CS594 Provenance & Explanations Paper Summary and Critique

Amy Byrnes

I. Introduction

In their 2022 paper "HypeR: Hypothetical Reasoning With What-If and How-To Queries Using a Probabilistic Causal Approach", Galhotra, Gilad, Roy, and Salimi describe their framework, HypeR, for performing what-if and how-to queries using probabilistic causal models. The paper is split into seven parts (1. Introduction, 2. Probabilistic Updates in Hyper, 3. Probabilistic What-If Queries, 4. Probabilistic How-To Queries, 5. Experiments, 6. Related Work, and 7. Conclusions) and includes an appendix with four proofs that were either not appropriate for or did not fit into the main body of the paper.

II. Summary of Paper

A. Introduction

The paper begins by motivating the desire for users to be able to perform hypothetical reasoning with a database. What-if and how-to queries are explained, with example queries given in Example 1. The problem the paper aims to address is then outlined: The value of a tuple t_i's attribute $A_x[t_i]$ may be affected by the values of other of the tuple's attributes $A_y[t_i]$, as well as the attributes of other tuples $A_y[t_j]$. What's more, these dependencies may be probabilistic; the exact effect of a change in attribute value is not known. The paper asserts that provenance-based systems are not sufficient to answer hypothetical queries on databases with these kinds of dependencies, as provenance does not capture changes in the values of attributes.

Instead, the authors promise to describe HypeR, a framework for performing hypothetical queries on databases with probabilistic dependencies between attributes. HypeR has been implemented as an extension to SQL and has been shown to produce accurate results in reasonable times.

Also in the paper's introduction, the running example of the **Amazon Product Database** is introduced (Figure 1). The Amazon Product Database consists of two tables: **Product**, a table of unique products and their attributes (PID, Category, Price, Brand, Color, and Quality), and **Review**, a table of unique reviews and their attributes (PID, ReviewID, Sentiment, and Rating). Each product in Product may have one or more corresponding reviews in Review, indicated by the foreign key PID in Review. A graph denoting the dependencies between the attributes is also given (Example 2, Figure 2).

B. Probabilistic Updates in HypeR

This section begins by listing the notation and terminology used in the rest of the paper for referring to databases, relations, tuples, and attributes. Example 3 is used to demonstrate the notation and terms applied to the Amazon Product Database.

The rest of this section is split into two parts, describing the necessary concepts for modeling a database with probabilistic attribute dependencies.

1) Probabilistic Hypothetical Updates: This subsection aims to define a hypothetical update on a database with probabilistic attribute dependencies. This first requires the definition of the concept of the **possible worlds of a database**, $PWD(D)$ (Definition 1). Then, a **hypothetical update** is defined as a four-tuple that describes an intervention to be performed on the database so that the update is carried out (Definition 2). Finally, an initial definition is given of the **post-update distribution** (Definition 3). As the database has probabilistic dependencies, the effect of any update to the database is non-deterministic. Rather than producing a single new database, a hypothetical update puts the database into an uncertain state, characterized by the probability distribution of possible worlds of the database after the update has been performed; this is the post-update distribution.

2) Causal Model for Probabilistic Updates: This subsection is intended to explain the concepts of **probabilistic relational causal models (PRCMs)** and how a database with probabilistic attribute dependencies is modeled as a PRCM.

A PRCM consists of a set of noise variables, the probability distribution of those noise variables, a set of observed variables, and a set of structural equations that define the causal dependencies between the variables. The noise variables and their distribution are, by definition, unobserved, and so must be represented by some other means. A ground causal graph is created for the PRCM, where the nodes are the observed variables and the directed edges represent causal dependencies among the variables. The effect of unobserved noise is captured by setting the structural equations to be conditional probability distributions of each variable, given its causal parents.

This model is applied to a database with probabilistic attribute dependencies as follows: the observed variables are the values of the attributes of each tuple (i.e. the grounded attributes). A database of N tuples and M attributes therefore produces a graph with $N \times M$ nodes. A directed edge from the value of an attribute in one tuple to the value of an attribute in another indicates that a change to the value of that attribute in that tuple may result in a change to the value of the second attribute in the second tuple.

The actual effect of changing parent variables on their children is given by the set of conditional probabilities from the structural equations. These can be estimated directly from the database using well-known techniques. Several assumptions are made to facilitate this estimation: 1. Dependencies are acyclic. 2. There is some function which may be applied to the parents of a variable that produces a fixed-size result, without changing the conditional probability distribution of the child. 3. The conditional probability of the value of a child variable given its parents values depends only on these values, and not the tuple they come from; i.e. the value of a review's rating is affected by the review's sentiment in the same way regardless of who left the review.

Example 4/Figure 3 show a partial causal DAG for the Amazon Product Database. Example 5 demonstrates the application of an aggregation function that reduces a variable-length set of values to a fixed-size value.

The section ends by combining the concepts of the PRCM and its DAG and the previously defined concepts of the **hypothetical update** and **post-update distribution**. Modeling the database to be updated with a PRCM, the intervention performed by the hypothetical update is not just the modification of database attributes, but the modification of the PRCM and its structural equations. The causal effects of the update can then be propagated through the database according to the conditional probability distributions of each grounded attribute. The post-update distribution over possible worlds of the updated database is similarly determined using the pre-update distribution of the database, defined by the conditional probability distributions of the grounded attributes. The paper promises that this reduction of the post-update distribution to the pre-update distribution is possible so long as the distribution is conditioned on a set of variables that satisfy the backdoor criterion, with further elaboration pushed to later sections of the paper.

C. Probabilistic What-If Queries

This section describes how HypeR extends SQL to include syntax for performing what-if queries, and the logic behind how HypeR performs these queries with relative efficiency.

1) Syntax of Probabilisitic What-If Queries: Here, the form of a HypeR what-if query in SQL is described. Several new operators (USE, PRE, POST, WHEN, UPDATE, FOR, and OUTPUT) are introduced to facilitate the various steps of a what-if query.

The PRE operator specifies that the pre-update value of an attribute should be used. The POST operator likewise specifies that the post-update value of an attribute should be used.

The USE operator defines the relevant view for the what-if query, or the set of tuples that the query should be performed on. The WHEN operator selects a subset of tuples from the relevant view to perform the update on. The UPDATE operator performs a given update on the tuples selected by FOR from the relevant view. The FOR operator selects tuples in the updated relevant view to use to compute the output. The OUTPUT operator computes the new values of the given attribute for each tuple selected by the FOR operator, and aggregates those values into a single result.

Example 6/Figure 4 demonstrate an example what-if query performed on the Amazon Product Database.

2) Semantics of Probabilistic What-If Queries: This section explains the logic each operator uses to perform its task. The operators are evaluated in the following order: USE, WHEN, UPDATE, FOR, and OUTPUT.

The logic of the USE, WHEN, UPDATE, and FOR operators is fairly simple. USE performs a standard SQL query to produce the relevant view. The remaining operators act on that relevant view. WHEN and FOR select tuples from the pre- and post-update relevant view according to given predicates μ_{WHEN} and μ_{FOR}. The UPDATE operator applies the given function f to the specified attribute B to produce the post-update relevant view.

Things become more complicated with the OUTPUT operator. The post-update values of the given attribute needs to be calculated for all the tuples selected by the FOR operator. However, these post-update values are governed not by a deterministic rule, but by a probability distribution. Therefore, for every tuple that is included in the output calculation, the output attribute of that tuple probabilistically takes one of several possible values. A common and sensible way to reduce possible results to a single result is to take the expectation value, which is what the authors do.

Definitions 4 and 5 define the result of the OUTPUT operator concretely. The possible values of the output attribute are enumerated via enumeration of the possible worlds of the post-update database. In a possible world of the post-update database, the output attribute of each tuple selected by FOR has a single value. The output attribute values are aggregated by some aggregation function, and the value of this aggregation is the **result of the query on the possible world** (Definition 4). The final result of the what-if query, then, is the expectation value of the **result of the what-if query over all the possible worlds** (Definition 5).

3) Computation of What-If Queries: Defintion 5 of the **result of a what-if query** leaves two important issues unresolved. The first is that enumerating all possible worlds of a database is exponential in the size of the database, and therefore hardly an "efficient" algorithm. The second is that computation of the expectation value of the results of the queries over each possible world requires knowledge of the post-update distribution over the possible worlds, which is as yet unknown.

The first problem is solved via **block-independent decomposition** of the database and the use of **decomposable aggregate functions**. A **block-independent decomposition** of a database is defined as a partitioning into the database of sets of tuples such that the sets span the entire database, the sets do not overlap, and the tuples of each set are causally independent of the tuples in every other set. Tuples are causally independent of each other if there are not paths, directed or otherwise, between any of their grounded attributes. Example

7 provides a block-independent decomposition of the Amazon Product Database.

To compute the result of a what-if query on a single possible world, an aggregate function must be applied to the values of the output attribute in that possible world. Therefore, for block-independent decomposition to be used, the aggregate function must be decomposable; it must be able to be applied to the tuples in each block of the block-independent decomposition, and the those results must be aggregated into a final value that is the same as if the aggregation function was applied to entire database. Definition 6 provides a formal listing of the requirements for a function to be decomposable. Example 8 demonstrates that the function AVG is decomposable; the paper asserts that the other aggregation functions supported by HypeR, SUM and COUNT, are also decomposable.

Finally, Proposition 1 proposes that the result of a what-if query on a database can be computed over a block-independent decomposition of that database, using a decomposable aggregate function. The proof for this is given in the Appendix.

The paper asserts that computing the what-if query result over independent blocks will speed up the computation, as "fewer tuples make the computation more efficient." Thus the first problem with computing what-if query results as defined in Definition 5 is addressed.

The second problem, that the post-update distribution appears necessary for computing the result of a what-if query but is unknown, is solved by reducing the post-update distribution to the pre-update distribution.

Proposition 2 suggests that, when aggregating with COUNT, the expectation value of the result of a what-if query on a database block can be written as the sum of the post-update probabilities that some condition is true in the post-update possible world, given that some condition is true in the pre-update database. These conditions are clauses from the predicate μ_{FOR}, after it has been converted into a CNF. Further explanation of this proposition and proof is given in the Appendix.

Generalizing from Proposition 2, the paper gives the post-update distribution as the conditional probability of attribute A_i taking the value a_i given that attribute A_j has the value a_j, along with some other predicate. The paper asserts that this conditional probability can be marginalized over some set of grounded attributes $\boldsymbol{C}$ that meets the backdoor criterion with respect to the attribute A_i and the update attribute B. The first term of the marginalized probability can be further marginalized over the values of the update attribute B. These two s produce terms that are explicitly from the pre-update distribution. The final term has the form $Pr_{D,U}(A_i = a_i | B = b, C = c, A_j = a_j, \mu_{\text{WHEN}})$, which, because C meets the backdoor criterion, **can be reduced to the equivalent pre-update distribution**, where the values of B are replaced with their updated values.

The paper acknowledges that these marginalizations require summing over the domains of the update attribute B and the backdoor attributes $\boldsymbol{C}$. However, the authors propose that as the pre-update distribution is ultimately what is used to calculate the result, it is sufficient to replace the domains of these attributes with the sets of observed values of these attributes, greatly reducing the values that would need to be summed over. Therefore, the second problem of producing the post-update distribution in a tractable way has also been addressed.

All that is left is to actually obtain the pre-update distribution, which can be done using the database and standard machine learning/causal techniques.

D. Probabilistic How-To Queries

This section describes how HypeR extends SQL to include syntax for performing how-to queries, and the logic behind how HypeR performs these queries by utilizing what-if queries.

1) Syntax of Probabilistic How-To Queries: How-to queries in HypeR use the PRE, POST, USE, WHEN, and FOR operators for the same purposes as in what-if queries.

What-if queries make specific use of the operators UPDATE and OUTPUT. Likewise, how-to queries have their own operators: HOWTOUPDATE, which optionally uses the operator LIMIT, and TOMAXIMIZE/TOMINIMIZE.

HOWTOUPDATE specifies which attributes in the relevant view produced by USE may be updated during the query computation. LIMIT defines to what extent those attributes may be updated. Updates to attributes can be restricted to keep those attributes in certain intervals, or in a set of values. Updates can also be restricted so that the post-update value of an attribute stays within a given L1 distance of the pre-update value of the attribute.

The TOMAXIMIZE and TOMINIMIZE operators specify whether the given output attribute should be maximized or minimized, respectively, and perform the computation to find the set of attribute updates that will produce the desired optimization of the output attribute.

Example 9/Figure 5 demonstrate an example how-to query performed on the Amazon Product Database.

2) Semantics of Probabilistic How-To Queries: This section explains the logic behind the TOMAXIMIZE and TOMINIMIZE operators. Because the logic is virtually identical for both operators, with the only difference being whether the maximum or minimum value is sought, the explanation is framed in terms of only the TOMAXIMIZE operator, but applies also to the TOMINIMIZE operator.

The TOMAXMIZE operator produces a how-to query result: the set of attributes to update and their new values, in order to produce the maximum value of the output attribute. This result is produced using the notion of what-if queries that was defined in the section "Probabilistic What-If Queries."

As with the results of what-if queries, the result of a how-to query requires two definitions to properly explain. Definition 7 defines the concept of a **candidate what-if query**. This is a what-if query that performs some update to the update attributes specified by HOWTOUPDATE, within the limits set by LIMIT, and calculates the resulting output on the attribute

that the how-to query seeks to optimize. Example 10 provides a candidate what-if query for the how-to query from Example 9/Figure 5. Given a how-to query, we can define a set of candidate what-if queries that produce all possible results of all possible updates within the update parameters defined by the how-to query. Then the **result of the how-to query** is the set of updates made by the candidate what-if query that produces the minimum or maximum result (Definition 8).

3) Computation of How-To Queries: While relatively efficient computation of what-if queries was described in the section "Computation of What-If Queries", the paper recognizes that computing every what-if query in the update space defined by a how-to query is not a feasible solution. Instead the authors show how the problem of optimizing the result of a what-if query can be transformed into an instance of an Integer Program.

The possible updates are enumerated, with continuous domains being discretized as needed. As in Proposition 2, the result of the candidate what-if queries is represented as a linear function, which is constrained according to the list of possible updates. Maximizing the value of a linear function subject to some constraints is an Integer Program. The problem can therefore be solved using existing efficient solutions.

E. Experiments

The authors tested HypeR in a number of capacities. Their overall takeaway from these tests is that HypeR is both accurate and performant.

1) Datasets and Baselines: The authors used a mix of real and synthetic datasets to establish HypeR's capabitlities. The real datasets are the **Adult** dataset, which contains demographic information about individuals, and the **German** dataset, which contains information about the credit risk of bank account holders. The **Amazon** dataset also uses real data, with the authors adding some attributes such as sentiment. The synthetic datasets are the **German-Syn** and **Student-Syn** datasets. The German-Syn dataset is based on the German dataset; the Student-Syn dataset has information about students that may affect their GPA.

Two variations of HypeR are described that were also tested. **HypeR-NB** is a version of HypeR where no prior causal model is available; in this case, HypeR assumes that update and output attributes are affected by all other attributes. **HypeR-sampled** is a version of HypeR where the pre-update distribution is learned from 100k randomly sampled tuples from the database rather than from the entire database.

Two baselines are chosen to compare HypeR against. The **Indep** baseline assumes no dependencies between any of the attributes of any of the tuples. The **Opt-HowTo** baseline performs how-to queries by actually performing all possible candidate what-if queries and finding the one that produces the optimal result.

2) HypeR and its sampling variant: The authors found that using a random sample of 100k tuples from a dataset to compute the conditional distributions used in the query computation produced results that were within 1% of the mean of the results produced when the conditional probabilities were calculated from the entire dataset. Additionally, they found that query-time is linear in the size of the number of tuples used to compute the conditional probabilities. Thus the authors determined that the sampling variant of HypeR, HypeR-sampled, gave the optimal balance of result quality vs time to result.

The results of this experiment are summarized in Figure 6.

3) What-If Real World Use Cases: The real-world datasets do not have ground truth data to compare HypeR's query results to. Instead, the authors used HypeR to perform what-if queries on the Adult and German datasets, and compared their results with the findings of previous regarding the causal dependencies of these datasets. Specifically, they use the query templates in Figure 7 to determine how changing attributes in the Adult dataset affects the Income attribute, and how changing attributes in the German dataset affects the Credit Score attribute. The output attributes that HypeR computed were deemed to be consistent with previously developed understanding of the causal dependencies between the attributes in these datasets.

The results of these experiments are summarized in Figure 8.

4) Solution Quality Comparison: Ground truth results can be computed for the synthetic datasets. As these datasets were generated from clearly defined causal models, the ground truth values were able to be calculated from the structural equations.

For these experiments, the authors tested each of the HypeR variants (HypeR, HypeR-sampled, HypeR-NB), as well as the indep and Opt-HowTo baselines.

On the German-Syn dataset, the HypeR variants all produced what-if results within 5% of ground truth. Specific values are not given for the what-if results on the Student-Syn dataset; however, visual analysis of the graphed results (Figure 10) shows that the what-if results are quite close to ground truth. For both datasets, the HypeR variants significantly outperform the indep results. The only exception is what-if queries on the Student-Syn dataset that update the Assignment attribute. In this case the indep baseline also produces results that are quite close to ground truth.

For how-to queries on German-Syn and Student-Syn, the Opt-HowTo method was used to generate ground truth results, compared to HypeR's method of modeling a how-to query as an IP problem. The authors report that HypeR's how-to results were consistent with Opt-HowTo's results for both datasets; however, no results are reported in the text or through tables or graphs.

How-to queries can be tested in another dimension; regardless of whether how-to queries are performed by iterating over all possible candidate what-if queries or by solving an IP problem, any continuous update variables need to be discretized, or else the enumeration of possible values those variables could take would be infinite. The authors tested how the number of buckets the continuous variables were discretized into affected the query result and time to compute the result. They found that the how-to results from Opt-HowTo

and HypeR remained consistent with each other regardless of the number of buckets used for discretization. However, as the number of buckets increased, the time for Opt-HowTo to reach a solution increased exponentially, while the time for HypeR to reach a solution increased very little. This demonstrates that HypeR is able to perform how-to queries with fine resolution of continuous variables and produce good quality results in a relatively short amount of time

5) Runtime Analysis and Comparison: The final analysis the authors perform regards the time it takes for HypeR to compute query solutions. They note that for each query, before the query can be computed, a relevant view must first be generated, and then conditional distributions must be learned from that view. They choose to generate these conditional distributions by training regression functions, and therefore say that HypeR's overall query compute time scales with the regression method.

The authors test the effect of database size, backdoor set size, and query complexity on query compute time when performing what-if queries. Their results are summarized in Table 1, and Figures 11 and 12. They find that increasing the database size and backdoor set size increases the runtime for HypeR, but not for HypeR-sampled, which always uses the same number of tuples to train the regression functions. This shows how the runtime is dominated by the size of the set used to train the regression functions. They additionally find that increasing query complexity by increasing the number of attributes involved in the query does not affect runtime, except when the number of attributes in the FOR operator increases, in which case the runtime increases.

For how-to queries, the authors test the effect of query complexity on runtime by varying the number of attributes that can be updated in the how-to query. They find that increased query complexity does increase runtime, but at several orders of magnitude less than when Opt-HowTo is used, which has a runtime exponential in the number of attributes that can be updated in the query.

F. Related Work

Here the authors reference works that use provenance to perform what-if and how-to queries. They list works presenting other ways that what-if and how-to queries may be performed. They also list works related to probabilistic databases, probabilistic relation causal models, and observational causal inference.

G. Conclusion

The authors summarize what has been presented in the paper: HypeR, its SQL extension, the logic underlying the performance of what-if and how-to queries, and experimental results demonstrating HypeR's efficacy. The authors conclude by outlining future improvements and modifications they would like to add to HypeR.

III. Critique

A. Caveats

I have little experience with the domains that this work falls under (databases, causal inference, etc.), and therefore little experience with the standards for papers in those domains. Some of my critique relates to the clarity of the concepts involved in the paper; however, this is obviously very subjective and perhaps less meaningful coming from the perspective of someone with a low level of background knowledge. It's not exactly a flaw in the paper if someone struggles with concepts in the paper that they've never studied before. All this is to say, this critique is based on *my personal experience* wrestling with this paper.

B. Overall Impression

Overall, I am pretty impressed with this work. The authors synthesize a number of complex ideas to produce a working system for performing hypothetical queries. Between the body of the paper and the appendix, there is a great amount of detail provided to back up the authors' assertions and ideas. It was quite a lot for me to take in. There were many points where I felt as if someone asked me to hold some things for later – but there were a lot of things and they were difficult to hold on to, and I kept dropping them, and when I bent over to pick up the things I'd dropped other things escaped my grasp. For example, reading about the semantics of what-if queries and trying to keep straight the components of a what-if query vs the components of a hypothetical update, which is *part* of a what-if query - this is when I came up with the analogy of trying and failing to hold many things.

None of this is necessarily a problem. Some papers just have a lot of information, a lot of definitions and notation, and this paper is one of those. Deconstructing this paper has definitely been an extremely valuable experience for my future academic career. Rereading a paragraph and realizing that at some point I had come to understand it was highly rewarding.

C. Logical Flow

The paper does a really good job of identifying all the different components needed to make HypeR come together. Hypothetical updates, results of what-if queries on possible worlds, results of what-if queries on the whole database, block-independent decompositions, etc. I really appreciate that all of these things and their place in the framework are distinctly labeled and defined. In that same vein, I think the authors succeed at always telling the reader what they are about to do and why; for example, in section "Computation of What-If Queries", they lay out two key problems that need to be addressed to have a viable method of actually computing what-if queries: efficiency of computing the query over the whole database, and performing calculations that seemingly require the post-update distribution.

There are, however, areas where I think the order in which these components are presented is not ideal for the reader to comprehend how they work together or their importance.

A very minor but illustrative example of my point is in the section "Probabilistic Hypothetical Updates". This section contains three definitions: 1. Possible worlds, 2. Hypothetical updates, and 3. The post-update distribution. I think the small change of reordering these definitions would help to ease the mental burden on the reader. In the paper, the definition of possible worlds comes first, potentially because this definition is not specific to hypothetical queries and is more likely "review" for a reader. The definition that follows is that for hypothetical updates, and then comes the definition for the post-update distribution. The problem is that hypothetical updates really have nothing to do with possible worlds. The reason possible worlds are involved is because the databases that the hypothetical updates are being performed on have probabilistic dependencies between their attributes. However, the definition given in the paper of a hypothetical update would be the same for a completely deterministic database. We need to know about possible worlds because the post-update distribution is a distribution over possible worlds of the database after the hypothetical update is performed. Because the concept of possible worlds is not really tied in to making hypothetical queries until the post-update distribution is defined, the current order of the definitions leaves the reader with a loose end for longer than necessary. It seems to me that the logic flows better if we start with hypothetical updates, explain that an update to a probabilistic database creates possible worlds and what a possible world is, and then explain the post-update distribution.

A more critical example of this issue appears in the section "Computation of What-If Queries". The authors launch right into how a query over the whole database can be decomposed into queries over blocks before getting to how the post-update distribution can be reduced to the pre-update distribution. I understand the logic here; the final equations they produce the result of a what-if query are in terms of independent blocks of the database. These blocks would be what these equations are actually calculated over, and if no block decomposition was done then the same equations can be used with the database as just one big block. However, I see problems with this order of explanation from the standpoint of both readability and emphasis.

A large amount of the paper really hinges on the idea that we are able to compute what-if queries from the pre-update database. The fact that the database can be broken into independent sets of tuples and decomposable functions can be performed on those tuples is good and important, but not really as critical or interesting a result as the reduction of the post-update database to the pre-update database. Having this discussion after the database decomposability discussion to me has the effect of obfuscating the importance of this concept.

There is also the unfortunate human problem that people tend to run out of steam and start skimming as they get to the end of things that they read. Especially long papers, and especially long sections in long papers. Overall, it just seems like it would be better for the paper if everything from Proposition 2 to the end of this section came before everything about decomposability.

Additionally, returning to the idea of mental load on the reader, the fact that the result in Proposition 2 is over a decomposed what-if query result rather than a full what-if query result just seems to add another small but unnecessary layer of complexity on an already complex result. Especially given that after Proposition 2 the authors go back to just referring to the pre- and post-update distributions over the whole database rather than over an independent database block.

My final point on this matter is that I don't really prefer the separation of syntax and semantics of the hypothetical queries. It again adds to that feeling of loose ends that I have to keep track of until they're explained in the future. I think this is more of a personal preference, and my impression from the limited reading in this domain that I've done is that this is a common practice, so I suppose I'll just have to deal with it.

D. Explanations of Crucial Topics

Just as I feel the overall laying out of each logical component of HypeR is quite good, I feel that the explanations of these components is, with some major exceptions that will be discussed shortly, perfectly followable. The SQL syntax the authors have introduced is very readable, and the explanation of its syntax and semantics is clear. The definitions are all very thorough (although I would appreciate a short plain-language summary of some!).

For me, unfortunately, some of the most critical concepts in the paper where the explanations seemed to lack necessary details/integration with the overall idea. The two notable examples are the explanation of probabilistic relational causal models and the use of the backdoor criterion to marginalize the expression from Proposition 2 to get an expression that could be written in terms of the pre-update distribution.

I actually struggled with the PRCMs so much I went and read the paper that this paper references in the section explaining them. I don't totally put that on the authors of this paper as this is obviously a complex idea that I'd never seen before, and sometimes you just have go do some more background reading. However, after doing that background reading and coming back, I still feel that there are some things that are glossed over. For example, the PRCM seems to be explicitly used in two places: the associated DAG is used to find independent sets of tuples for block-decomposition, and the structural equations are the conditional probabilities that form the pre-update distribution. But the structural equations are explained to represent conditional distributions for specific attributes, not for the entire database; we never get an explicit definition of the pre-update distribution in terms of the structural equations' conditional probabilities. It's not *really* necessary, but I do think its piece of connective tissue missing from a concept that can't really afford to have connective tissue missing.

There are also assumptions made about the conditional probabilities in the section "Causal Model for Probabilistic Updates" that are given little justification beyond that these

are normal assumptions and a source to refer to for "more discussion." As I have acknowledged that sometimes you do just need to put in the work and do the background reading, I feel comfortable saying that when I'm reading a 24-page paper, I don't really want to have to go and read something else to understand the authors decisions. What's missing from these assumptions isn't so much justification of why they *can* be made as much as why these authors are making them.

The worst offender is definitely the explanation of the use of the backdoor criterion to make the post-update distribution equivalent to the pre-update distribution. This equivalence is initially asserted with "it is known" - but it comes with a promise of further explanation in a later section. However, while there is more explanation of the backdoor criterion in that section ("Computation of What-If Queries"), it is still just stated that an expression involving the post-update distribution can be simplified to the pre-update distribution. Further explanation is sadly not to be found in the appendix, either. I'm of two minds about this: on the one hand, if this is just an obvious result from the backdoor criterion that I'm somewhat mystified by because I haven't studied causal inference, then my lack of understanding is on me. On the other hand, the authors do explain what it means, formally, for a set of attributes $\boldsymbol{C}$ to meet the backdoor criterion with respect to attributes A and B. It does seem that if that is appropriate to include then a couple sentences elaborating on what utilizing the backdoor criterion means in this context (from what I gather, preventing any interference between A and B by conditioning on $\boldsymbol{C}$) and why it allows us to simplify an expression involving the post-update distribution down to an expression involving the pre-update distribution would also be appropriate, and appreciated.

E. Running Example

The quality and necessity of examples is fairly subjective. I personally love to see a fully worked example. When I am learning something, I love to work out a full example for myself. That being said, I understand that there are many reasons to not include lengthy examples in a published paper.

Having disclosed my personal bias re: examples, I do think the utilization of the running example in this paper is a bit of a mixed bag. The paper's running example is introduced in the introduction, and consists of an Amazon Product Database with two tables, Product and Reivew. Where the example is used, I find it helpful. However, I think the authors created a problem for themselves by choosing a fairly complex example. The output attribute in the examples is generally the average Rating of a product, which depends on *eight* other attributes. There are also a lot of tuples in the Amazon Product Database for it to be a really "small" example. The complexity really starts to show in Figure 3 when the authors depict the DAG for the Amazon Product Database's PRCM. They are only able to show a partial DAG because of how many attributes there are.

I do appreciate that this is a particularly difficult balance to get right. I think an example that properly demonstrates *all* of the things going on when a hypothetical query is computed needs a certain level of complexity - i.e. multiple reviews for the same product, multiple products of the same type, etc. But an example that's too complex becomes difficult to follow for a reader and difficult to utilize for an author in a paper because of all the space they would have to devote to it. In this paper it seems that after a certain point the authors were only using the running example in situations where an example could be adequately gotten across in a brief paragraph.

F. Presentation of Results

The authors perform what I think are a thorough series of tests of HypeR and some variations on it, and present them in a logical order. They consider HypeR's performance from multiple angles, as well as ways the performance can be improved.

The "Solution Quality Comparison" and "Runtime Analysis and Comparison" where they look at how good HypeR's results actually are and how long it takes to get those results are the types of evaluation I would expect, and they go pretty deeply into those evaluations. The evaluation of solution quality doesn't just include the comparison of HypeR's what-if and how-to results with ground truth. There is also an examination of how how-to query results are affected by the granularity of discretization of the continuous update variables. This is not a test I was thinking about when reading the paper and it produced some very instructive results - that HypeR could handle finer discretization and produce similar results to the ground truth method in a much shorter amount of time. Runtime testing is approached with similar throroughness, obviously examining the effect of database size on the runtime, but also the effect of the size of the backdoor set and query complexity.

What I found most interesting was how they examined HypeR on real-world datasets ("What-If Real World Use Cases"). They found real-world datasets with dependencies that had already been studied and understood, and they checked that when they used HypeR to modify attributes with what-if queries, the results of those modifications made sense! Written out like that it seems pretty straightforward and obvious, but I think it can be easy to feel lost if you don't have something like a ground truth to compare results to. This is a great example of another way a solution can be validated.

G. Final Thoughts

Having done all my complaining, I feel the need to reiterate that I think this paper is overall very very good. I certainly am not at a point where I could explain such a complicated piece of work as HypeR is anywhere near as clearly as its authors do here. It's instructive as a reader and a writer to consider what parts of a paper seem to work and what don't, even if it is just to become aware of a personal bias or preference. It is likewise always instructive as a researcher to see how people are approaching problems and evaluating their solutions. Here the problem identified by the authors is the lack of a system for performing hypothetical updates on databases that have probabilistic dependencies among their attributes that actually

accounts for those probabilistic dependencies. I of course don't know what the actual workflow was here, but the paper does sort of hint to how the problem was broken down into smaller problems until they were all solved. For example, there's the question of what the result of a what-if query should even be and the decision to reduce several output attributes by aggregating and taking the expectation value. Then there's the question of how the expectation values over possible worlds can be computed efficiently, and the dual-pronged answer of block-independent decomposition and reducing the post-update distribution to the pre-update distribution - which itself introduces a new efficiency problem that has to be solved. There are obviously a lot of decisions that had to be made here, a lot of compromises and assumptions that rely on the authors understanding of the problem and the underlying concepts to eventually produce a working solution. It's really quite impressive.

As I said in the beginning of my critique, I do think I have learned and benefited a great deal from sitting with this paper for so long. I also might be happy to never see it again.