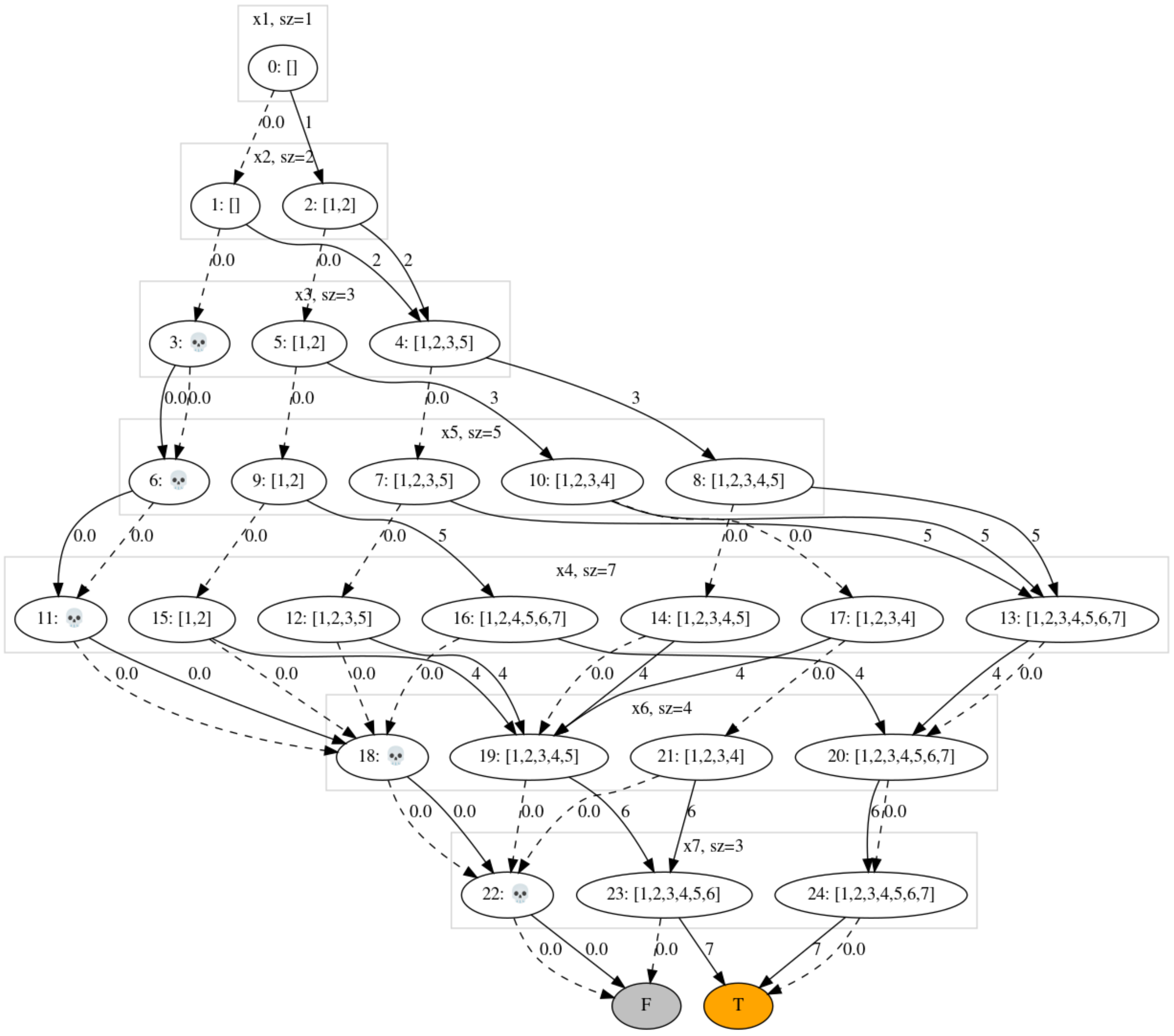


Figure 1: The **cover** diagram (numbers in `[brackets]` are nodes covered in the respective state; "trash sleeve" is denoted by skulls). I start with ①, adding layers for ① and ②. Note that ③ has 3 freedoms now (depending on ②, ③, and ④). The next node to process will be one with the minimal freedoms – either ⑥ or ⑦ (each with two). So ⑥ and ⑤ are added, and so on.

  - *between* colors... Well, I move colors around (as blocks) in a procedure **very** similar to a bubble sort.[1]

- in fact, I did not use any weights in this diagram (all are zeroes).

The resulting diagram for the problem above looks as presented in Figure 2.

6 The intersection DD.

The procedure is (conceptually) simple: a "state" is now defined by a pair of node IDs. For example, if I intersect diagrams D_1 and D_2, considering nodes $u \in D_1$ and $v \in D_2$, the intersection diagram will have the following two nodes in the next layer: $(1(u), 1(v)$ and $(0(u), 0(v))$ (where $1(a)$ denotes a head of one arc emanating from a, and $0(a)$ – respectively, for a zero arc).

So, we have the following statistics for the number of nodes for our particular example (excluding the two terminal nodes):

```
BEFORE the alignment:
cover size: 25, color size: 17
AFTER the alignment:
cover size: 26, color size: 18
intersection size: 38 nodes
```

An intersection DD, of course, allows to build a network flow problem (which is an LP, albeit in a large network) – neither the idea, nor the code are new.

7 Some results of numerical modeling.

7.1 Diagram sizes (table)

Let me provide a raw table for the diagram sizes in some random experiments – see Listing 1. To me, this table highlights the key problem I am having at the moment. I have also run some experiments to benchmark BDD-based methods with this "plain MIP": of course, given the intersection sizes, for $n = 30$ nodes in G I have the intersection-based thing 1–3 orders of magnitude slower than the plain the MIP.

8 Random instances generation

9 Discussion

- sorting variables in `color` diagram.
- random graph generation (limit node degree?)
- next node selection in the (cover) BDD generation procedure.

[1]oh, this is a strange story. I can "compare" any two colors C_1 and C_2 (relative to the target order of nodes in G), in the sense that I can say, what gives me more inversions with the cover diagram: $C_1 \prec C_2$ or $C_2 \prec C_1$. The only thing is: I am not sure the resulting relation is transitive... And so, perhaps, every 300th or 400th random example is off by 1–2 inversions from the optimum. Anyways, the way I have implemented it now allowed to cut no. of nodes and runtimes approx. 2–3 times, I think, compared to the trivial ordering (1, 2, 3, ...).

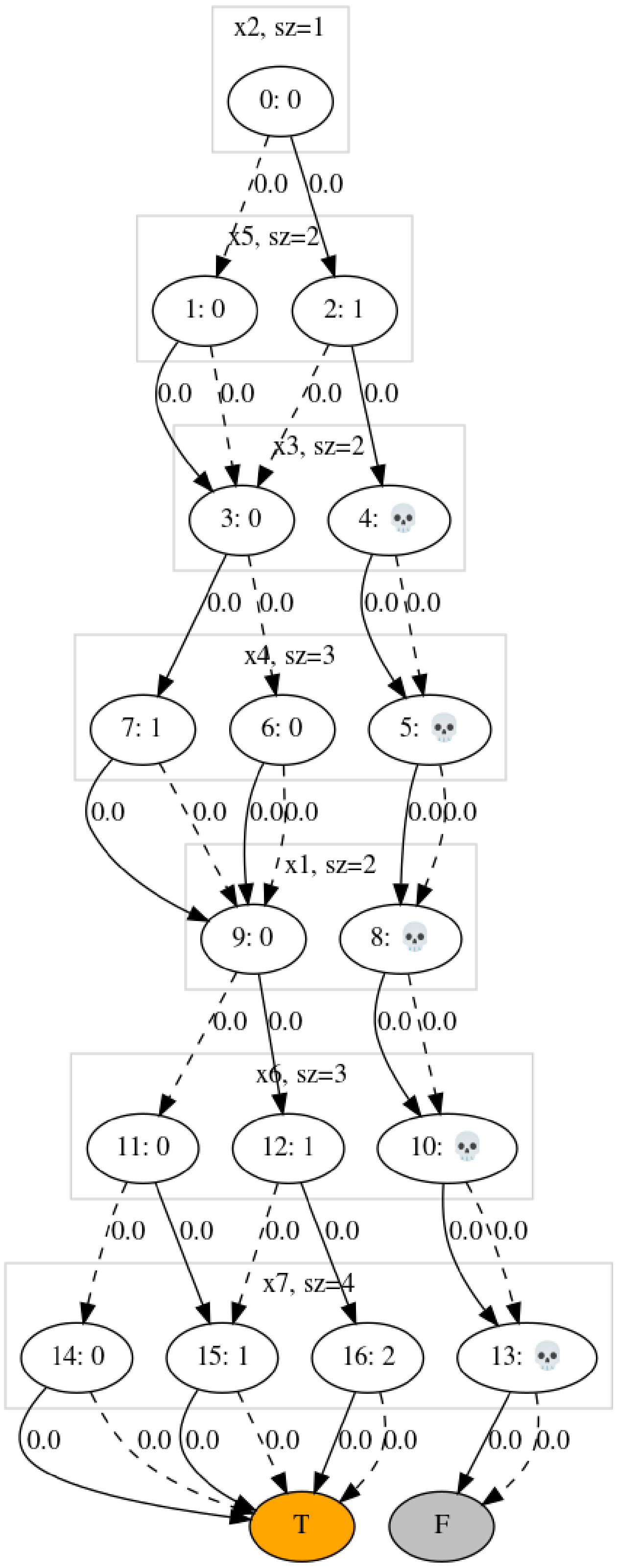


Figure 2: The **color** diagram. The order of colors is: blue (②, ⑤) – green (③, ④) – red (①, ⑥, ⑦).

```
Raw data: numerical experiment 1 (2020-04-02)

                                          |          Plain MIP              | Experiment
n |size(color)|size(cover)| Intersection  | No. of vars    | No. of constr. | time (sec.)
--+-----------+-----------+---------------+----------------+----------------+-------------
5 | 9         | 10        | 14            | 5              | 8              | 0.0
5 | 6         | 11        | 13            | 5              | 9              | 0.0
5 | 7         | 11        | 12            | 5              | 8              | 0.0
5 | 7         | 11        | 18            | 5              | 8              | 0.0
5 | 9         | 14        | 20            | 5              | 7              | 0.0
7 | 10        | 18        | 24            | 7              | 12             | 0.0
7 | 14        | 19        | 30            | 7              | 10             | 0.0
7 | 18        | 22        | 41            | 7              | 9              | 0.0
7 | 11        | 27        | 33            | 7              | 11             | 0.0
7 | 10        | 22        | 33            | 7              | 12             | 0.0
10| 32        | 71        | 130           | 10             | 12             | 0.0
10| 23        | 46        | 73            | 10             | 14             | 0.0
10| 27        | 60        | 128           | 10             | 14             | 0.0
10| 11        | 60        | 60            | 10             | 19             | 0.0
10| 31        | 43        | 121           | 10             | 13             | 0.0
15| 31        | 411       | 633           | 15             | 21             | 0.1
15| 70        | 256       | 966           | 15             | 17             | 0.1
15| 33        | 146       | 262           | 15             | 23             | 0.1
15| 50        | 108       | 445           | 15             | 19             | 0.1
15| 48        | 156       | 587           | 15             | 20             | 0.1
20| 47        | 607       | 930           | 20             | 28             | 0.1
20| 54        | 3115      | 4481          | 20             | 28             | 1.0
20| 65        | 1756      | 4498          | 20             | 25             | 0.9
20| 84        | 2270      | 3953          | 20             | 22             | 0.8
20| 67        | 1689      | 3531          | 20             | 25             | 0.9
25| 122       | 10400     | 22658         | 25             | 27             | 4.0
25| 119       | 10281     | 22248         | 25             | 29             | 2.8
25| 82        | 5234      | 13295         | 25             | 29             | 3.1
25| 78        | 3320      | 9282          | 25             | 31             | 1.7
25| 104       | 6934      | 14346         | 25             | 28             | 2.7
30| 86        | 21872     | 24613         | 30             | 36             | 9.8
30| 103       | 24867     | 46515         | 30             | 39             | 13.2
30| 96        | 29752     | 44876         | 30             | 38             | 13.1
30| 119       | 35965     | 53781         | 30             | 36             | 17.7
30| 120       | 17616     | 51990         | 30             | 37             | 14.1
--+-----------+-----------+---------------+----------------+----------------+-------------
```

Listing 1: Diagram sizes vs. number of variables and constraints in a plain MIP (depending on n – number of nodes in G, the original graph).