On special classes of instances for align-BDD

Alexey Bochkarev

May 23, 2022

In order to find a special class of instances, where `align-BDD` heuristic would give us a clear advantage, I tried two things.

- First, I looked into *typed UFLP* on 'cavemen' graph structure and checked if anything would change if the order of the type diagram would be randomized. It seems the situation changes slightly: indeed, I am able to find smaller diagrams more often than not. But for some reason this does not really yields benefit in runtimes, as compared to a simple `align-to-A` heuristic.
- Second, I tried to invent a *joint-UFLP*, and it seems this is where the heuristic does make sense. However, we'd need to discuss if this type of problem is natural enough.

  More details on everything is below.

1 UFLP with overlaps: general info

We have been discussing variations of uncapacitated facility location problem (UFLP). In a nutshell (and this is no different, conceptually, from what we have in the paper):

- We have M points of an undirected graph.
- I can *locate a facility* at each one, at cost c_i for the i-th point.
- If a facility is located at point i, it 'covers' all the points from its neighborhood, S_i.

New things:

- **Overlap costs**. That is, if a point j is covered a times, we bear cost (or profit) f_{ja}, with no particular assumptions on the structure of f_{ja} depending on a (i.e., no monotonicity / convexity / concavity).
- **Objective:** We do not demand all points to be covered, but we assume certain cost f_{j0} for not covering a point at all. The objective is to minimize the sum of costs, including location costs and overlap costs.
- **Special structure / 'cavemen':** We restricted our attention to a special type of instances, where points are split into clusters, with many edges within a cluster, but only a single edge between two clusters (and each cluster being connected to at most two other ones). An example of such instance is provided in Fig. 1. I call head and tail of an edge connecting two clusters a *connection point.*

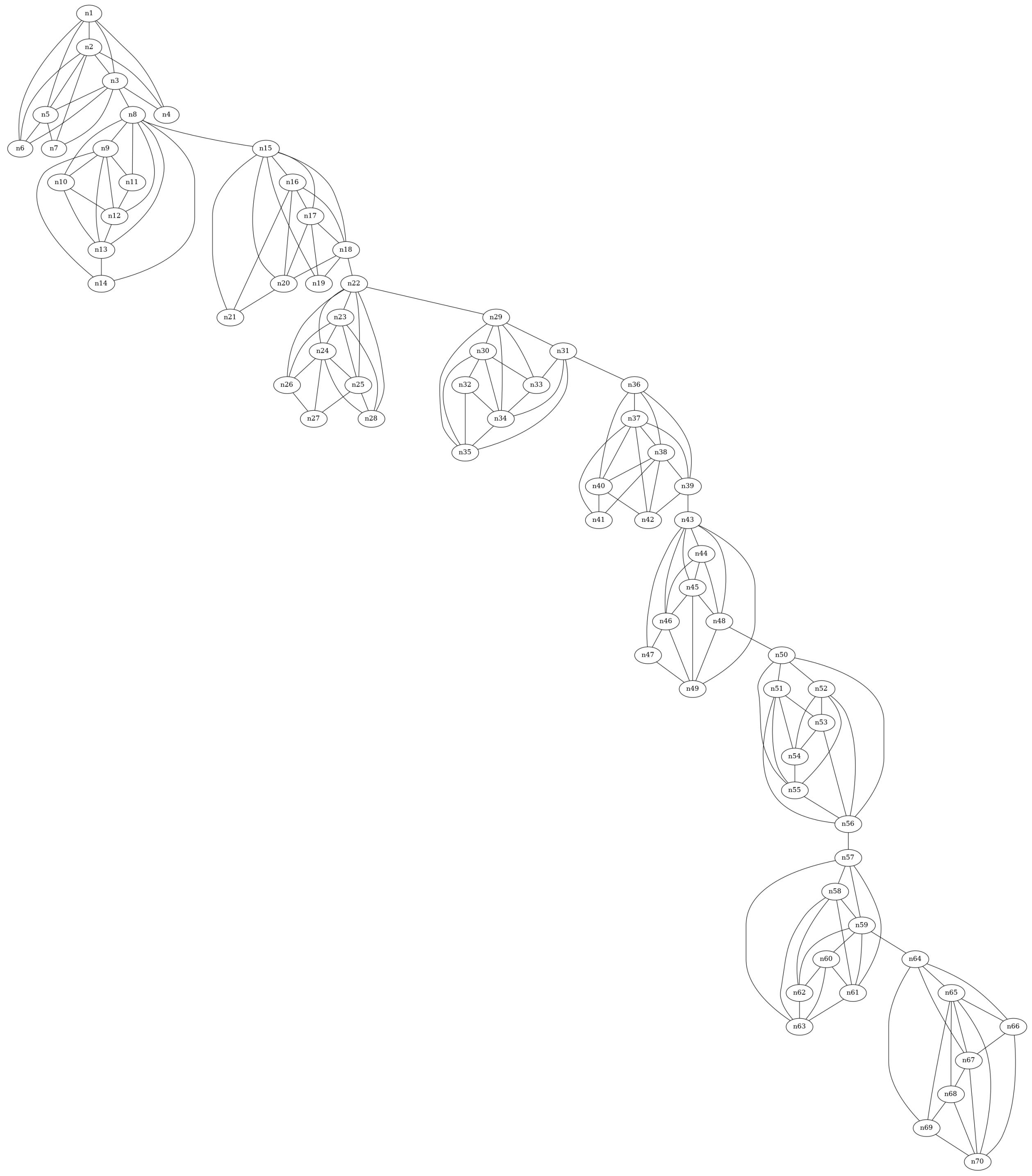


Figure 1: Instance example: special graph structure.

This special structure allowed us to design a tailored algorithm exploiting these 'caves.' The key idea is that as soon as I decide whether I locate a facility at four connection points adjacent to a cluster, the subproblem corresponding to this cluster becomes independent of the rest of the problem. This way I can process clusters one by one and build a BDD encoding the objective value depending on the choices regarding endpoints only, having width of at most eight nodes.

Such algorithm beats a naive MIP formulation (solved by Gurobi solver), since it exploits the special structure. Now, I'd need to introduce another sort of conditions, to illustrate our BDD-alignment heuristic.

First thing I tried was the introduction of *types* for connection points. That is, I assumed that each one has a type, and there is a (budget) constraint on the maximum number of points of each type I can locate a facility at. While the BDD-based approach was indeed faster than a naive MiP, it was not clear why use our heuristic, since aligning 'type' diagram to 'cover' diagram to build an intersection later would work just as well, and in many cases even faster than our simplified-problem-based heuristic. We wanted to find a situation where it would not be the case.

2 t-UFLP + randomized type diagram.

Our hypothesis was that we just happened to supply a fortunate initial ordering by building the type diagram. (So that revising this order to a more complex cover DD was relatively simple.) Therefore, I tried to destroy this simple structure to see if it was indeed the case. Viz, before aligning, I randomized the order of variables in the type diagram. The results in terms of the resulting intersection diagram sizes are presented in Fig. 2 and 3.

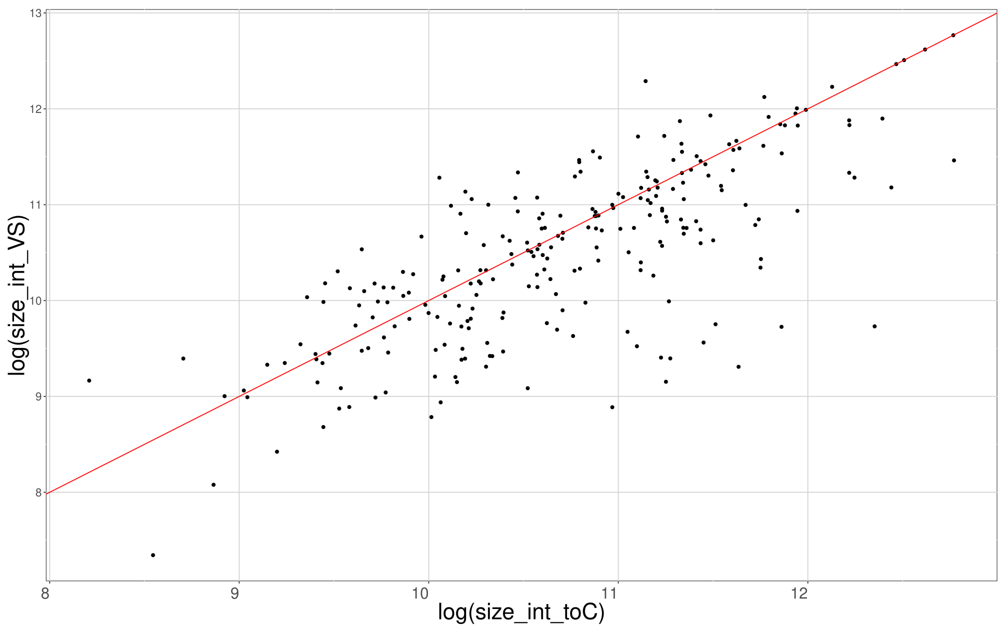


Figure 2: Intersection diagram sizes (nominal values), depending on the alignment heuristic used: `VS` for variable-sequence based, and `toC` for a simple alternative, "align-to-cover-DD" heuristic.

So, it seems the intersection DD size is smaller with our heuristic than it is for the simple `align-to-cover` approach more often than not. That is, on about 60% of my random instances (about 62% if I count the cases when the size is the same). In terms the histogram of relative intersection diagram sizes, see Fig. 3.

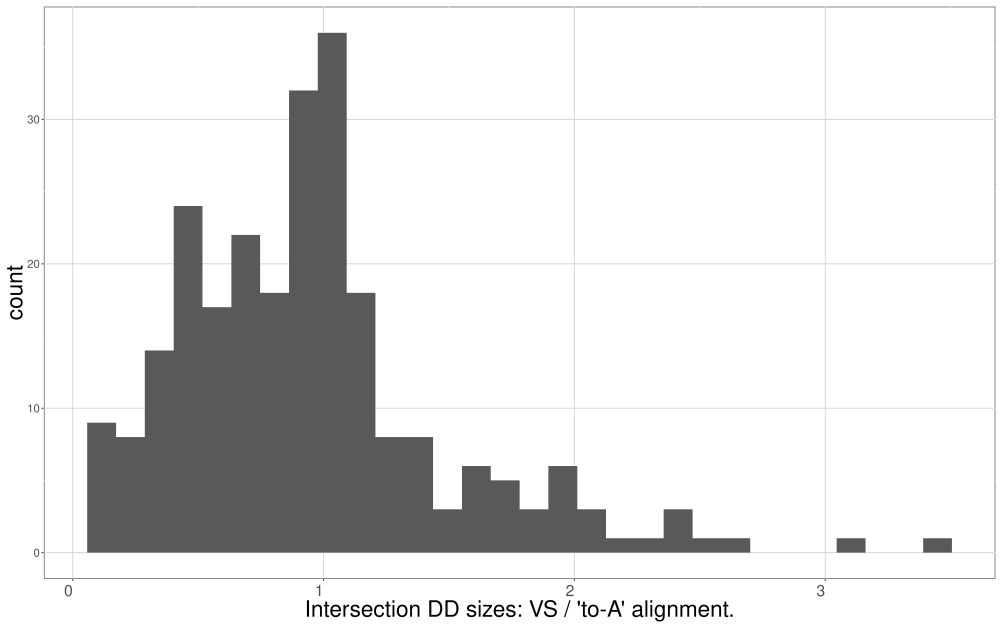


Figure 3: Intersection diagram sizes, VS-based heuristic relative to 'align-to'cover' alternative.

Interestingly, though, runtimes still suggest that a simple alignment heuristic is by far faster.

However, I think I was able to invent a class of problems where using the heuristic does make sense.

3 Another variation of 'joint-UFLP.'

Again, consider UFLP (without types) over this 'cavemen' type of graph with general overlap cost. Let me forget types altogether, but assume we have not one, but **two** such graphs, stitched together via the connection points in the following sense.

Both graphs possess the special 'cavemen' structure, and connection points coincide. That is, there is a one-to-one mapping between the connection points in G_1 and the ones in G_2. Moreover, locating a facility at a connection point in G_1 automatically implies locating the corresponding one in G_2 at no additional location cost. Otherwise, the two UFLP instances are independent. I think even the total number of nodes per cluster and for the graph in total can be different. Overlaps are also calculated for the two graphs independently, aside from the dependent location decisions.

My primary objective when designing this type of instances was to have a pair of **comparable** diagrams to align. I will further try to provide some intuition on what this problem might actually mean / how to interpret it. However, let's see first if it would work for the purposes of the illustration for our heuristic.

I generated random instances as follows:

1. Create the necessary number of nodes split into the given number of clusters (n clusters, some fixed number of points each)

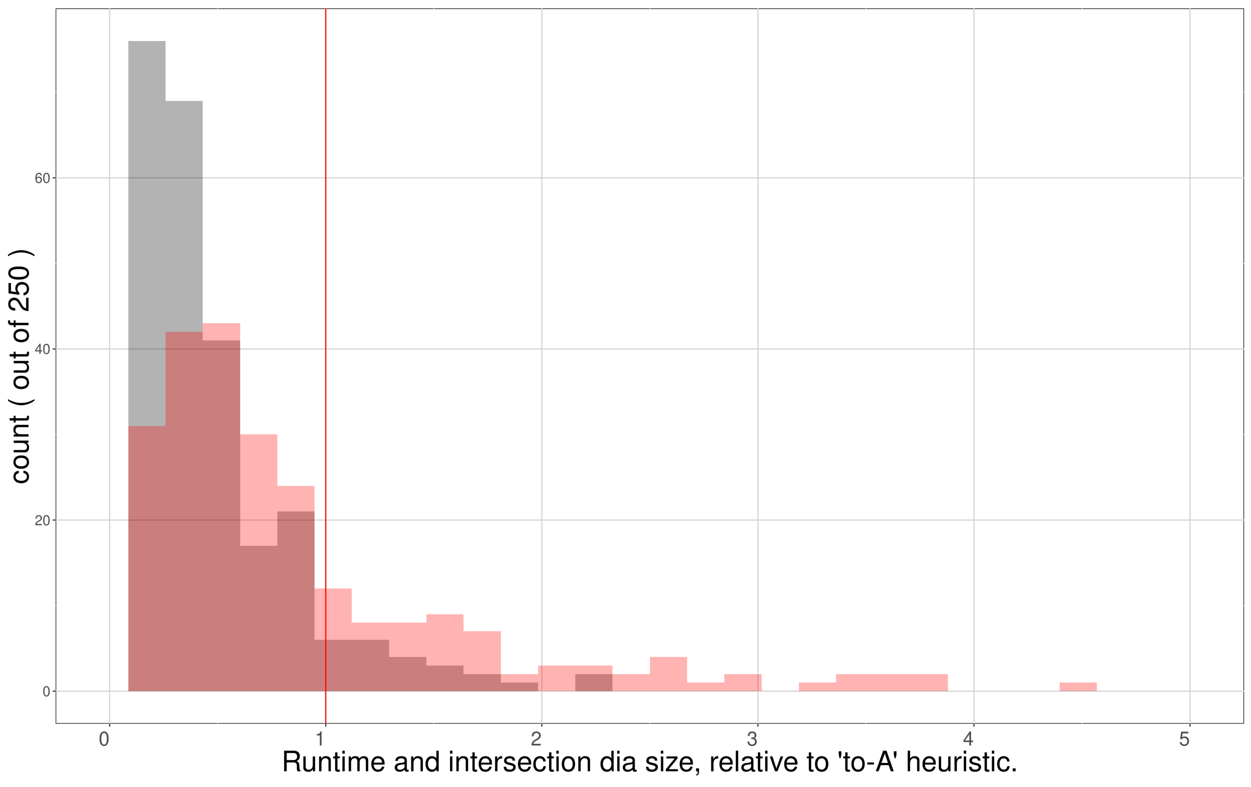
2. Make sure that each cluster is connected, and that the clusters are connected to each other as necessary.

3. Create two copies of this graph — these will give me G_1 and G_2.

4. Within each graph, add random edges within each cluster to reach the necessary ‘density’ (parameterized as the number of edges)

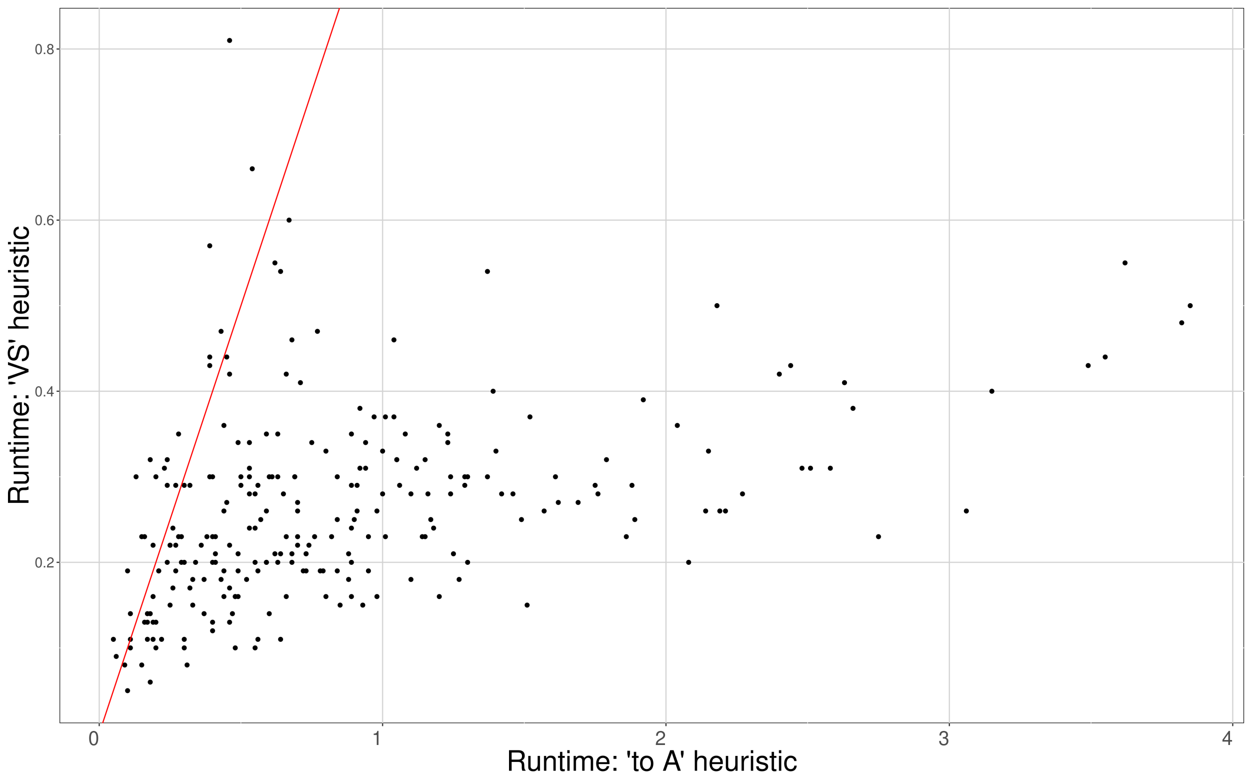
The procedure gives me G_1 and G_2, but there is no point in aligning them, because the sequence of connection points coincide. Therefore, I just rename the connection points in G_2 (by randomizing their order – I am actually doing this renaming already for a BDD, before the alignment.)

In this fashion, I generated 250 random instances and solved them using the BDD-based approach (naive MIP seems slower anyways), aligning the diagrams using the variable-sequence-based heuristic (VS) and a simple align-to-A heuristic (toA). The results are presented in Fig. 3 — that’s a histogram of relative intersection diagram sizes, our heuristic relative to the simple baseline (‘align-to-A’). Here, on 92% of the cases our heuristic performs better in terms of the sizes. Nominal/absolute runtimes and sizes are depicted in Fig.~4.

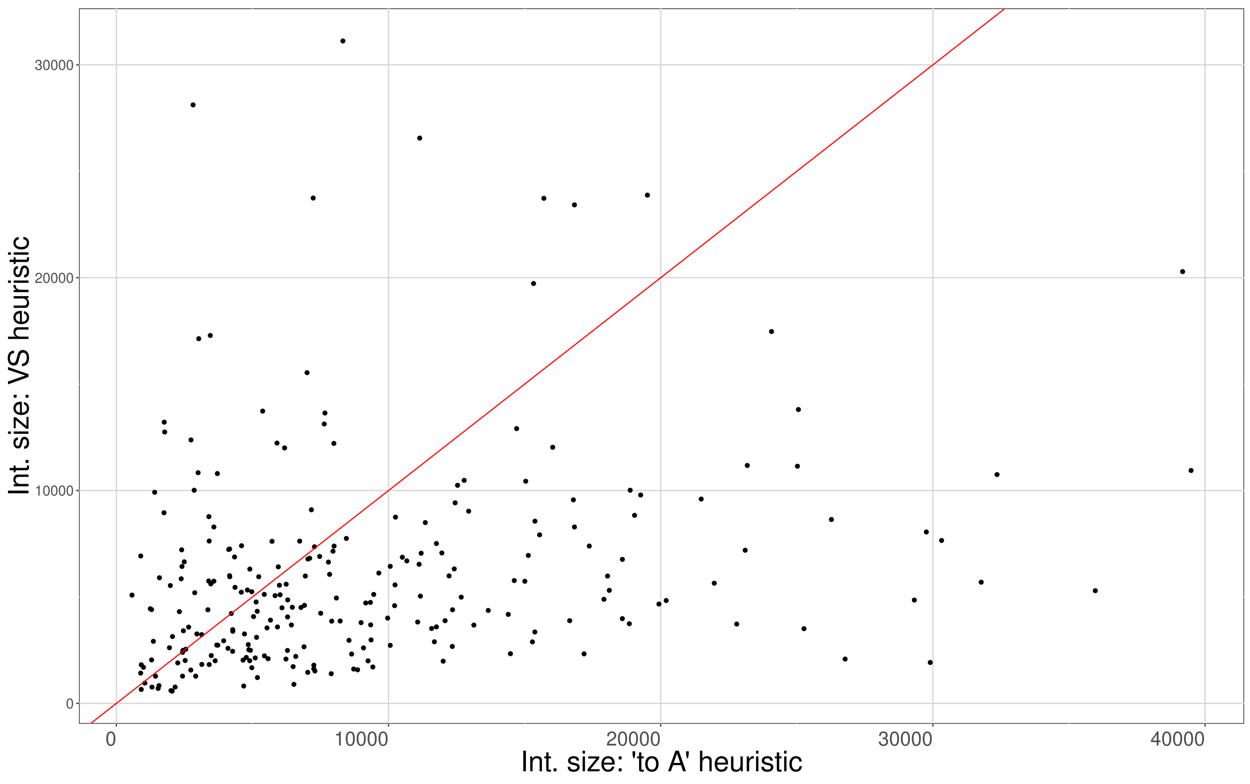


Basically, in this setting it is not obvious how to align the diagrams, and despite the simple diagram structure, I think the diagrams might explode if we try to change the variables order carelessly. So, our heuristic beats a naive one both in terms of runtime and the diagram sizes.

It seems to me this problem would work as an illustration, if we think that it is not too theoretical/artificial. Speaking about interpretations, I do not have a quite good case here. But I would say it is like thinking about propagation of information through social graphs of people with respect to different interests. For example, my social graph as a researcher is very different from my social graph as a hobby Go player. However, I can carry some idea or information through both graphs (at least one hop forward), so it makes sense that ‘locating’ me just once is enough for both graphs. I do not quite see a good intuition why **connection points** should coincide in the two graphs, though. Anyways, I am not sure if it looks solid enough to you, but I would be grateful for any hints in this regard!



(a) Runtimes, sec.



(b) Intersection dia sizes, nodes

Figure 4: Nominal parameters for the two BDD alignment heuristics.