Enabling d-separation in Graph Databases - Supplementary material

MATTIA PALMIOTTO, Lyon1 University, CNRS Liris, France

ANGELA BONIFATI, Lyon1 University, CNRS Liris & IUF, France

ANDREA MAURI, Lyon1 University, CNRS Liris, France

10 THEOREM 4.1

Remark 10.1. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a DAG, $Z \subset \mathcal{V}$, and let $\mathcal{G}_C$ be the d-collision graph of $\mathcal{G}$ relative to Z. It is remarked that:

(a) If $e = (u, v) \in \mathcal{E}$ and $u \notin Z$, then there exists an edge $e' \in \mathcal{E}_C$ having u and v as endpoints. Moreover, if such an edge e' is directed, it is equal to e, that is, it has u as source and v as target, the latter being an unconditioned collider in $\mathcal{G}$.

(b) If $e' \in \mathcal{E}_C$, then there exists an edge $e = (u, v) \in \mathcal{E}$, where u and v are the same endpoints of e', and $u \notin Z$. Moreover, if e' is directed, then it is equal to e, that is, it has u as source and v as target, the latter being an unconditioned collider in $\mathcal{G}$.

(c) Thanks to (a), by setting $\mathcal{E}_k = \{(u, v) \in \mathcal{E} \mid u \notin Z\}$, we can introduce the function $g : \mathcal{E}_k \longrightarrow \mathcal{E}_C$, such that for each $e \in \mathcal{E}'_k$, $g(e)$ corresponds to the edge $e' \in \mathcal{E}_C$ whose endpoints are the same as e. From (b), we deduce that g is bijective.

Definition 10.2 (Head-to-head path)**.** Let $\mathcal{G}_m = (\mathcal{V}_m, \mathcal{E}_m)$ be a mixed graph, and let $p' = e'_1 e'_2 \dots e'_n$ be a path in $\mathcal{G}_m$. We say that p' is a *head-to-head* path if there exists $i \in \{1, \dots n-1\}$ such that $e'_i : N_i \longrightarrow N_{i+1}$ is right-oriented, and $e'_{i+1} : N_{i+1} \longleftarrow N_{i+2}$ is left-oriented. In particular, any head-to-head path is face-to-face.

Lemma 10.1. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a DAG, $Z \subset \mathcal{V}$, and $\mathcal{G}_C = (V, \mathcal{E}_C)$ the d-collision graph of $\mathcal{G}$ relative to Z. Then, a path p' in $\mathcal{G}_C$ is face-to-face if and only if it is head-to-head.*

Proof. Let $p' = e'_1 e'_2 \dots e'_n$ be a path in $\mathcal{G}_C$.

By Definition 10.2, if p' is head-to-head, that immediately implies that p' is face-to-face.

Now, suppose that p' is face-to-face, and we prove by contradiction that p' is a head-to-head path.

Assume that p' is a face-to-face path that is not head-to-head. Then, there exist $1 \leq i, j \leq n$, $j \geq i + 2$, such that $e_i : N_i \longrightarrow N_{i+1}$ is right-oriented, $e'_j : N_j \longleftarrow N_{j+1}$ is left-oriented and for each $i < k < j$ we have that $e'_k : N_k - N_{k+1}$ is undirected. For instance, something like:

$$N_i \longrightarrow N_{i+1} - N_{i+2} - N_{i+3} \longleftarrow N_{i+4} \quad (j = i + 3)$$

Observe that, since N_{i+1} and N_j have an incoming directed edge in $\mathcal{G}_C$, they are unconditioned colliders in $\mathcal{G}$. In particular, both N_{i+1} and N_j are not in Z and do not have descendants in Z.

Let g be the function introduced in Remark 10.1. Let $p = e_1 e_2 \dots e_n$ be the path in $\mathcal{G}$ such that $g(e_i) = e'_i$ for $i = 1, \dots, n$. For a generic i, the edges e_i and e'_i both have N_i and N_{i+1} as endpoints. In other words, the paths p and p' correspond to the same sequence of nodes. Take the *smallest* k, $i < k \leq j$ such that e_k is left-oriented; note that such a k always exists because e_j is left-oriented. We distinguish among three cases:

Authors' addresses: Mattia Palmiotto, Lyon1 University, CNRS Liris, Lyon, France, mattia.palmiotto@univ-lyon1.fr; Angela Bonifati, Lyon1 University, CNRS Liris & IUF, Lyon, France, angela.bonifati@univ-lyon1.fr; Andrea Mauri, Lyon1 University, CNRS Liris, Lyon, France, andrea.mauri@univ-lyon1.fr.

Case (i). $k = i + 1$. This means that in $\mathcal{G}$ we have

$$N_i \longrightarrow N_{i+1} \longleftarrow N_{i+2}$$

Therefore, N_{i+2} is a parent of N_{i+1} in $\mathcal{G}$. By the previous observation, N_{i+1} is an unconditioned collider, thus the edge e'_{i+1} should be directed in $\mathcal{G}_C$. However, $e'_{i+1} : N_{i+1} - N_{i+2}$ is undirected, a contradiction.

Case (ii). $k = j$. This means that in $\mathcal{G}$ we have

$$N_{j-1} \longrightarrow N_j \longleftarrow N_{j+1}$$

Therefore, N_{j-1} is a parent of N_j in $\mathcal{G}$. By the previous observation, N_j is an unconditioned collider, thus the edge e'_{j-1} should be directed in $\mathcal{G}_C$. However, $e'_{j-1} : N_{j-1} - N_j$ is undirected, a contradiction.

Case (iii). $i < k < j$. We have in p' pattern:

$$N_{k-1} - N_k - N_{k+1}$$

corresponding to the pattern in p:

$$N_{k-1} \longrightarrow N_k \longleftarrow N_{k+1}$$

Since the edges e'_{k-1} and e'_k are undirected, N_k is a totally or partially conditioned collider. This means that either N_k is in Z or it has a descendant in Z. In either case, since N_k is a descendant of N_{i+1} (by the way we chose k), we also have that N_{i+1} has a descendant in Z, a contradiction by the previous observation. In all cases, we end up with a contradiction. Therefore, our initial assumption that p' is not a head-to-head path is wrong. □

10.1 Proof of Theorem 4.1

The proof is divided into two parts: soundness and completeness. Before proceeding, it is useful to recall the function g introduced in Remark 10.1(c).

Soundness. Claim: *if there exists an unhindered path in $\mathcal{G}_C$ between u and v, then u and v are d-connected in $\mathcal{G}$ conditional on Z.*

Let $p' = e'_1 e'_2 \dots e'_n$ be an unhindered path between u and v in $\mathcal{G}_C$, $p = e_1 e_2 \dots e_n$ be the path in $\mathcal{G}$ such that $e'_i = g(e_i)$ for each $1 \leq i \leq n$. For a generic i, the edges e_i and e'_i both have N_i and N_{i+1} as endpoints. In other words, the paths p and p' correspond to the same sequence of nodes; in particular, they have the same start node u and end node v. Therefore, it is sufficient to show that p is an open path in $\mathcal{G}$.

To prove that any intermediate node of p is not a blocking node, it is sufficient to demonstrate that an arbitrary intermediate node N_i of the paths p and p' is not a blocking node in p. Consider the five possible patterns involving N_i in p':

(i) $N_{i-1} \longrightarrow N_i \longrightarrow N_{i+1}$ (or $N_{i-1} \longleftarrow N_i \longleftarrow N_{i+1}$)
(ii) $N_{i-1} \longleftarrow N_i \longrightarrow N_{i+1}$
(iii) $N_{i-1} - N_i \longrightarrow N_{i+1}$ (or $N_{i-1} \longleftarrow N_i - N_{i+1}$)
(iv) $N_{i-1} \longrightarrow N_i - N_{i+1}$ (or $N_{i-1} - N_i \longleftarrow N_{i+1}$)
(v) $N_{i-1} - N_i - N_{i+1}$

For each case, we prove that N_i is not a blocking node in p:

(i) By Remark 10.1 (b) we have $e_{i-1} = e'_{i-1}$ and $e_i = e'_i$, that is, p has the same pattern $N_{i-1} \rightarrow N_i \rightarrow N_{i+1}$; in addition, $N_i \notin Z$. Therefore, N_i is a mediator in p that is not totally conditioned, meaning that it is not a blocking node in p.

(ii) By Remark 10.1 (b) we have $e_{i-1} = e'_{i-1}$ and $e_i = e'_i$, that is, p has the same pattern $N_{i-1} \leftarrow N_i \rightarrow N_{i+1}$; in addition, we have $N_i \notin Z$. Therefore, N_i is a confounder in p that is not totally conditioned, which means that it is not a blocking node in p.

(iii) Since $e'_{i-1} = N_{i-1}{-}N_i$ is an undirected edge, then e_{i-1} can either be left-oriented or right-oriented. In the first case, N_i is a confounder in p, in the second case N_i is a mediator in p. Furthermore, $N_i \notin Z$ by Remark 10.1 (b). Consequently, N_i does not block the path p.

(iv) Since $e'_i = N_i{-}N_{i+1}$ is an undirected edge, then e_i can be, in principle, either left-oriented or right-oriented. However, we show that e_i can only be right-oriented. The fact that $e'_{i-1} : N_{i-1}{\rightarrow}N_i$ in $\mathcal{G}_C$ is directed means that N_i is an unconditioned collider between N_{i-1} and all the other parents of N_i. Thus, if e_i was incoming in N_i, that would mean that N_{i+1} is a parent of N_i, and consequently e'_i would be directed (by definition of d-collision graph), a contradiction. Therefore, e_i has to be right-oriented, meaning that N_i is a mediator in p. In addition, $N_i \notin Z$ by Remark 10.1 (b), implying that it is not a blocking node in p.

(v) If the corresponding pattern of N_i in p is $N_{i-1} \rightarrow N_i \rightarrow N_{i+1}$, (resp. $N_{i-1} \leftarrow N_i \leftarrow N_{i+1}$), then N_i is mediator which is not totally conditioned, otherwise the edge e'_i (resp. e'_{i-1}) would not exist. Similary, if instead $N_{i-1} \leftarrow N_i \rightarrow N_{i+1}$ in p, then N_i is a confounder that is not totally conditioned. Lastly, if $N_{i-1} \rightarrow N_i \leftarrow N_{i+1}$ in p, then N_i is a totally or partially conditioned collider in $\mathcal{G}$, since e'_{i-1} and e'_i are undirected. In all cases, N_i is not a blocking node.

Completeness. Claim: *if u and v are d-connected in $\mathcal{G}$ conditional on Z, then there exists an unhindered path in $\mathcal{G}_C$ between u and v.*

Let $p = e_1 e_2 \dots e_n$ be an open path between u and v, and let $p' = e'_1 e'_2 \dots e'_n$ in $\mathcal{G}_C$ such that $e'_i = g(e_i)$. For a generic i, the edges e_i and e'_i have N_i and N_{i+1} as endpoints. The paths p and p' correspond to the same sequence of nodes, in particular they have the same start node u and end node v. It is sufficient to prove that p is not a face-to-face path. By Lemma 10.1, this is equivalent to proving that p' is not a head-to-head path.

By contradiction, assume that p' is a head-to-head path. Then, there exists $2 \leq i \leq n$ such that $e'_{i-1} : N_{i-1}{\rightarrow}N_i$ is right-oriented and $e'_i : N_i{\leftarrow}N_{i+1}$ is left-oriented. We find in p' the pattern:

$$N_{i-1}{\rightarrow}N_i{\leftarrow}N_{i+1} \tag{1}$$

By Remark 10.1(b), the edges e_i and e_{i+1} in p are also right-oriented and left-oriented, respectively. Therefore, we find the same pattern (1) in p too. Now, since the edges e'_{i-1} and e'_i are directed in $\mathcal{G}_C$, by Remark 10.1(b) N_i is an unconditioned collider in p, a contradiction because the path p is open.

In conclusion, the initial assumption that p' is a head-to-head path is wrong; therefore, in $\mathcal{G}_C$ there exists the unhindered path p' between u and v.

11 PG TRANSFORMATION RULES R_V AND R_E

In §5 we introduced two PG transformation rules, R_V and R_E. Rules R_V and R_E match all nodes and edges, respectively, in the input graph, and recreate them in the output graph, associating them with the same property keys and values as they have in the input graph. It is not possible to do this with the edge rule R_E alone, because it would not match the isolated nodes, if there are any.

Let $\mathcal{G}_{in}$ be the input property graph, with labeling function λ_{in}, and let δ_{in} be the partial function assigning key-value properties in $\mathcal{G}_{in}$. Let $\mathcal{G}_{out}$ be the output property graph, with labeling function λ_{out}, and let δ_{out} be the partial function assigning key-value properties in $\mathcal{G}_{out}$.

Rule R_V is expressed as:

$$R_V: \quad (a) \Longrightarrow \underset{\langle\ \delta_{out}(\{a\}\times\mathcal{K}) = \delta_{in}(\{a\}\times\mathcal{K})\ \rangle}{(\ a : \lambda_{in}(a)\ )} \tag{2}$$

For the *d-collision graph generation* phase, rule R_E can be represented by a rule R_E^d that replicates only the directed edges:

$$R_E^d: \quad (a) \xrightarrow{\quad r \quad} (b) \Longrightarrow (a) \xrightarrow[\langle\ \delta_{out}(\{r\}\times\mathcal{K}) = \delta_{in}(\{r\}\times\mathcal{K})\ \rangle]{r:\lambda_{in}(r)} (b) \tag{3}$$

Differently, for the *Identification of d-separated nodes* phase, we need to match both directed and undirected edges. Unfortunately, we cannot match simultaneously both directed and undirected edges in a natural way. We can still do it if we use unions of patterns in the LHS of the rule:

$$R_E: \quad (a) \xrightarrow{r_1} (b), \quad (c) \overset{r_2}{-\!\!-\!\!-} (d) \Longrightarrow (a) \xrightarrow[\langle\ \delta_{out}(\{r_1\}\times\mathcal{K}) = \delta_{in}(\{r_1\}\times\mathcal{K})\ \rangle]{r_1:\lambda_{in}(r_1)} (b), \quad (c) \xrightarrow[\langle\ \delta_{out}(\{r_2\}\times\mathcal{K}) = \delta_{in}(\{r_2\}\times\mathcal{K})\ \rangle]{r_2:\lambda_{in}(r_2)} (d) \tag{4}$$

However, if $\mathcal{E}^u$ and $\mathcal{E}^d$ are the sets of undirected and directed edges f $\mathcal{G}_{in}$, respectively, then Rule R_E in 4 matches $|\mathcal{E}^d|\cdot 2|\mathcal{E}^u|$ occurrences, which is inefficient. We can make the process more efficient by utilizing two rules to replicate the edges, of which one is R_E^d defined above for directed edges, and the other for undirected edges is defined as follows:

$$R_E^u: \quad (a) \overset{r}{-\!\!-\!\!-\!\!-} (b) \Longrightarrow (a) \underset{\langle\ \delta_{out}(\{r\}\times\mathcal{K}) = \delta_{in}(\{r\}\times\mathcal{K})\ \rangle}{\overset{r:\lambda_{in}(r)}{\rule{8em}{0.4pt}}} (b) \tag{5}$$

Rule R_E^u matches an arbitrary undirected edge $\{u, v\}$ twice, one with $u = (a)$ and $v = (b)$, and the other with $u = (b)$ and $v = (a)$. By sequentially applying R_E^d and R_E^u we obtain $|\mathcal{E}^d| + |\mathcal{E}^u|$ matches, which is often much lower than $|\mathcal{E}^d| \cdot 2|\mathcal{E}^u|$. Moreover, if each node is assigned with a mutual exclusive numerical identifier, we can constrain (a) and (b) such that (a) has a lower numerical identifier than (b), producing only one match for each undirected edge.

12 SYNTHETIC DAGS

In the following, we provide detailed info about how the synthetic DAGs were generated.

ER: Erdős–Rényi. To generate the DAG, we first used the Erdős–Rényi model that takes as input an ordered pair (N, p), with $N \in \mathbb{N}$ and $p \in [0, 1]$. The algorithm returns a random undirected graph $G(N, p)$ with N nodes, and each possible edge $\{u, v\}$ is created with probability p. In our experiments, $N \in \{1000, 10000, 20000\}$, and $p = 0.0025$. To generate the undirected graph, we used the method `networkx.erdos_renyi_graph(N,p,directed=False)` of the NetworkX Python library. If we had set `directed=True` the resulting graph could have contained directed cycles. Therefore, we instead transformed the obtained undirected graph into a DAG using an ad-hoc function which assigns a topological ordering to the nodes in the input graph, and then substitutes each undirected edge with a directed edge that targets the node with higher order. In this way, the resulting graph is directed and acyclic. In the case of ER0, we obtained a graph with more than one connected component, and only the largest containing 903 nodes was maintained.

BA: Barabasi-Albert. : To generate the DAG, we first used the Barabasi-Albert model that creates an undirected *scale-free* graph. A graph is scale-free means that the fraction of nodes with k neighbors is inversely proportional to k. In particular, there are a very few nodes with a lot of neighbors and many nodes with scarcely any neighbors. A BA graph is generated with preferential attachment criterion; that is, given an initial input graph, each newly created node is randomly attached to a fixed m already existing nodes, and the probability of a node to be chosen is proportional to the number of connections that it has already. Consequently, the more neighbors a node has, the more likely it is connected to the newly created nodes. To generate the `BA` undirected graph, we used the method `networkx.barabasi_albert_graph(N, m=3)` of NetworkX. We used the default initial graph given by a star of $m + 1$ nodes. Then, we transformed the obtained undirected graph into a DAG in the same way we did for `ER` graphs.

LF: Layered or feed-forward. In this model, the DAG can be partitioned into a sequence of *layers* $\{L_i\}_{i=1}^{I}$ of varying number of nodes, and the edges only go from layer i to layer $i + 1$. We set the number of layers I equal to $\texttt{int}\left(\sqrt{|\mathcal{V}|}\right) + 1$. The number of nodes in each layer was determined using an ad-hoc function which takes I as input and returns a sequence of I natural numbers, where the $i-$th element establishes the number of nodes in the $i-$th layer. The returned sequence of I numbers was constrained to return numbers between $\texttt{min} = \texttt{int}(N/2)$ and $\texttt{max} = \texttt{int}(3N/2)$. The returned sequence was passed as input together with the probability parameter $p = 5/(\texttt{min} - 1)$ to an ad-hoc method `layered_dag`. This method iteratively considers each layer L_i, $i = 2, \ldots, n$, and connects each node in the layer with k random nodes of the previous layer, where $k \sim 1 + Binomial(|L_{i-1}|, p)$. Then, if any node in L_{i-1} does not have any outgoing edge, that node is connected to $k' \sim 1 + Binomial(|L_i|, p)$ of layer L_i. This way, we ensure that each node has at least 1 incoming edge and 1 outgoing edge (except for the extreme layers).

TR: Tree. In this type of DAG, there are no (undirected) cycles, which means that almost every node of the DAG has exactly one incoming edge, except the only root. In particular, there are no colliders and each pair of nodes is connected by exactly one path. To obtain the DAG, we first created an undirected tree using the method `networkx.random_labeled_rooted_tree` of NetworkX. Then, we used `networkx.bfs_tree(Tree,source=root)`, another method to transform the undirected tree into a DAG.

13 OUTLIERS

During our experiments, three lists of runtimes were built for each input tuple $(\mathcal{G}, |X|, |Z|)$: one with the runtimes r_h^g of the *d-collision graph generation* phase, one with the runtimes r_h^i *Identification of d-separated nodes* phase, and another with the total runtimes $r_h = r_h^g + r_h^i$. We noticed that for the smallest graphs SACHS, C01 and C01, Neo4j used different internal functions than usual to execute the queries of the d-collision Graph Method, which resulted in unreasonably high runtimes. This lead to necessity to remove these outliers before the computation of the statistics.

To identify the outliers, we first detected the ones for the lists of total runtimes of the d-collision Graph Method. if $r_h = r_h^g + r_h^i$ was an outlier, then r_h, r_h^g and r_h^i were all removed from their respective lists. We used the following criterion to classify an element r_h in a list of total runtimes as an outlier or not. We computed its associated modified z-score:

$$z_h = \frac{0.6745(r_h - \mu)}{\text{MAD}}$$

where μ is the mean of the runtimes in the list, and

$$\text{MAD} = \text{median}(|r_h - \mu|)$$

is the Median Absolute Deviation of the list. The total runtime r_h was classified as an outlier for the list if $z_i > 5$.

For three graphs SACHS, C01 and C01, we identified and removed outliers from the list of runtimes of the baseline in the same way. This allows a fair comparison between our method and the baseline. Once the lists of the runtimes were cleaned off the outliers, we computed the mean runtime and the standard deviation for each list.

14 RESULTS OF THE EXPERIMENTS

When executing a query that marks all nodes which are reachable from a node in a small set (Z for the *d-collision graph generation* phase and X for the *Identification of d-separated nodes* phase) through a directed or possibly directed path, we observe substantially higher runtimes on the ER DAGs than on the BA and TR DAGs. The reason lies in their structural properties.

ER DAGs are characterized by a homogeneous degree distribution (approximately Poisson). Their nodes are relatively uniformly connected. Consequently, the search expands broadly and traverses a large portion of the graph before terminating.

Differently, the degree distribution in BA DAGs is strongly asymmetrical: most nodes have small degree and limited reachability, and a very few nodes, the “hubs”, have many connections. Therefore, traversals typically terminate quickly, except in regions near hubs. This leads to significantly faster query execution when the set from which the search starts, or must end, is small.