HADAD: A Lightweight Approach for Optimizing Hybrid Complex Analytics Queries

Anonymous Author(s)

ABSTRACT

Hybrid complex analytics workloads typically include (*i*) data management tasks (joins, selections, etc.), easily expressed using relational algebra (RA)-based languages, and (*ii*) complex analytics tasks (regressions, matrix decompositions, etc.), mostly expressed in linear algebra (LA) expressions. Such workloads are common in many application areas, including scientific computing, web analytics, and business recommendation. Existing solutions for evaluating hybrid analytical tasks – ranging from LA-oriented systems, to relational systems (extended to handle LA operations), to hybrid systems – either optimize data management and complex tasks separately, exploit RA properties only while leaving LA-specific optimization opportunities unexploited, or focus heavily on low-level physical optimization, and leaving semantic query optimization opportunities unexplored. Additionally, they are not able to exploit *precomputed (materialized) results* to avoid computing again (part of) a given mixed (LA and/or RA) computation.

In this paper, we take a major step towards filling this gap by proposing *a unified semantic query optimization framework* for such a hybrid setting. We describe HADAD, an extensible lightweight approach for optimizing hybrid complex analytics queries, based on a common abstraction that facilitates a unified reasoning: *a relational model endowed with integrity constraints*, which can be used to express the properties of the two computation formalisms. Our solution can be naturally and portably applied on top of cross RA-LA (or LA) platforms; while their existing code need not be rewritten. An extensive empirical evaluation shows that HADAD yields significant performance gains on diverse workloads, from LA-centered ones to hybrid ones.

CCS CONCEPTS

• **Data management systems** → *Semantic query optimization.*

KEYWORDS

linear algebra, query rewriting, integrity constraints, chase

ACM Reference Format:

Anonymous Author(s). 2021. HADAD: A Lightweight Approach for Optimizing Hybrid Complex Analytics Queries. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 62 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Modern analytical tasks typically include (*i*) data management tasks (e.g., joins, filters) to perform pre-processing steps, including feature selection, transformation, and engineering [20, 24, 37, 45, 47], tasks that are easily expressed using RA-based languages, as well as (*ii*) complex analytics tasks (e.g., regressions, matrices decompositions), which are mostly expressed using LA operations [36]. To perform such analytical tasks, data scientists can choose from a variety of systems, tools, and languages. Languages/libraries such as R [8] and NumPy [7], as well as LA systems such as SystemML [21], TensorFlow [13] and MLlib [10] treat matrices and LA operations as first-class citizens: they offer a rich set of built-in LA operations. However, it can be difficult to express pre-processing tasks in these systems. Further, expression rewrites, based on equivalences that hold due to well-known LA properties, are not exploited in some of these systems, leading to missed optimization opportunities.

Many works propose integrating RA and LA processing , where both algebraic styles can be used together [9, 23, 27, 35, 38, 40, 49]. [9] offers calling LA packages through user defined functions (UDFs), where libraries such as NumPy are embedded in the host language. Others suggest to extend RDBMS to treat LA objects as first-class citizens by using built-in functions to express LA operations [35, 40]. However, LA operations' semantics remain hidden behind these functions, where the optimizers treat as black-boxes. SPORES [49] and SPOOF [23] optimize LA expressions by converting them into RA, optimizing the latter, and then converting the result back to an (optimized) LA expression. They only focus on optimizing LA pipelines containing operations that can be expressed in RA. The restriction is that LA properties of complex operations such as inverse, matrix-decompositions are entirely unexploited. Morpheus [27] speeds up LA pipelines over large joins by pushing computation into each joined table, thereby avoiding expensive materialization. LARA [38] focuses on *low-level* optimization by exploiting data layouts (e.g., column-wise) to choose LA operators' physical implementations.

A limitation of such approaches is that they lack *high-level reasoning about LA properties and rewrites*, which can drastically enhance the pipelines' performance [48]. Further, they do not support *semantic query optimization*, which includes exploiting integrity constraints and materialized views and can bring enormous performance advantages in hybrid RA-LA and even plain LA settings.

We propose HADAD, an extensible lightweight framework for providing semantic query optimization (including views-based, integrity constraint-based, and LA property-based rewriting) on top of both pure LA and hybrid RA-LA platforms, with no need to modify their internals. At the core of HADAD lies a common abstraction: *relational model with integrity constraints*, which enables reasoning in hybrid settings. Moreover, it makes it very easy to extend HADADś semantic knowledge of LA operations by simply declaring appropriate constraints, with no need to change HADAD

code. As we show, constraints are sufficiently expressive to declare (and thus allow HADAD to exploit) more properties of LA operations than previous work could consider.

Last but not least, our holistic, cost-based approach enables to judiciously apply for each query *the best available optimization*. For instance, given the computation $M(NP)$ for some matrices M, N and P, we may rewrite it into $(MN)P$ if its estimated cost is smaller than that of the original expression, *or* we may turn it into MV if a materialized view V stores exactly the result of (NP).

HADAD capitalizes on a framework previously introduced in [15] for rewriting queries across many data models, using materialized views, in a polystore setting that does not include the LA model. The novelty of HADAD is to extend the benefits of rewriting and views optimizations to pure LA and hybrid RA-LA computations, which are crucial for ML workloads.

Contributions. The paper makes the following contributions:

❶ We propose an *extensible lightweight approach to optimize hybrid complex analytics queries*. Our approach can be implemented on top of existing systems without modifying their internals; it is based on a powerful intermediate abstraction that supports reasoning in hybrid settings, namely *a relational model with integrity constraints*.

❷ We formalize the problem of *rewriting computations using previously materialized views in hybrid settings*. To the best of our knowledge, ours is the first work that brings views-based rewriting under integrity constraints in the context of LA-based pipelines and hybrid analytical queries.

❸ We provide *formal guarantees* for our solution in terms of soundness and completeness.

❹ We conduct an extensive set of empirical experiments on typical LA- and hybrid-based expressions, which show the benefits of HADAD.

Outline. The rest of this paper is organized as follows: §2 highlights HADAD's optimizations that go beyond the state of the art based on *real-world* scenarios, §3 formalizes the query optimization problem in the context of a hybrid setting. §5 provides an end-to-end overview of our approach. §6 presents our novel reduction of the rewriting problem into one that can be solved by existing techniques from the relational setting. §7 describes our extension to the query rewriting engine, integrating two different cost models, to help prune out inefficient rewritings as soon as they are enumerated. We formalize our solution's guarantees in §8 and present the experiments in §9. We discuss related work and conclude in §10.

2 HADAD OPTIMIZATIONS

We highlight below examples of performance-enhancing opportunities that are exploited by HADAD and not being addressed by LA-oriented and cross RA-LA existing solutions.

LA Pipeline Optimization. Consider the Ordinary-Least Regression (OLS) pipeline: $(X^TX)^{-1}(X^Ty)$, where X is a square matrix of size 10K×10K and y is a vector of size 10K×1. Suppose available a materialized view $V = X^{-1}$. HADAD rewrites the pipeline to $(V(V^T(X^Ty))$, by exploiting the LA properties $(CD)^{-1} = D^{-1}C^{-1}$, $(CD)E = C(DE)$ and $(D^T)^{-1} = (D^{-1})^T$ as well as the view V. The rewriting is more efficient than the original pipeline since it avoids computing the expensive inverse operation. Moreover, it optimizes the matrix chain multiplication order to minimize the intermediate result size. This leads to a 150× speed-up on MLlib [42]. Current popular LA-oriented systems [7, 8, 13, 21, 42] are not capable of exploiting such rewrites, due to the lack of systematic exploration of standard LA properties and views.

Hybrid RA-LA Optimization. Cross RA-LA platforms such as Morpheus [27], SparkSQL [17] and others [34, 38] can greatly benefit from HADAD's cross-model optimizations, which can find these platforms' rewrites but also additional ones that they miss.

Factorization of LA Operations over Joins. For instance, Morpheus implements a powerful optimization that *factorizes an LA operation on a matrix* $\boldsymbol{M}$ *obtained by joining tables* $\boldsymbol{R}$ *and* $\boldsymbol{S}$ *and casting the join result as a matrix. Factorization pushes the LA operation before the join, to operate on* $\boldsymbol{R}$ *and* $\boldsymbol{S}$*, cast as matrices.*

Consider a specific instantiation of factorization: `colSums`($\boldsymbol{M}N$)*, where matrix* $\boldsymbol{M}$ *has size 20M× 120 and N has size 120×100; both matrices are dense. The LA operation is* `colSums`*, which sums up the elements in each column, returning the vector of these sums (the operation is common in ML-relevant algorithms such as K means-clustering [41]). On this pipeline, Morpheus applies a multiplication factorization rule to push the multiplication by N down to* $\boldsymbol{R}$ *and* $\boldsymbol{S}$*: it computes* $\boldsymbol{R}N$ *and* $\boldsymbol{S}N$*, then concatenates the resulting matrices to obtain* $\boldsymbol{M}N$*. This intermediate result is materialized as a dense 20M×100 matrix of size ≈16GB (Morpheus is prototyped on R, which uses double precision). Finally,* `colSums` *is applied to the intermediate result, reducing it to a vector of size 1×100.*

HADAD can help Morpheus do much better, by pushing the `colSums` *operator to* $\boldsymbol{R}$ *and* $\boldsymbol{S}$ *(instead of the multiplication with N), then concatenating the resulting vectors. This leads to much smaller intermediate results, since the combined size of vectors* `colSums`($\boldsymbol{R}$) *and* `colSums`($\boldsymbol{S}$) *is only 1×120.*

To this end, HADAD rewrites the pipeline to `colSums`($\boldsymbol{M}$)N *by exploiting the LA property* `colSums`(AB) =`colSums`$(A)B$ *and applying its cost estimator, which favors rewritings with a small intermediate result size. Evaluating this HADAD-produced rewriting, Morpheus's multiplication pushdown rule no longer applies, while the* `colSums` *pushdown rule is now enabled, leading to 125× speed-up.*

Pushing Selection from LA Analysis to RA Preprocessing. Consider another hybrid example on a Twitter dataset [12]. The JSON dataset contains tweet ids, extended tweets, entities including hashtags, filter-level (e.g., sensitive level), media, URL, and tweet text, etc. We implemented it on SparkSQL (with SystemML [21]).

In the preprocessing stage, our SparkSQL query constructs a tweet-hashtag filter-level matrix $\boldsymbol{N}$ *of size 2M×1000, for all tweets posted from "USA" mentioning "covid", where rows are tweets, columns are hashtags, and values are filter-levels.*

$\boldsymbol{N}$ *is then loaded into SystemML, where rows with filter-level less than 4 are selected. The result undergoes an Alternating Least Square (ALS) [44] computation in order to predict, given a set of tweets and hashtags, the sensitivity level of these tweets.*

A core building block of the ALS computation is the LA pipeline $(uv^T - \boldsymbol{N})v$*. In our example,* u *is a tweet feature vector (of size 2M×1) and* v *is a hashtag feature vector (1000×1).*

We have two materialized views available: V_1 stores the tweet id and text as a text datasource in Solr, *and V_2 stores tweet id, hashtag id, and filter-level for all tweets posted from "USA", and is materialized on disk as CSV file. The rewriting* modifies the preprocessing *of $\boldsymbol{N}$ by introducing V_1 and V_2; it also* pushes the filter-level selection *from the LA pipeline into the preprocessing stage. To this end, it rewrites $(uv^T - \boldsymbol{N})v$ to $uv^Tv - \boldsymbol{N}v$, which is more efficient for two reasons. First, $\boldsymbol{N}$ is ultra sparse (0.00018% non-zero), which renders the computation of $\boldsymbol{N}v$ extremely efficient. Second, SystemML evaluates the chain uv^Tv efficiently, computing v^Tv first, which results in a scalar, instead of computing uv^T, which results in a dense matrix of size 2M×1000 (HADAD's cost model realizes this). Without the rewriting help from HADAD, SystemML is unable to exploit its own efficient operations for lack of awareness of the distributivity property of vector multiplication over matrix addition, $Av + Bv = (A + B)v$. The rewriting achieves 14× speed-up.*

HADAD detects and applies all the above-mentioned optimizations combined. It captures RA-, LA-, and cross-model optimizations precisely because it reduces all rewrites to a single setting in which they can synergize: relational rewrites under integrity constraints.

3 PROBLEM STATEMENT

We consider a set of *value domains* $\mathcal{D}_i$, e.g., $\mathcal{D}_1$ denotes integers, $\mathcal{D}_2$ denotes real numbers, $\mathcal{D}_3$ strings, etc, and two basic data types: *relations (sets of tuples)* and *matrices* (bi-dimensional arrays). Any attribute in a tuple or cell in a matrix is a value from some $\mathcal{D}_i$. We assume *a matrix can be implicitly converted into a relation* (the order among matrix rows is lost), and *the opposite conversion* (each tuple becomes a matrix line, in some order that is unknown, unless the relation was explicitly sorted before the conversion).

We consider a hybrid language $\mathcal{L}$, comprising a set R_{ops} of (unary or binary) *RA operators*; concretely, R_{ops} comprises the standard relational selection, projection, and join. We also consider a set L_{ops} of *LA operators*, comprising: unary (e.g., inversion and transposition) and binary (e.g., matrix product) operators The full set L_{ops} of LA operations we support is detailed in §6.1. A *hybrid expression* in $\mathcal{L}$ is defined as follows:

- any value from a domain $\mathcal{D}_i$, any matrix, and any relation, is an expression;
- (RA operators): given some expressions E, E', $ro_1(E)$ is also an expression, where $ro_1 \in R_{ops}$ is a unary relational operator, and E's type matches ro_1's expected input type. The same holds for $ro_2(E, E')$, where $ro_2 \in R_{ops}$ is a binary relational operator (i.e., the join);
- (LA operators): given some expressions E, E' which are either numeric matrices or numbers (which can be seen as degenerate matrices of 1×1), and some real number r, the following are also expressions: $lo_1(E)$ where $lo_1 \in L_{ops}$ is a unary operator, and $lo_2(E, E')$ where $lo_2 \in L_{ops}$ is a binary operator (again, provided that E, E' match the expected input types of the operators).

Clearly, an important set of *equivalence rules* hold over our hybrid expressions, well-known respectively in the RA and the LA literature. These equivalences lead to *alternative evaluation strategies* for each expression. Further, we assume given a (possibly empty) set of *materialized views* $\boldsymbol{V} \in \mathcal{L}$, which have been previously computed over some inputs (matrices and/or relations), and whose results are directly available (e.g., as a file on disk). Detecting when a materialized view can be used instead of evaluating (part of) an expression is another important source of alternative evaluation strategies.

Given an expression E and a *cost model* that assigns a cost (a real number) to an expression, we consider the problem of **identifying the most efficient rewrite** derived from E by: (i) exploiting RA and LA equivalence rules, and/or (ii) replacing part of an expression with a scan of a materialized view equivalent to that expression.

Below, we detail our approach, the equivalence rules we capture, and two alternative cost models we devised for this hybrid setting. Importantly, our solution (based on a *relational encoding with integrity constraints*) capitalizes on the framework previously introduced in [15], where it was used to *rewrite queries using materialized views in a polystore setting*, where the data, views, and query cover a variety of data models (relational, JSON, XML, etc.). Those queries can be expressed in a combination of query languages, including SQL, JSON query languages, XQuery, etc. **The ability to rewrite such queries using heterogeneous views directly and fully transfers to HADAD**: thus, instead of a relation, we could have the (tuple-structured) results of an XML or JSON query; views materialized by joining an XML document with a JSON one and a relational database could also be reused. The novelty of our work is to **extend the benefits of rewriting and view-based optimization to LA computations, crucial for ML workloads**. In §6, we focus on capturing matrix data and LA computations in the relational framework, along with relational data naturally; this enables our novel, holistic optimization of hybrid expressions.

4 PRELIMINARIES

We recall conjunctive queries [26], integrity constraints [14], and query rewriting under constraints [32]; these concepts are at the core of our approach.

4.1 Conjunctive Query and Constraints

A conjunctive query (or simply CQ) Q is an expression of the form $Q(\overline{x})$:- $R_1(\overline{y}_1), \ldots, R_n(\overline{y}_n)$, where each R_i is a predicate (relation) of some finite arity, and $\overline{x}, \overline{y}_1, \ldots, \overline{y}_n$ are tuples of variables or constants. Each $R_i(\overline{y}_i)$ is called a relational atom. The expression $Q(\overline{x})$ is the *head* of the query, while the conjunction of relational atoms $R_1(\overline{y}_1), \ldots, R_n(\overline{y}_n)$ is its *body*. All variables in the head are called *distinguished*. Also, every variable in $\overline{x}$ must appear at least once in $\overline{y}_1, \ldots, \overline{y}_n$. Different forms of constraints have been studied in the literature [14]. In this work, we use Tuple Generating Dependencies (**TGD**s) and Equality Generating Dependencies (**EGD**s), stated by formulas of the form $\forall x_1, \ldots x_n\ \phi(x_1, \ldots x_n) \rightarrow \exists z_1, \ldots, z_k\ \psi(y_1, \ldots, y_m)$, where $\{z_1, \ldots, z_k\} = \{y_1, \ldots, y_m\} \backslash \{x_1, \ldots, x_n\}$. The constraint's *premise* ϕ is a possibly empty conjunction of relational atoms over variables $x_1, \ldots, x_n$ and possibly constants. The constraint's *conclusion* ψ is a non-empty conjunction of atoms over variables $y_1, \ldots, y_m$ and possibly constants, atoms that are relational ones in the case of **TGD**s or equality atoms – of the form $w = w'$ – in the case of **EGD**s. For instance, consider a relation *Review(paper, reviewer, track)* listing reviewers of papers submitted to a conference's tracks, and a relation *PC (member, affiliation)* listing the affiliation of every program committee member [29].The fact that a

paper can only be submitted to a single track is captured by the following EGD: $\forall p \forall r \forall t \forall r' \forall t' Review(p, r, t) \wedge Review(p, r', t') \rightarrow t = t'$. We can also express that papers can be reviewed only by PC members by the following TGD: $\forall p \forall r \forall t\ Review(p, r, t) \rightarrow \exists a\ PC(r, a)$.

4.2 Provenance-Aware Chase & Back-Chase

A key ingredient leveraged in our approach is relational query rewriting using views, in the presence of constraints. The state-of-the-art method for this task, called Chase & Backchase, was introduced in [30] and improved in [32], as the Provenance-Aware Chase & Back-Chase (**PACB** in short). At the core of these methods is the idea to *model views as constraints*, in this way reducing the view-based rewriting problem to constraints-only rewriting. Specifically, for a given view V defined by a query, the constraint V_{IO} states that *for every match of the view body against the input data, there is a corresponding (head) tuple in the view output*, while the constraint V_{OI} states the converse inclusion, i.e., *each view output tuple is due to a view body match*. From a set $\mathcal{V}$ of view definitions, PACB therefore derives a set of view constraints $C_{\mathcal{V}} = \{V_{IO}, V_{OI} \mid V \in \mathcal{V}\}$.

Given a source schema σ with a set of integrity constraints $\mathcal{I}$, a set $\mathcal{V}$ of views defined over σ, and a conjunctive query Q over σ, the **rewriting problem** thus becomes: find every reformulation query ρ over the schema of view names $\mathcal{V}$ that is equivalent to Q under the constraints $\mathcal{I} \cup C_{\mathcal{V}}$.

Example 4.1. *For instance, if $\sigma = \{R, S\}$, $\mathcal{I} = \emptyset$, $\tau = \{V\}$ and we have a view V materializing the join of relations R and S, $V(x, y)$:- $R(x, z), S(z, y)$, the pair of constraints capturing V is the following:*

$$V_{IO}: \quad \forall x \forall z \forall y\ R(x, z) \wedge S(z, y) \rightarrow V(x, y)$$
$$V_{OI}: \quad \forall x \forall y V(x, y) \rightarrow \exists z\ R(x, z) \wedge S(z, y).$$

Given the query $Q(x, y)$:- $R(x, z), S(z, y)$, PACB finds the reformulation $\rho(x, y)$:- $V(x, y)$. Algorithmically, this is achieved by:

(i) chasing Q with the constraints $\mathcal{I} \cup C_{\mathcal{V}}^{IO}$, where $C_{\mathcal{V}}^{IO} = \{V_{IO} \mid V \in \mathcal{V}\}$; intuitively, this enriches (extends) Q with all the consequences that follow from its atoms and the constraints $\mathcal{I} \cup C_{\mathcal{V}}^{IO}$.

(ii) restricting the chase result to only the $\mathcal{V}$-atoms; the result is called the universal plan *U.*

(iii) annotating each atom of the universal plan U with a unique ID called a provenance term.

(iv) chasing U with the constraints in $\mathcal{I} \cup C_{\mathcal{V}}^{OI}$, where $C_{\mathcal{V}}^{OI} = \{V_{OI} \mid V \in \mathcal{V}\}$, and annotating each relational atom a introduced by these chase steps with a provenance formula[1] *$\pi(a)$, which gives the set of U-subqueries whose chasing led to the creation of a; the result of this phase, called the* backchase*, is denoted B.*

(v) matching Q against B and outputting as rewritings the subsets of U that are responsible for the introduction (during the backchase) of the atoms in the image $h(Q)$ of Q; these rewritings are read off directly from the provenance formula $\pi(h(Q))$.

In our example, $\mathcal{I}$ is empty, $C_{\mathcal{V}}^{IO} = \{V_{IO}\}$, and the result of the chase in phase (i) is $Q_1(x, y)$:- $R(x, z), S(z, y), V(x, y)$. The universal plan obtained in (ii) by restricting Q_1 to the schema of view names is $U(x, y)$:- $V(x, y)^{p_0}$, where p_0 denotes the provenance term of atom $V(x, y)$. The result of backchasing U with $C_{\mathcal{V}}^{OI}$ in phase (iv) is $B(x, y)$:- $V(x, y)^{p_0}$, $R(x, z)^{p_0}$, $S(z, y)^{p_0}$. Note that the provenance formulas of the R and S atoms (a simple term, in this example) are introduced by chasing the view V. Finally, in phase (v) we find one match image given by h from Q's body into the R and S atoms from B's body. The provenance formula $\pi(h(Q))$ of the image h is p_0, which corresponds to an equivalent rewriting $\rho(x, y)$:- $V(x, y)$.

[1]Provenance formulas are constructed from provenance terms using logical conjunction and disjunction.

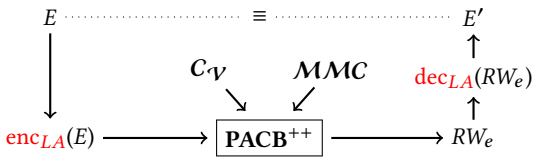


Figure 1: Outline of Our Reduction

5 HADAD OVERVIEW

We outline here our approach as an extension to [15] for solving the rewriting problem introduced in §3.

Hybrid Expressions and Views. A hybrid expression (whether asked as a query, or describing a materialized view) can be purely relational (RA), in which case we assume it is specified as a conjunctive query [26]. Other expressions are purely LA ones; we assume that they are defined in a dedicated LA language such as R [8], DML [21], etc. , using LA operators from our set L_{ops} (see §6.1), commonly used in real-world ML workloads. Finally, a hybrid expression can combine RA and LA, e.g., an RA expression (resulting in a relation) is treated as a matrix input by an LA operator, whose output may be converted again to a table and joined further, etc.

Our approach is based on a reduction to a relational model. Below, we show how to bring our hybrid expressions - and, most specifically, their LA components - under a relational form (the RA part of each expression is already in the target formalism).

Encoding into a Relational Model. Let E be an LA expression (query) and $\mathcal{V}$ be a set of materialized views. We reduce the LA-views based rewriting problem to the relational rewriting problem under integrity constraints, as follows (see Figure 1). First, we *encode relationally* E, $\mathcal{V}$, and the set L_{ops} of LA operators. Note that the relations used in the encoding are *virtual* and *hidden*, i.e., invisible to both the application designers and users. They only serve to support query rewriting via relational techniques.

These virtual relations are accompanied by a set of relational *integrity constraints* $enc_{LA}(LA_{prop})$ that reflect a set LA_{prop} of LA properties of the supported L_{ops} operations. For instance, we model the *matrix addition* operation using a relation $\texttt{add}_M(M, N, R)$, denoting that R is the result of $M+N$, together with a set of constraints stating that $\texttt{add}_M$ is a *functional* relation that is *commutative*, *associative*, etc. These constraints are Tuple Generating Dependencies (**TGD**s) or Equality Generating Dependencies (**EGD**s) [14], which are a generalization of key and foreign key dependencies. We detail our relational encoding in §6.

Reduction from LA-based to Relational Rewriting. Our reduction translates the declaration of each view $V \in \mathcal{V}$ to constraints $enc_{LA}(V)$ that reflect the correspondence between V's input data and its output. Separately, E is also encoded as a relational query $enc_{LA}(E)$ over the relational encodings of L_{ops} and its matrices.

Operation	Encoding	Operation	Encoding	Operation	Encoding
Matrix scan	$name(M, n)$	Inversion	$\texttt{inv}_M(M, R)$	Cells sum	$\texttt{sum}(M, s)$
Multiplication	$\texttt{multi}_M(M, N, R)$	Scalar Multiplication	$\texttt{multi}_{MS}(s, M, R)$	Row sum	$\texttt{rowSums}(M, R)$
Addition	$\texttt{add}_M(M, N, R)$	Determinant	$\texttt{det}(M, R)$	Colsums	$\texttt{colSums}(M, R)$
Division	$\texttt{div}_M(M, N, R)$	Trace	$\texttt{trace}(M, s)$	Direct sum	$\texttt{sum}_D(M, N, R)$
Hadamard product	$\texttt{multi}_E(M, N, R)$	Exponential	$\texttt{exp}(M, R)$	Direct product	$\texttt{dirct}_D(M, N, R)$
Transposition	$\texttt{tr}(M, R)$	Adjoints	$\texttt{adj}(M, R)$	Diagonal	$\texttt{diag}(M, R)$

Table 1: Snippet of the $\mathcal{VREM}$ Schema

Now, the reformulation problem is reduced to a purely relational setting, as follows. We are given a relational query $enc_{LA}(E)$ and a set $\mathcal{C}_\mathcal{V} = enc_{LA}(V_1) \cup \ldots \cup enc_{LA}(V_n)$ of relational integrity constraints encoding the views $\mathcal{V}$. We add as further input a set of relational constraints $enc_{LA}(LA_{prop})$, we called them Matrix-Model Encoding constraints, or $\mathcal{MMC}$ in short. We must find the rewritings RW_r^i expressed over the relational views $\mathcal{C}_\mathcal{V}$ and $\mathcal{MMC}$, for some integer k and $1 \leq i \leq k$, such that each RW_r^i is equivalent to $enc_{LA}(E)$ under these constraints ($\mathcal{C}_\mathcal{V} \bigcup \mathcal{MMC}$). Solving this problem yields a *relationally encoded rewriting* RW_e expressed over the (virtual) relations used in the encoding; a final *decoding* step is needed to obtain E', the rewriting of (LA or, more generally, hybrid) E using the views $\mathcal{V}$.

The challenge in coming up with the reduction consists in designing an encoding, i.e., one in which rewritings found by (*i*) encoding relationally, (*ii*) solving the resulting relational rewriting problem, and (*iii*) decoding a resulting rewriting over the views, is guaranteed to produce an equivalent expression E' (see §6).

Relational Rewriting Using Constraints. To solve the relational rewriting problem under constraints, the engine of choice is Provenance Aware Chase&Backchase (PACB) [32]. Our PACB rewriting engine ($PACB^{++}$ hereafter) has been extended to utilize the $Pruned_{prov}$ algorithm sketched and discussed in [31, 32], which prunes inefficient rewritings during the search phase, based on a simple cost model (see §7).

Decoding of the Relational Rewriting. For the selected relational reformulation RW_e by $PACB^{++}$, a *decoding step* $dec(RW_e)$ is performed to translate RW_e into the native syntax of its respective underlying store/engine (e.g., R, DML, etc.).

6 REDUCTION TO THE RELATIONAL MODEL

Our internal model is relational, and it makes prominent use of expressive integrity constraints. This framework suffices to describe the features and properties of most data models used today, notably including relational, XML, JSON, graph, etc [15, 16].

Going beyond, in this section, we present a novel way to *reason relationally about LA primitives/operations* by treating them as uninterpreted functions with black-box semantics, and adding *constraints that capture their important properties*. First, we give an overview of a wide range of LA operations that we consider in (§6.1). Then, in (§6.2), we show how matrices and their operations can be represented (*encoded*) using a set of *virtual* relations, part of a schema we call $\mathcal{VREM}$ (for *Virtual Relational Encoding of Matrices*), together with the integrity constraints $\mathcal{MMC}$ that capture the LA properties of these operations. Regardless of matrix data's physical storage, we only use $\mathcal{VREM}$ to encode LA expressions and views relationally to reason about them. (§6.3) exemplifies relational rewritings obtained via our reduction.

6.1 Matrix Algebra

We consider a wide range of matrix operations [18, 39], which are common in real-world machine learning algorithms [3]: element-wise multiplication (i.e., Hadamard-product) ($\texttt{multi}_E$), matrix-scalar multiplication ($\texttt{multi}_{MS}$), matrix multiplication ($\texttt{multi}_M$), addition ($\texttt{add}_M$), division ($\texttt{div}_M$), transposition (`tr`), inversion ($\texttt{inv}_M$), determinant (`det`), trace (`trace`), diagonal (`diag`), exponential (`exp`), adjoints (`adj`), direct sum ($\texttt{sum}_D$), direct product ($\texttt{dirct}_D$), summation (`sum`), rows and columns summation (`rowSums`, `colSums`, respectively), QR decomposition (`QR`), Cholesky decomposition `CHO`), LU decomposition (`LU`), and pivoted LU decomposition (`LUP`).

6.2 VREM Schema and Relational Encoding

To model LA operations on the $\mathcal{VREM}$ relational schema (part of which appears in Table 1), we also rely on a set of integrity constraints $\mathcal{MMC}$, which are encoded using relations in $\mathcal{VREM}$. We detail the encoding below.

6.2.1 ***Base Matrices and Dimensionality Modeling.*** We denote by $\mathrm{M}_{k\times z}(\mathcal{D})$ a matrix of k rows and z columns, whose entries (values) come from a domain $\mathcal{D}$, e.g., the domain of real numbers $\mathbb{R}$. For brevity we just use $\mathrm{M}_{k\times z}$. We define a virtual relation $name(M, n) \in \mathcal{VREM}$ attaching a unique ID M to any matrix identified by a name denoted n (which may be e.g. of the form "/data/M.csv"). This relation (shown at the top left in Table 1) is accompanied by an EGD key constraint $\mathcal{I}_{name} \in \mathcal{MMC}_m$, where $\mathcal{MMC}_m \subset \mathcal{MMC}$, and $\mathcal{I}_{name}$ states that *two matrices with the same name n have the same ID*:

$$\mathcal{I}_{name}\colon \forall M \forall N\ name(M, n) \wedge name(N, n) \rightarrow M = N$$

Note that the matrix ID in *name* (and all the other virtual relations used in our encoding) are not IDs of individual matrix objects: rather, each identifies an *equivalence class* (induced by value equality) of expressions. That is, two expressions are assigned the same ID iff they yield value-based-equal matrices. In Table 1, we use M and N to denote an operation's input matrices' IDs and R for the resulting matrix ID, and s for scalar's input and output.

The dimensions of a matrix are captured by a $size(M, k, z)$ relation, where k and z are the number of rows, resp. columns and M is an ID. An EGD constraint $\mathcal{I}_{size} \in \mathcal{MMC}_m$ holds on the *size* relation, stating that the ID determines the dimensions:

$$\mathcal{I}_{size}\colon \forall M \forall k_1 \forall z_1 \forall k_2 \forall z_2$$
$$size(M, k_1, z_1) \wedge size(M, k_2, z_2) \rightarrow k_1 = k_2 \wedge z_1 = z_2$$

The identity I and zero O matrices are captured by the $Zero(O)$ and $Identity(I)$ relations, where O and I denote their IDs, respectively. They are accompanied by EGD constraints $\mathcal{I}_{iden}, \mathcal{I}_{zero} \in \boldsymbol{\mathcal{MMC}}_m$, stating that zero matrices with the same sizes have the same IDs, and this also applies for identity matrices with the same size:

$$\mathcal{I}_{zero}: \forall O_1 \forall O_2 \forall k \forall z$$
$$Zero(O_1) \wedge size(O_1, k, z) \wedge Zero(O_2) \wedge size(O_2, k, z) \rightarrow O_1 = O_2$$
$$\mathcal{I}_{iden}: \forall I_1 \forall I_2$$
$$Identity(I_1) \wedge size(I_1, k, k) \wedge Identity(I_2) \wedge size(I_2, k, k) \rightarrow I_1 = I_2$$

6.2.2 ***Encoding Matrix Algebra Expressions.*** LA operations are encoded into dedicated relations, as shown in Table 1. We now illustrate the encoding of an LA expression on the $\boldsymbol{\mathcal{VREM}}$ schema.

Example 6.1. *Consider the LA expression E: $((MN)^T)$, where the two matrices $M_{100\times1}$ and $N_{1\times10}$ are stored as "M.csv" and "N.csv", respectively. The encoding function $enc_{LA}(E)$ takes as argument the LA expression E and returns a conjunctive query whose: (i) body is the relational encoding of E using $\boldsymbol{\mathcal{VREM}}$ (see below), and (ii) head has one distinguished variable, denoting the equivalence class of the result. For instance:*

$enc(((MN)^T)$ =
  Let $enc(MN)$ =
    Let $enc(M) = \boldsymbol{Q}_0(M)$:- $name(M, \text{"}M.csv\text{"})$;
      $enc(N) = \boldsymbol{Q}_1(N)$:- $name(N, \text{"}N.csv\text{"})$;
      R_1 = freshId()
    in
  $\boldsymbol{Q}_2(R_1)$:- $\texttt{multi}_M(M, N, R_1) \wedge \boldsymbol{Q}_0(M) \wedge \boldsymbol{Q}_1(N)$;
  R_2 = freshId()
  in
$\boldsymbol{Q}(R_2)$:- $\texttt{tr}(R_1, R_2) \wedge \boldsymbol{Q}_2(R_1)$;

In the above, nesting is dictated by the syntax of E. From the inner (most indented) to the outer, we first encode M and N as small queries using the name relation, then their product (to whom we assign the newly created identifier R_1), using the $multi_M$ relation and encoding the relationship between this product and its inputs in the definition of $\boldsymbol{Q}_2(R_1)$. Next, we create a fresh ID R_2 used to encode the full E (the transposed of $\boldsymbol{Q}_2$) via relation tr, in the query $\boldsymbol{Q}(R_2)$. For brevity, we omit the matrices' size relations in this example and hereafter. Unfolding $\boldsymbol{Q}_2(R_1)$ in the body of $\boldsymbol{Q}$ yields:

$\boldsymbol{Q}(R_2)$:- $\texttt{tr}(R_1, R_2) \wedge \texttt{multi}_M(M, N, R_1) \wedge$
  $\boldsymbol{Q}_0(M) \wedge \boldsymbol{Q}_1(N)$;

Now, by unfolding $\boldsymbol{Q}_0$ and $\boldsymbol{Q}_1$ in $\boldsymbol{Q}$, we obtain the final encoding of $((MN)^T)$ as a conjunctive query $\boldsymbol{Q}$:

$\boldsymbol{Q}(R_2)$:- $\texttt{tr}(R_1, R_2) \wedge \texttt{multi}_M(M, N, R_1) \wedge$
  $name(M, \text{"}M.csv\text{"}) \wedge name(N, \text{"}N.csv\text{"})$;

6.2.3 ***Encoding LA Properties as Integrity Constraints.*** Figure 2 shows some of the constraints $\boldsymbol{\mathcal{MMC}}_{LA_{prop}} \subset \boldsymbol{\mathcal{MMC}}$, which capture textbook LA properties [18, 39] of our LA operations (§6.1). The TGDs (1), (2) and (3) state that matrix addition is commutative, matrix transposition is distributive with respect to addition, and the transposition of the inverse of matrix M is equivalent to the inverse of the transposition of M, respectively. We also express that the *virtual relations are functional* by using EGD key constraints. For example, the following $\mathcal{I}_{multi_M} \in \boldsymbol{\mathcal{MMC}}_{LA_{prop}}$ constraint

$$\forall M \forall N \forall R \ \texttt{add}_M(M, N, R) \rightarrow \texttt{add}_M(N, M, R) \quad (1)$$
$$\forall M \forall N \forall R_1 \forall R_2 \ \texttt{add}_M(M, N, R_1) \wedge \texttt{tr}(R_1, R_2) \rightarrow$$
$$\exists R_3 \exists R_4 \ \texttt{tr}(M, R_3) \wedge \texttt{tr}(N, R_4) \wedge \texttt{add}_M(R_3, R_4, R_2) \quad (2)$$
$$\forall M \forall R_1 \forall R_2 \ \texttt{inv}_M(M, R_1) \wedge \texttt{tr}(R_1, R_2) \rightarrow$$
$$\exists R_3 \ \texttt{tr}(M, R_3) \wedge \texttt{inv}_M(R_3, R_2) \quad (3)$$

Figure 2: $\boldsymbol{\mathcal{MMC}}$ Constraints Capturing Basic LA Properties

$$\forall M \forall N \forall R_1 \forall R_2 \forall R_3 \forall R_4$$
$$name(M, \text{"}M.csv\text{"}) \wedge \ name(N, \text{"}N.csv\text{"}) \wedge \ \texttt{tr}(N, R_1) \wedge$$
$$\texttt{tr}(M, R_2) \wedge \ \texttt{inv}_M(R_2, R_3) \wedge$$
$$\texttt{add}_M(R_1, R_3, R_4) \rightarrow name(R_4, \text{"}V.csv\text{"})$$

Figure 3: Relational Encoding of view V

states that $multi_M$ is functional, that is the products of pairwise equal matrices are equal.

$$\mathcal{I}_{multi_M}: \forall M \forall N \forall R_1 \forall R_2$$
$$\texttt{multi}_M(M, N, R_1) \wedge \texttt{multi}_M(M, N, R_2) \rightarrow R_1 = R_2$$

Other properties [18, 39] of the LA operations we consider are similarly encoded; due to space constraints, we delegate them to Appendix A [11].

6.2.4 ***Encoding LA Views as Constraints.*** We translate each view definition V (defined in LA language such as R, DML, etc) into relational constraints $enc_{LA}(V) \in \boldsymbol{C}_{\boldsymbol{\mathcal{V}}}$, where $\boldsymbol{C}_{\boldsymbol{\mathcal{V}}}$ is the set of relational constraints used to capture the views $\boldsymbol{\mathcal{V}}$. These constraints show how the view's inputs are related to its output over the $\boldsymbol{\mathcal{VREM}}$ schema. Figure 3 illustrates the encoding as a TGD constraint of the view $V : (N)^T + (M^T)^{-1}$ stored in a file "V.csv" and computed based on the matrices N and M (e.g., stored as "N.csv" and "M.csv", respectively).

6.2.5 ***Encoding Matrix Decompositions.*** Matrix decompositions play a crucial role in many LA computations. For instance, for every symmetric positive definite matrix M there exists a unique Cholesky Decomposition (CD) of the form $M = LL^T$, where L is a lower triangular matrix. We model CD, as well as other well-known decompositions (LU, QR, and Pivoted LU or PLU) as a set of virtual relations $\boldsymbol{\mathcal{VREM}}_{dec}$, which we add to $\boldsymbol{\mathcal{VREM}}$. For instance, to CD we associate a relation $\texttt{CHO}(M, L)$, which denotes that L is the output of the CD decomposition for a given matrix M whose ID is M. CHO is a functional relation, meaning every symmetric positive definite matrix has a unique CD decomposition. This functional aspect is captured by an EGD, conceptually similar to the constraint $\mathcal{I}_{multi_M}$ (§6.2.3). The property $M = LL^T$ is captured as a TGD constraint $I_{cho} \in \boldsymbol{\mathcal{MMC}}_{LA_{prop}}$:

$$\mathcal{I}_{cho}: \forall M \ type(M, \text{"}S\text{"}) \rightarrow \exists L_1 \exists L_2 \ cho(M, L_1) \wedge$$
$$type(L_1, \text{"}L\text{"}) \wedge tr(L_1, L_2) \wedge multi_M(L_1, L_2, M) \quad (4)$$

The atom $type(M,$"S") indicates the type of matrix M, where the constant "S" denotes a matrix that is *symmetric positive definite*; similarly, $type(L_1,$"L") denotes that the matrix L_1 is a lower triangular matrix. For each base matrix, its type (if available) (e.g., symmetric, upper triangular, etc.) is specified as TGD constraint. For example, we state that a certain matrix M (and any other matrix value-equal to M) is symmetric positive definite as follows:

$$\forall M\ name(M, \text{“}M.csv\text{”}) \rightarrow type(M, \text{“}S\text{”}) \quad (5)$$

Example 6.2. *Consider a view V=$N + LL^T$, where $L = cho(M)$ and M is a symmetric positive definite matrix encoded as in (5). Let E be the LA expression $M + N$. The reader realizes easily that V can be used to answer E directly, thanks to the specific property of the CD decomposition (4), and since $M + N = N + M$, which is encoded in (1). However, at the syntactic level, V and E are very dissimilar. Knowledge of (1) and (4) and the ability to reason about them is crucial in order to efficiently answer E based on V.*

The output matrix of CD decomposition is a lower triangular matrix L, which is not symmetric positive definite, meaning that CD decomposition can not be applied again on L. For other decompositions, such as $QR(M) = [Q, R]$ decomposition, where M is a square matrix, Q is an orthogonal matrix [39] and R is an upper triangular matrix, there exists a QR decomposition for the orthogonal matrix Q such that $QR(Q) = [Q, I]$, where I is an identity matrix and $QR(R) = [I, R]$. We say the *fixed point* of the QR decomposition is $QR(I) = [I, I]$. These properties of the Q decompositions are captured with the following constraints, which are part of $\mathcal{MMC}_{LAprop}$:

$$\forall M \forall n \forall k\ name(M, n) \wedge size(M, k, k) \rightarrow \exists Q \exists R$$
$$QR(M, Q, R) \wedge type(Q, \text{“}O\text{”}) \wedge type(R, \text{“}U\text{”})$$
$$\wedge\ multi_M(Q, R, M) \quad (6)$$
$$\forall Q_1\ type(Q_1, \text{“}O\text{”}) \rightarrow \exists I\ QR(Q_1, Q_1, I) \wedge identity(I)$$
$$\wedge\ multi_M(Q_1, I, Q_1) \quad (7)$$
$$\forall R_1\ type(R_1, \text{“}U\text{”}) \rightarrow \exists I\ QR(R_1, I, R_1) \wedge identity(I)$$
$$\wedge\ multi_M(I, R_1, R_1) \quad (8)$$
$$\forall I\ identity(I) \rightarrow QR(I, I, I) \quad (9)$$

Known LA properties of the other matrix decompositions (LU and PLU) are similarly encoded in Appendix A.

6.2.6 ***Encoding LA-Oriented System Rewrite Rules.*** Most LA-oriented systems [7, 8] execute an incoming expression (LA pipeline) *as-is*, that is: run operations in a sequence, whose order is dictated by the expression syntax. Such systems do not exploit basic LA properties, e.g., reordering a chain of multiplied matrices in order to reduce the intermediate size. SystemML [21] is the only system that models *some* LA properties as static rewrite rules. It also comprises a set of *rewrite rules* which modify the given expressions to avoid large intermediates for aggregation and statistical operations such as $\texttt{rowSums}(M)$, $\texttt{sum}(M)$, etc. For example, SystemML uses rule:

$$\texttt{sum}(MN) = \texttt{sum}(\texttt{colSums}(M)^T \odot \texttt{rowSums}(N)) \quad (i)$$

to rewrite $\texttt{sum}(MN)$ (summing all cells in the matrix product) where $\odot$ is a matrix element-wise multiplication, to avoid actually computing MN and materializing it; similarly, it rewrites $\texttt{sum}(M^T)$ into $\texttt{sum}(M)$, to avoid materializing M^T, etc. However, the performance benefits of rewriting depend on the rewriting power (or, in other words, on *how much the system understands the semantics of the incoming expression*), as the following example shows.

$$RW_1 : (M^{-1})^T + N^T \qquad RW_2 : (M^T)^{-1} + N^T$$
$$RW_3 : N^T + (M^{-1})^T \qquad RW_4 : (N^T)^{-1} + N^T$$
$$RW_5 : (N + M^{-1})^T$$

Figure 4: Equivalent rewritings of the pipeline Q_p.

Example 6.3. *Consider the LA expression E=$((M^T)^k(M + N)^T)$, where M and N are square matrixes, and expression E'=$\texttt{sum}(E)$, which computes the sum of all cells in E. The expression E' can be rewritten to $RW_1 : \texttt{sum}(E'')$, where E'' is:*

$$\texttt{sum}(\texttt{colSums}(M + N)^T \odot \texttt{rowSums}(M^k))$$

Failure to exploit the properties $LA_{prop1} : M^T N^T = (MN)^T$ and $LA_{prop2} : (M^n)^T = (M^T)^n$ prevents from finding rewriting RW_1. E' admits the alternative rewriting

RW_2: $\texttt{sum}((\texttt{colSums}((M^T)^k))^T \odot (\texttt{colSums}(M + N)^T)$

which can be obtained by directly applying the rewrite rule (i) above and $\texttt{rowSums}(M^T)$=$\texttt{colSums}(M)^T$, without exploiting the properties LA_{prop1} and LA_{prop2}. However, RW_2 introduces more intermediate results than RW_1.

To fully exploit the potential of rewrite rules (for statistical or aggregation operations), they should be accompanied by sufficient knowledge of, and reasoning on, known properties of LA operations.

To bring such fruitful optimization to other LA-oriented systems lacking support of such rewrite rules, we have incorporated SystemML's rewrite rules into our framework, encoding them as a set of integrity constraints over the virtual relations in the schema $\mathcal{VRELM}$, denoted $\mathcal{MMC}_{StatAgg}$ ($\mathcal{MMC}_{StatAgg} \subset \mathcal{MMC}$). Thus, these rewrite rules can be exploited together with other LA properties. For instance, the rewrite rule (i) is modeled by the following integrity constraint $\mathcal{I}_{sum} \in \mathcal{MMC}_{StatAgg}$:

$$\forall M \forall N \forall R\ \texttt{multi}_M(M, N, R) \wedge \texttt{sum}(R, s) \rightarrow$$
$$\exists R_1 \exists R_2 \exists R_3 \exists R_4 \texttt{colSums}(M, R_1) \wedge \texttt{tr}(R_1, R_2)$$
$$\wedge\ \texttt{rowSums}(N, R_3) \wedge \texttt{multi}_E(R_2, R_3, R_4) \wedge \texttt{sum}(R_4, s)$$

We refer the reader to Appendix B for a full list of SystemML's encoded rewrite rules.

6.3 Relational Rewriting Using Constraints

With the set of views constraints $\mathcal{C}_\mathcal{V}$ and $\mathcal{MMC} = \mathcal{MMC}_m \cup \mathcal{MMC}_{LA_{prop}} \cup \mathcal{MMC}_{StatAgg}$, we rely on $PACB^{++}$ to rewrite a given expression under integrity constraints. We exemplify this below, and detail $PACB^{++}$'s inner workings in Section 7.

The view V shown in Figure 3 can be used to *fully* rewrite (return the answer for) the pipeline $Q_p : (M^{-1} + N)^T$ by exploiting the TGDs (1), (2) and (3) listed in Figure 2, which describe the following three LA properties, denoted LA_{prop_1}: $M+N = M+N$; $((M+N))^T = (M)^T + (N)^T$ and $((M)^{-1})^T = ((M)^T)^{-1}$. The relational rewriting RW_0 of Q_p using the view V is $RW_0(R_4)$:- $name(R_4, \text{“}V.csv\text{”})$. In this example, RW_0 is the only *views-based* rewriting of Q_p. However, *five* other rewritings exist (shown in Figure 4), which reorder its operations just by exploiting the set LA_{prop_1} of LA properties.

Rewritings RW_0 to RW_5 have different evaluation costs. We discuss next how we estimate which among these alternatives (including evaluating Q_p directly) is likely the most efficient.

7 CHOICE OF AN EFFICIENT REWRITING

We introduce our cost model (§7.1), which can take two different sparsity estimators (§7.2). Then, we detail our extension to the PACB rewriting engine based on the $Prune_{prov}$ algorithm (§7.3) to prune out inefficient rewritings.

7.1 Cost Model

We estimate the cost of an expression E, denoted $\gamma(E)$, as the sum of the intermediate result sizes if one evaluates E "as stated", in the syntactic order dictated by the expression. Real-world matrices may be *dense* (most or all elements are non-zero) or *sparse* (a majority of zero elements). The latter admit more economical representations that do not store zero elements, which our intermediate result size measure excludes. To estimate the number of non-zeros (*nnz*, in short), we incorporated two different sparsity estimators from the literature (discussed in §7.2) into our framework.

Example 7.1. *Consider $E_1 = (MN)M$ and $E_2 = M(NM)$, where we assume the matrices $M_{50K\times100}$ and $N_{100\times50K}$ are dense. The total cost of E_1 is $\gamma(E_1) = 50K \times 50K$ and $\gamma(E_2) = 100 \times 100$.*

7.2 LA-based Sparsity Estimators

We outline below two existing *sparsity estimators* [21, 49] that we have incorporated into our framework to estimate *nnz*.[2].

7.2.1 ***Naïve Metadata Estimator.*** The naïve metadata estimator [22, 49] derives the sparsity of the output of LA expression solely from the base matrices' sparsity. This incurs no runtime overhead since metadata about the base matrices, including the *nnz*, columns and rows are available before runtime in a specific metadata file. The most common estimator is the *worst-case estimator* [22], which we use in our framework.

7.2.2 ***Matrix Non-zero Count (MNC) Estimator.*** The MNC estimator [46] exploits matrix structural properties such as single non-zero per row, or columns with varying sparsity, for efficient, accurate, and general sparsity estimation; it relies on count-based histograms that exploit these properties. We have also adopted this framework into our approach, and compute histograms about the *base* matrices offline. However, the MNC framework still needs to derive and construct histograms for *intermediate results* online (during rewriting cost estimation). We study this overhead in (§9).

7.3 Rewriting Pruning: $PACB^{++}$

We extended the PACB rewriting engine with the $Prune_{prov}$ algorithm sketched and discussed in [31, 32], to eliminate inefficient rewritings during the rewriting search phase. The naïve PACB algorithm generates all minimal (by join count) rewritings before choosing a *minimum-cost* one. While this sufficies on the scenarios considered in [15, 32], the settings we obtain from our LA encoding stress-test the naïve algorithm, as commutativity, associativity, etc. blow up the space of alternate rewritings exponentially. Scalability considerations forced us to further optimize naïve PACB to find only *minimum-cost rewritings*, aggressively pruning the others during the generation phase. We illustrate $Prune_{prov}$ and our improvements next.

$Prune_{prov}$ Minimum-Cost Rewriting. Recall from (§4) that the minimal rewritings of a query Q are obtained by first finding the set $\mathcal{H}$ of all matches (i.e., containment mappings) from Q to the result B of backchasing the universal plan U. Denoting with $\pi(S)$ the provenance formula of a set of atoms S, PACB computes the DNF form D of $\bigvee_{h\in\mathcal{H}} \pi(h(Q))$. Each conjunct c of D determines a subquery $sq(c)$ of U which is guaranteed to be a rewriting of Q.

The idea behind cost-based pruning is that, whenever the naive PACB backchase would add a provenance conjunct c to an existing atom a's provenance formula $\pi(a)$, $Prune_{prov}$ does so more conservatively: if the cost $\gamma(sq(c))$ is larger than the minimum cost threshold T found so far, then c will never participate in a minimum-cost rewriting and need not be added to $\pi(a)$. Moreover, atom a itself need not be chased into B in the first place if all its provenance conjuncts have above-threshold cost.

Example 7.2. *Let $E = M(NM)$, where we assume for simplicity that $M_{50K\times100}$ and $N_{100\times50K}$ are dense. Exploiting the associativity of matrix-multiplication $(MN)M = M(NM)$ during the chase leads to the following universal plan U annotated with provenance terms:*

$$U(R_2) : name(M, \text{"}M.csv\text{"})^{p_0} \wedge size(M, 50000, 100)^{p_1} \wedge name(N, \text{"}N.csv\text{"})^{p_2} \wedge size(N, 100, 50000)^{p_3} \wedge \texttt{multi}_M(M, N, R_1)^{p_4} \wedge \texttt{multi}_M(R_1, M, R_2)^{p_5} \wedge \texttt{multi}_M(N, M, R_3)^{p_6} \wedge \texttt{multi}_M M, R_3, R_2)^{p_7}$$

Now, consider in the back-chase the associativity constraint C:

$$\forall M \forall N \forall R_1 \forall R_2 \quad \texttt{multi}_M(M, N, R_1) \wedge \texttt{multi}_M(R_1, M, R_2) \rightarrow \exists R_4 \texttt{multi}_M(N, M, R_4) \wedge \texttt{multi}_M(M, R_4, R_2)$$

There exists a containment mapping h embedding the two atoms in the premise P of C into the U atoms whose provenance annotations are p_4 and p_5. The provenance conjunct collected from P's image is $\pi(h(P))$=$p_4 \wedge p_5$.

Without pruning, the backchase would chase U with the constraint C, yielding U' which features additional $\pi(h(P))$-annotated atoms $\texttt{multi}_M(N, M, R_4)^{p_4 \wedge p_5} \wedge \texttt{multi}_M(M, R_4, R_2)^{p_4 \wedge p_5}$

E has precisely two matches h_1, h_2 into U'. $h_1(E)$ involves the newly added atoms as well as those annotated with p_0, p_1, p_2, p_3. Collecting all their provenance annotations yields the conjunct $c_1 = p_0 \wedge p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge p_5$. c_1 determines the U-subquery $sq(c_1)$ corresponding to the rewriting $(MN)M$, of cost $(50K)^2$.

$h_2(E)$'s image yields the provenance conjunct $c_2 = p_0 \wedge p_1 \wedge p_2 \wedge p_3 \wedge p_6 \wedge p_7$, which determines the rewriting $M(NM)$ that happens to be the original expression E of cost 100^2.

The naive PACB would find both rewritings, cost them, and drop the former in favor of the latter.

With pruning, the threshold T is the cost of the original expression 100^2. The chase step with C is never applied, as it would introduce the provenance conjunct $\pi(h(P))$ which determines U-subquery $sq(\pi(h(P))) = \texttt{multi}_M(M, N, R_1)^{p_4} \wedge \texttt{multi}_M(R_1, M, R_2)^{p_5}$

of cost $(50K)^2$ exceeding T. The atoms needed as image of E under h_1 are thus never produced while backchasing U, so the expensive rewriting is never discovered. This leaves only the match image $h_2(E)$, which corresponds to the efficient rewriting $M(NM)$.

Our improvements on $Prune_{prov}$. Whenever the pruned chase step is applicable and applied for each TGD constraint, the original algorithm searches for all *minimal-rewritings* $\mathcal{RW}$ that can be found "so far", then it costs each $rw \in \mathcal{RW}$ to find the "so far" *minimum-cost one* rw_e and adjusts the threshold T to the cost of rw_e.

[2] Solving the problem of sparsity estimation is beyond the scope of this paper

However, this strategy can cause *redundant* costing of $rw \in \mathcal{RW}$ whenever the pruned chase step is applied again for another constraint. Therefore, in our modified version of $Prune_{prov}$, we keep track of the rewriting costs already estimated, to prevent such redundant work. Additionally, the search for *minimal-rewritings* "so far" (matches of the query Q into the evolving universal plan instance U', see §4) whenever the pruned chase step is applied is modeled as a query evaluation of Q against U' (viewed as a symbolic/canonical database [14]). This involves repeatedly evaluating the same query plan. However, the query is evaluated over evolutions of the same instance. Each pruned chase step *adds a few new tuples* to the evolving instance, corresponding to atoms introduced by the step, while most of the instance is unchanged. Therefore, instead of evaluating the query plan from scratch, we employ incremental evaluation as in [32]. The plan is kept in memory along with the populated hash tables, and whenever new tuples are added to the evolving instance, we push them to the plan.

8 GUARANTEES ON THE REDUCTION

We detail the conditions under which we guarantee that our approach is *sound* (i.e., generates only equivalent, cost-optimal rewritings), and *complete* (i.e., finds all equivalent cost-optimal rewritings).

Let $\mathcal{V} \subseteq \mathcal{L}$ be a set of materialized view definitions, where $\mathcal{L}$ is the language of hybrid expressions described in §3.

Let LA_{prop} be a set of properties of the LA operations in L_{ops} that admits relational encoding over $\mathcal{VRELM}$. We say that LA_{prop} is *terminating* if it corresponds to a set of TGDs and EGDs with terminating chase (this holds for our choice of LA_{prop}).

Denote with γ a cost model for expressions from $\mathcal{L}$. We say that γ is *monotonic* if expressions are never assigned a lower cost than their subexpressions (this is true for both models we used).

We call $E \in \mathcal{L}$ $(\gamma, LA_{prop}, \mathcal{V})$-*optimal* if for every $E' \in \mathcal{L}$ that is $(LA_{prop}, \mathcal{V})$-equivalent to E we have $\gamma(E') \geq \gamma(E)$.

Let $\text{Eq}^{\gamma}\langle LA_{prop}, \mathcal{V}\rangle(E)$ denote the set of all $(\gamma, LA_{prop}, \mathcal{V})$-optimal expressions that are $(LA_{prop}, \mathcal{V})$-equivalent to E.

We denote with $HADAD\langle LA_{prop}, \mathcal{V}, \gamma\rangle$ our parameterized solution based on relational encoding followed by PACB++ rewriting and next by decoding all the relational rewritings generated by the cost-based pruning PACB++ (recall Figure 1). Given $E \in \mathcal{L}$, $HADAD\langle LA_{prop}, \mathcal{V}, \gamma\rangle(E)$ denotes all expressions returned by $HADAD\langle LA_{prop}, \mathcal{V}, \gamma\rangle$ on input E.

Theorem 8.1 (Soundness). *If the cost model γ is monotonic, then for every $E \in \mathcal{L}$ and every $rw \in HADAD\langle LA_{prop}, \mathcal{V}, \gamma\rangle(E)$, we have $rw \in Eq^{\gamma}\langle LA_{prop}, \mathcal{V}\rangle(E)$.*

Theorem 8.2 (Completeness). *If γ is monotonic and LA_{prop} is terminating, then for every $E \in \mathcal{L}$ and every $rw \in Eq^{\gamma}\langle LA_{prop}, \mathcal{V}\rangle(E)$, we have $rw \in HADAD\langle LA_{prop}, \mathcal{V}, \gamma\rangle(E)$.*

9 EXPERIMENTAL EVALUATION

We evaluate HADAD to answer research questions below about our approach:

- **§9.1.1, §9.1.2, §9.2.1** and **§9.2.2**: Can HADAD find rewrites with/without views that lead to a greater performance improvement than original pipelines without modifying the internals of the existing systems?. Are the identified rewrites found by state-of-the-art platforms?.
- **§9.1.3** and **§9.2.3**: Is HADAD's optimization overhead compensated by the performance gains in execution?

We evaluate our approach, first on **LA-based pipelines** (§9.1), then on **hybrid expressions** (§9.2). Due to space constraints, we only discuss a subset of our results here and delegate other results to Appendix C, D, E, and F.

Experimental Environment. We used a single node with an Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz, 40 Cores (hyperthreading), 123GB RAM, disk read speed 616 MB/s, and disk write speed 455 MB/s. We run on OpenJDK Java 8 VM . As for **LA systems/libraries**, we used **R 3.6.0**, **Numpy 1.16.6 (python 2.7)**, **TensorFlow 1.4.1**, **Spark 2.4.5 (MLlib)**, **SystemML 1.2.0** For hybrid experiments, we use **Morpheus** [4] and **SparkSQL** [17].

Systems Configuration Tuning. We discuss here the most important installation and configuration details. We use a JVM-based linear algebra library for SystemML as recommended in [48], at the optimization level **4**. Additionally, we enable multi-threaded matrix operations in a single node. We run Spark in single node setting and OpenBLAS (built from the sources as detailed in [5]) to take advantage of its accelerations [48]. SparkMLlib's datatypes do not support many basic LA operations, such as scalar-matrix multiplication, matrix element-wise multiplication, etc. To support them, we use the `Breeze` Scala library [2], convert MLlib's datatypes to Breeze types and express the basic LA operations in Spark. The driver memory allocated for Spark and SystemML is 115GB. To maximize TensorFlow performance, we compile it from sources. For all systems/libraries, we set the number of **cores** to **24**; all systems use **double precision** numbers.

9.1 LA-based Experiments

In this experiments, we study the performance benefits of our approach on LA-based pipelines as well as our optimization overhead.

Datasets. We used several **real-world, sparse matrices**, for which Table 4 lists the dimensions and the sparsity ($\mathbf{S}_X$) (*i*) dielFilterV3real (**DFV** in short) is an analysis of a microwave filter with different mesh qualities [28]; (*ii*) 2D_54019_highK (**2D_54019** in short) is a 2D semiconductor device simulation [28]; (*iii*) we used several subsets of an Amazon books review dataset [1] (in JSON), and similarly (*iv*) subsets of a Netflix movie rating dataset [6]. The latter two were easily converted into matrices where columns are items and rows are customers [49]; we extracted smaller subsets of all real datasets to ensure the various computations applied on them fit in memory (e.g., Amazon/(AS) denotes the small version of the Amazon dataset). We also used a set **synthetic, dense matrices**, described in Table 5.

LA benchmark. We select a set $\mathcal{P}$ of **57 LA pipelines** used in prior studies and/or frequently occurring in real-world LA computations, as follows:

- **Real-world pipelines** include: a chain of matrix self products used for reachability queries and other graph analytics [46] (**P1.29** in Table 2); expressions used in Alternating Least Square Factorization (ALS) [49] (**P2.25** in Table 14); Poisson Nonnegative Matrix Factorization (PNMF)(**P1.13** in Table 15) [49]; Nonnegative Matrix Factorization (NMF)(**P1.25** and **P1.26** in Table 15) [48]; complex predicate for image

No.	Expression	No.	Expression	No.	Expression
P1.1	$(MN)^T$	P1.2	$A^T + B^T$	P1.3	$C^{-1}D^{-1}$
P1.4	$(A+B)v_1$	P1.5	$((D)^{-1})^{-1}$	P1.6	$\mathtt{trace}(s_1 D)$
P1.7	$((A)^T)^T$	P1.8	$s_1A + s_2A$	P1.9	$\mathtt{det}(D^T)$
P1.10	$\mathtt{rowSums}(A^T)$	P1.11	$\mathtt{rowSums}(A^T + B^T)$	P1.12	$\mathtt{colSums}((MN)$
P1.13	$\mathtt{sum}(MN)$	P1.14	$\mathtt{sum}(\mathtt{colSums}((N^T M^T))$	P1.15	$(MN)M$
P1.16	$\mathtt{sum}(A^T)$	P1.17	$\mathtt{det}(CDC)$	P1.18	$\mathtt{sum}(\mathtt{colSums}((A))$
P1.19	$(C^T)^{-1}$	P1.20	$\mathtt{trace}(C^{-1})$	P1.21	$(C + D^{-1})^T$
P1.22	$\mathtt{trace}((C+D)^{-1})$	P1.23	$\mathtt{det}((CD)^{-1}) + D)$	P1.24	$\mathtt{trace}((CD)^{-1})) + \mathtt{trace}(D)$
P1.25	$M \odot (N^T/(MNN^T))$	P1.26	$N \odot (M^T/(M^T MN))$	P1.27	$\mathtt{trace}(D(CD)^T)$
P1.28	$A \odot (A \odot B + A)$	P1.29	$DCCC$	P1.30	$NM \odot NMR^T$

Table 2: LA Benchmark Pipelines (Part 1)

No.	Expression	No.	Expression	No.	Expression
P2.1	$\mathtt{trace}(C + D)$	P2.2	$\mathtt{det}(D^{-1})$	P2.3	$\mathtt{trace}(D^T)$
P2.4	$s_1A + s_1B$	P2.5	$\mathtt{det}((C+D)^{-1})$	P2.6	$C^T(D^T)^{-1}$
P2.7	$DD^{-1}C$	P2.8	$\mathtt{det}(C^T D)$	P2.9	$\mathtt{trace}(C^T D^T + D)$
P2.10	$\mathtt{rowSums}(MN)$	P2.11	$\mathtt{sum}(A + B)$	P2.12	$\mathtt{sum}(\mathtt{rowSums}(N^T M^T))$
P2.13	$((MN)M)^T$	P2.14	$((MN)M)N$	P2.15	$\mathtt{sum}(\mathtt{rowSums}(A))$
P2.16	$\mathtt{trace}(C^{-1}D^{-1}) + \mathtt{trace}(D)$	P2.17	$((((C+D)^{-1})^T)((D^{-1})^{-1})C^{-1}C$	P2.18	$\mathtt{colSums}((A^T + B^T)$
P2.19	$(C^T D)^{-1}$	P2.20	$(M(NM))^T$	P2.21	$(D^T D)^{-1}(D^T v_1)$
P2.22	$\mathtt{exp}((C+D)^T)$	P2.23	$\mathtt{det}(C) * \mathtt{det}(D) * \mathtt{det}(C)$	P2.24	$(D^{-1}C)^T$
P2.25	$(u_1 v_2^T - X)v_2$	P2.26	$\mathtt{exp}((C+D)^{-1})$	P2.27	$((((C+D)^T)^{-1})D)C$

Table 3: LA Benchmark Pipelines (Part 2)

Name	Rows n	Cols m	Nnz $\|X\|_0$	S_X
DFV	1M	100	8050	0.0080%
2D_54019	50K	100	3700	0.0740%
Amazon/(AS)	50K	100	378	0.0075%
Amazon/(AM)	100K	100	673	0.0067%
Amazon/(AL_1)	1M	100	6539	0.0065%
Amazon/(AL_2)	10M	100	11897	0.0011%
Amazon/(AL_3)	100K	50K	103557	0.0020%
Netflix/(NS)	50K	100	69559	1.3911%
Netflix/(NM)	100K	100	139344	1.3934%
Netflix/(NL_1)	1M	100	665445	0.6654%
Netflix/(NL_2)	10M	100	665445	0.0665%
Netflix/(NL_3)	100K	50K	15357418	0.307%

Table 4: Overview of Used Real Datasets.

Name	Rows n	Cols m	Name	Rows n	Cols m
Syn_1	50K	100	Syn_6	20K	20K
Syn_2	100	50K	Syn_7	100	1
Syn_3	1M	100	Syn_8	50K	1
Syn_4	5M	100	Syn_9	100K	1
Syn_5	10K	10K	Syn_{10}	100	100

Table 5: Syntactically Generated Dense Datasets

Matrix Name	Used Data
A and B	AM, AL_1, AL_2, NM, NL_1, NL_2, dielFilter, Syn_3 or Syn_4
C and D	Syn_5 or Syn_6
M	AS, NS, Syn_1, or 2D_54019
N	Syn_2
R	Syn_{10}
X	AL_3 or NL_3
v_1,v_2 and u_1	Syn_7, Syn_8 and Syn_9, respectively.

Table 6: Matrices used for each matrix name in a pipeline

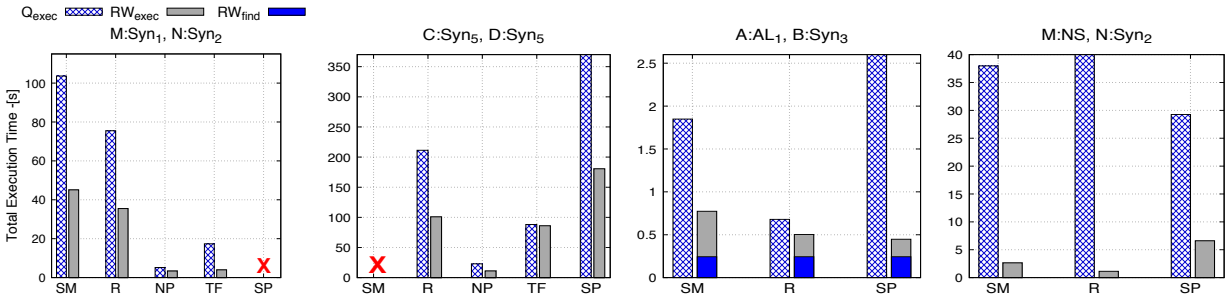
masking [46] (**P1.30** in Table 15); recommendation computation [46] (**P1.30** in Table 15); finally, Ordinary Least Squares Regression (OLS) [48] (**P2.21** in Table 14).

- **Synthetic pipelines** were also generated, based on a set of basic matrix operations (inverse, multiplication, addition, etc.), and a set of combination templates, written as a Rule-Iterated Context-Free Grammar (RI-CFG) [43]. Expressions thus generated include **P2.16**, **P2.16**, **P2.23**, **P2.24** in Table 15.

Methodology. In (§9.1.1), we show the performance benefits of our approach to LA-oriented systems/tools mentioned above using a set $\mathcal{P}^{\neg Opt} \subset \mathcal{P}$ of **38** pipelines in Table 15 and Table 14, and the matrices in Table 6. The performance of these pipelines can be improved *just by exploiting LA properties* (in the absence of views). For TensorFlow and NumPy, we present the results only for dense matrices, due to limited support for sparse matrices. In (§9.1.2), we show how our approach improves the performance of **30** pipelines from $\mathcal{P}$, denoted $\mathcal{P}^{Views}$, using pre-materialized views. Finally, in (§9.1.3), we study our rewriting performance and *optimization overhead* for the set $\mathcal{P}^{Opt} = \mathcal{P} \setminus \mathcal{P}^{\neg Opt}$ of **19** pipelines that are already optimized.

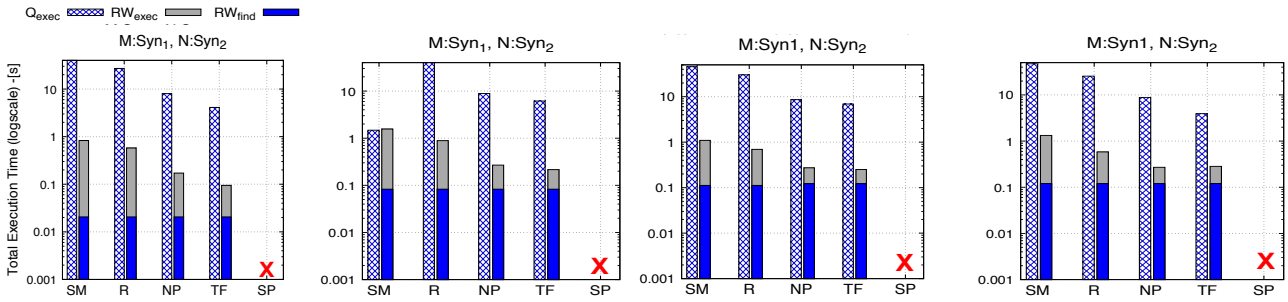
9.1.1 ***Effectiveness of LA Rewriting (No Views).*** For each system, we run the original pipeline and our rewriting 5 times; we report the average of the last 4 running times. We exclude the data loading time. For fairness, we ensured SparkMLib and SystemML compute the entire pipeline (despite their lazy evaluation mode).

Figure 5 illustrates the original pipeline execution time Q_{exec} and the selected rewriting execution time RW_{exec} for **P1.1**, **P1.3**, **P1.4**, and **P1.15**, including the rewriting time RW_{find}, using the MNC cost model. For each pipeline, the used datasets are on top of the figure. For brevity in the figures, we use **SM** for SystemML, **NP** for NumPy, **TF** for Tensorflow, and **SP** for MLlib. For **P1.1** (see Figure 5(a)), both matrices are dense. The speed-up (1.5× to 4×) comes from rewriting $(MN)^T$ (intermediate result size to $(50K)^2$) into $N^T M^T$, much cheaper since both N^T and M^T are of size $50K \times 100$.



(a) **P1.1** (b) **P1.3** (c) **P1.4** (d) **P1.15**

Figure 5: P1.1, P1.3, P1.4, and P1.15 evaluation before and after rewrite



(a) **P1.13** (b) **P1.25** (c) **P1.14** (d) **P2.12**

Figure 6: P1.13, P1.25, P1.14 and P2.12 evaluation before and after rewrite

We exclude MLlib from this experiment since it failed to allocate memory for the intermediate matrix (Spark/MLLib limits the maximum size of a dense matrix). As a variation (not plotted in the Figure), we ran the same pipeline with the ultra-sparse AS matrix (0.0075% non-zeros) used as M. The Q_{exec} and RW_{exec} time are very comparable using SystemML, because we avoid large dense intermediates. In R, this scenario lead to a c since the multiplication operator tries to densify the matrix M. To avoid it, we cast M during load time to a dense matrix type. Thus, the speed-up achieved is the same as if M and N were both dense. If, instead, NS (1.3860% non-zeros) plays the role of M, our rewrite achieves $\approx 1.8\times$ speed-up for SystemML.

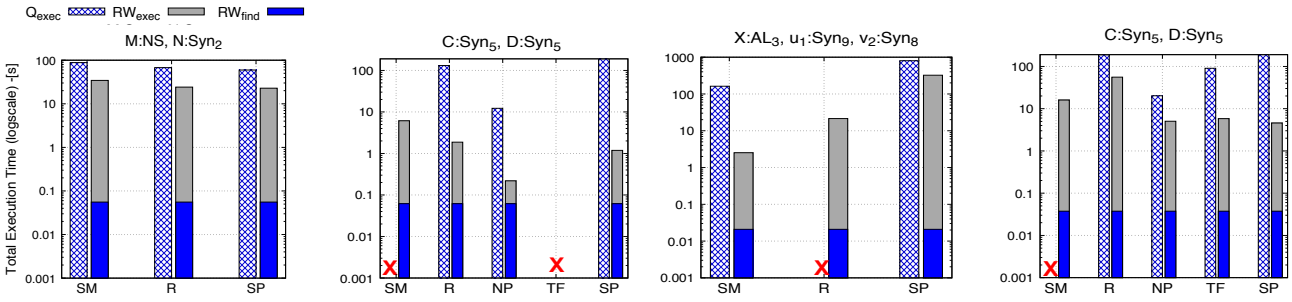
For **P1.3** (Figure 5(b)), the speed-up comes from rewriting $C^{-1}D^{-1}$ to $(DC)^{-1}$. Interestingly, TensorFlow is the only system that applies this optimization by itself. SystemML timed-out (>1000 secs) for both original pipeline and its rewriting.

For pipeline **P1.4** (Figure 5(c)), we rewrite $(A + B)v_1$ to $Av_1 + Bv_1$. Adding a sparse matrix A to a dense matrix B results into materializing a dense intermediate of size $1M \times 100$. Instead, $Av_1 + Bv_1$ has fewer non-zeros in the intermediate results, and Av_1 can be computed efficiently since A is sparse. The MNC sparsity estimator has a noticeable overhead here. We run the same pipeline, where the dense Syn_4 matrix plays both A and B (not shown in the Figure). This leads to speed-up of up to 9× for MLlib, which does not natively support matrix addition, thus we convert its matrices to Breeze types in order to perform it (as in [48]).

P1.15 (Figure 5(d)) is a matrix chain multiplication. The naïve left-to-right evaluation plan $(MN)M$ computes an intermediate matrix of size $O(n^2)$, where n is $50K$. Instead, the rewriting $M(NM)$ only needs an $O(m^2)$ intermediate matrix, where m is 100, and is much faster. To avoid MLLib memory failure on **P1.15**, we use the distributed matrix of type **BlockMatrix** for both matrices. While M thus converted has the same sparsity, Spark views it as being of a dense type (multiplication on BlockMatrix is considered to produce dense matrices) [10]. SystemML does optimize the multiplication order if the user does not enforce it. Further (not shown in the Figure), we ran **P.15** with AS in the role of M. This is 4× faster in SystemML since with an ultra sparse M, multiplication is more efficient. This is not the case for MLlib which views it as dense. For R, we again had to densify M during loading to prevent crashes.

Figures 6(a) and 6(b) study **P1.13** and **P1.25**, two real-world pipelines involved in ML algorithms, using the MNC cost model; note the log-scale y axis. Rewriting **P1.13** to $\texttt{sum}(t(\texttt{colSums}((M)) * \texttt{rowSums}(N))$ yields a speed-up of 50×; while SystemML *has* this rewrite as a static rule, it did not apply it. Our rewrite allowed SystemML and the others to benefit from it. Not shown in the Figure, we re-ran this with M ultra sparse (using AS) and SystemML: the rewrite did not bring benefits, since MN is already efficient. *In this experiment and subsequently, whenever MLlib is absent, this is due to its lack of support for LA operations* (here, sum of all cells in a matrix) on BlockMatrix. For **P1.25**, the important optimization is selecting the multiplication order in MNN^T (Figure 6(b)). SystemML is efficient here, due to its dedicated operator `tsmm` for *t*ranspose-*s*elf *m*atrix *m*ultiplication and `mmchain` for matrix multiply chains.

Figures 6(c) and 6(d) shows up to 42× rewriting speed-up achieved by turning **P1.14** and **P2.12** into $\texttt{sum}(t(\texttt{colSums}((M)) * \texttt{rowSums}(N))$. This exploits several properties: $(i)\ (MN)^T = N^T M^T$, $(ii)\ \texttt{sum}(M^T) =$



(a) **P2.14** (b) **P2.21** (c) **P2.25** (d) **P2.27**

Figure 7: P2.14, P2.21, P2.25 and P2.27 evaluation before and after rewriting using the views V_{exp}

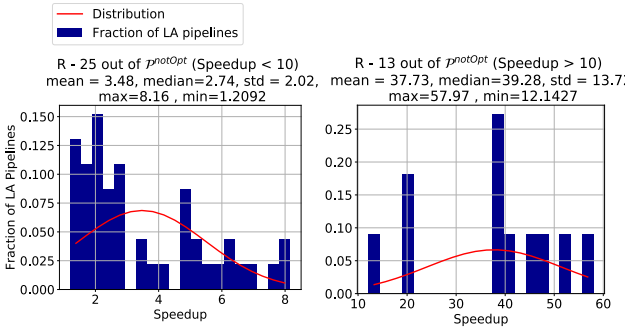


Figure 8: R speed-up on $\mathcal{P}^{\neg Opt}$

$\texttt{sum}(M)$, (iii) $\texttt{sum}(row/\texttt{colSums}((M)) = \texttt{sum}(M)$, and (iv) $\texttt{sum}(MN) = \texttt{sum}(t(\texttt{colSums}((M)) * \texttt{rowSums}(N))$. SystemML captures (ii), (iii), and (iv) as static rewrite rules, however, it is unable to exploit these performance-saving opportunities since it is unaware of (i). Other systems lack support for more or all of these properties.

Figure 8 shows the distribution of the significant rewriting speed-up on $\mathcal{P}^{\neg Opt}$ running on R, and using the MNC-based cost model. For clarity, we split the distribution into two figures: on the left, 25 $\mathcal{P}^{\neg Opt}$ pipelines with speed-up lower than 10×; on the right, the remaining 13 with greater speed-up. Among the former, 87% achieved at least 1.5× speed-up. The latter are sped up by 10× to 60×. **P1.5** is an extreme case here (not plotted): it is sped up by about 1000×, simply by rewriting $((D)^{-1})^{-1}$ into D.

9.1.2 ***Effectiveness of view-based LA rewriting.*** We have defined a set V_{exp} of 12 views that pre-compute the result of some expensive operations (multiplication, inverse, determinant, etc.) which can be used to answer our $\mathcal{P}^{Views}$ pipelines, and materialized them on disk as CSV files. The experiments outlined below used the naïve cost model; all graphs have a log-scale y axis.

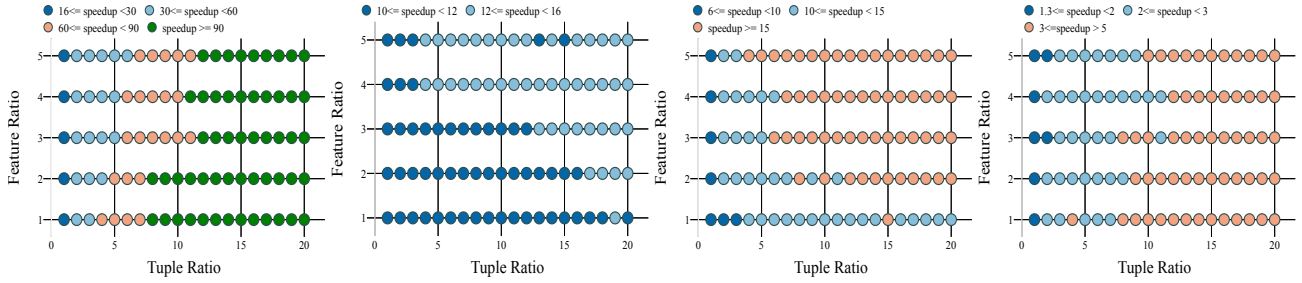
Discussion. For **P2.14** (Figure 7(a)), using the view $V_4 = NM$ by and the multiplication associativity leads to up to 2.8× speed-up.

Figure 7(b) shows the gain due to the view $V_1 = D^{-1}$, for the ordinary-least regression (OLS) pipeline **P2.21**. It has 8 rewritings, 4 of which use V_1; they are found thanks to the properties $(CD)^{-1} = D^{-1}C^{-1}$, $(CD)E = C(DE)$ and $(D^T)^{-1} = (D^{-1})^T$ among others. The cheapest rewriting is $V(V^T(D^T v_1))$, since it introduces small intermediates due to the optimal matrix chain multiplication order. This rewrite leads to 70×, 55× and 150× speed-ups on R, NumPy and MLlib, respectively; TensorFlow is omitted as `matmul` operator does not support matrix-vector multiplication. It could run this pipeline by converting the matrices to NumPy, whose performance we already report separately. On SystemML, the original pipeline timed out (> 1000 seconds).

Pipeline **P2.25** (Figure 7(c)) benefits from a view V_5, which pre-computes a dense intermediate vector multiplication result; then, rewriting based on the property $(A + B)v = Av + Bv$ leads to a 65× speed-up in SystemML. For MLlib, as discussed before, to avoid memory failure, we used BlockMatrix types. for all matrices and vectors, thus they were treated as dense. In R, the original pipeline triggers a memory allocation failure, which the rewriting avoids. Figure 7(d) shows that for **P2.27** exploiting the views $V_2 = (D+C)^{-1}$ and $V_3 = DC$ leads to speed-ups of 4× to 41× on different systems. Properties enabling rewriting here are $C + D = D + C$, $(D^T)^{-1} = (D^{-1})^T$ and $(CD)E = C(DE)$.

9.1.3 ***Rewriting Performance and Overhead.*** We now study the running time RW_{find} of our rewriting algorithm, and the *rewrite overhead* defined as $RW_{find}/(Q_{exec} + RW_{find})$, where Q_{exec} is the time to run the pipeline "as stated". We ran each experiment 100 times and report the average of the last 99 times. The global trends are as follows. (i) For a fixed pipeline and set of data matrices, *the overhead is slightly higher using the MNC cost model*, since histograms are built during optimization. (ii) For a fixed pipeline and cost model, *sparse matrices lead to a higher overhead* simply because Q_{exec} tends to be smaller. (iii) *Some* (system, pipeline) pairs lead to a low Q_{exec} when *the system applies internally the same optimization* that HADAD finds "outside" of the system.

Concretely, for the $\mathcal{P}^{\neg Opt}$ pipelines, on the dense and sparse matrices listed in Table 6, using the naïve cost model, 64% of the RW_{find} times are under 25ms (50% are under 20ms), and the longest is about 200m. Using the MNC estimator, 55% took less than 20ms, and the longest (outlier) took about 300ms. Among the 39 $\mathcal{P}^{\neg Opt}$ pipelines, SystemML finds efficient rewritings for a set of 6, denoted $\mathcal{P}^{\neg Opt}_{SM}$, while TensorFlow optimizes a different set of 11, denoted $\mathcal{P}^{\neg Opt}_{TF}$. On these subsets, where HADADś optimization is redundant, using *dense* matrices, the overhead is very low: with the *MNC* model, 0.48% to 1.12% on $\mathcal{P}^{\neg Opt}_{SM}$ (0.64% on average), and 0.0051% to 3.51% on $\mathcal{P}^{\neg Opt}_{TF}$ (1.38% on average). Using the *naïve* estimator slightly reduces this overhead, but across $\mathcal{P}^{\neg Opt}$, this model misses 4 efficient rewritings. On *sparse* matrices, the overhead is at most 4.86% with the *naïve* estimator and up to 5.11% with the *MNC* one.



(a) **P1.12** (b) **P2.10** (c) **P2.11** (d) **P2.15**

Figure 9: Speed-ups of Morpheus (with HADAD rewrites) over Morpheus (without HADAD rewrites) for pipelines P1.12, P2.10, P2.11 and P2.15 on synthetic data for a PK-FK join.

Among the already-optimal pipelines $\mathcal{P}^{Opt}$, 70% involve expensive operations such as inverse, determinant, matrix exponential, leading to rather long Q_{exec} times. Thus, the rewriting overhead is less than 1% of the total time, on all systems, using sparse or dense matrices, and the naïve or the MNC-based cost models. For the other $\mathcal{P}^{Opt}$ pipelines with short Q_{exec}, mostly matrix multiplications chains already in the optimal order, on *dense* matrices, the overhead reaches 0.143% (SparkMlLib) to 9.8% (TensorFlow) using the naïve cost model, while the MNC cost model leads to an overhead of 0.45% (SparkMlib) up to 10.26% (TensorFlow). On *sparse* matrices, using the naïve and MNC cost models, the overhead reaches up to 0.18% (SparkMLlib) to 1.94% (SystemML), and 0.5% (SparkMLlib) to 2.61% (SystemML), respectively.

9.2 Hybrid (LA and RA) Experiments

We now study the benefits of rewriting on hybrid expressions combining relational and linear algebra. In (§9.2.1), we show the performance benefits of our approach to Morpheus [4]. In (§9.2.2), we evaluate our micro-hybrid benchmark on SparkSQL with SystemML. (§9.2.3) discusses our optimization overhead.

9.2.1 ***Morpheus Experiments.*** In this experiment, we study the performance-saving opportunity that HADAD adds to Morpheus. Morpheus is a cross RA-LA platform for pushing LA computations through joins, where LA pipelines can be automatically "factorized" using its framework of algebraic rewrite rules. The rules rewrite LA operations over the join output's matrix (table) into LA operations over the base tables' matrices, thereby avoiding data redundancy introduced by the join, which leads to runtime inefficiencies due to redundancy in the computations. For example, (based on [4]), consider a data scientist using ML to predict if a customer will move to a competitor. She joins a table *Customers* with columns (features) such as income, employerID, etc. with another table *Employers* such as revenue, city, etc. The column vectors in the base tables are viewed as matrices, **C** and **E**, respectively. The materialized join output is viewed as a matrix **M**. Factorized LA rewrites LA operation over **M**, e.g., rowSums(**M**) into LA operations over **C** and **E**. Thus, the user writes LA pipeline on one table **M** but under the cover Morpheus rewrites the pipeline to operate on **C** and **E**. However, based on equivalences that hold due to LA properties, LA pipelines' rewrites are not exploited by Morpheus, leading to missed optimization opportunities, as we outlined below.

Methodology. We used the same experiments setup introduced in [4] for generating synthetic datasets for PK-FK join of **R** and **S** tables. The quantities varied are the *tuple ratio* (n_S/n_R) and feature ratio (d_R/d_S), where n_S and n_R are the number of rows and d_R and d_S are the number of columns (features) in **R** and **S**, respectively. We fixed $n_R = 1M$ and $d_S = 20$, this helps quantify the amount of redundancy introduced by a PK-FK join on **R** and **S**. The join's output matrix $\mathbf{M}(n_s \times (d_R + d_S))$ is always dense. We evaluate a set $\mathcal{P}_M^{\neg opt} \subset \mathcal{P}^{\neg Opt}$ of 8 pipelines and their selected rewritings found by HADAD (using naïve cost-model) on Morpheus.[3]

Discussion. **P1.9**: colSums(**M***N*) is the example from §2, with **M** the output (viewed as matrix) of joining tables **R** and **S** generated as described above. N is a $ncol(\mathbf{M}) \times 100$ dense matrix. HADAD's rewriting yields up to 125× speed-up (see Figure 9(a)).

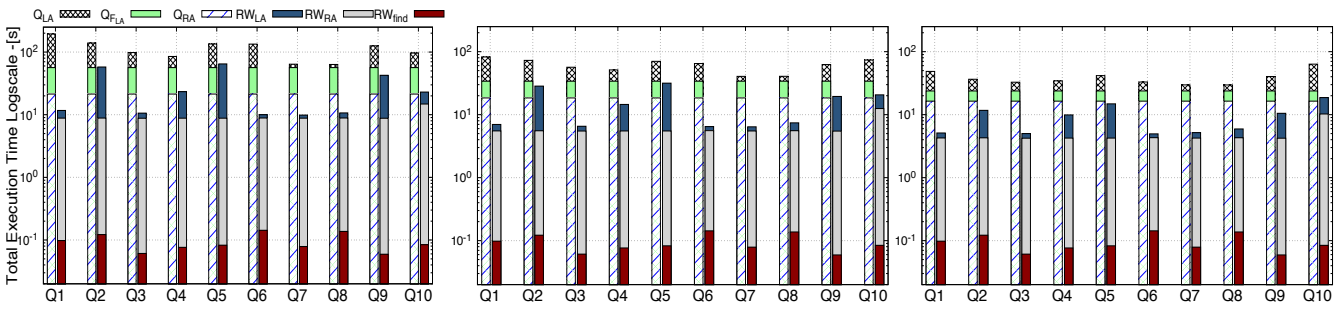
Figure 9(b) shows up to 15× speed-up for **P1.10**: rowSums(*N***M**), where the size of N is $100 \times nrow(\mathbf{M})$, This is due to HADAD's rewriting: NrowSums(**M**), which enables Morpheus to push the rowSums operator to **R** and **S** instead of computing the large intermediate matrix multiplication.

P1.11: sum(N+**M**) is run *as-is* by Morpheus since it does not factorize element-wise operations, e.g., addition. However, HADAD rewrites **P1.11** into sum(N)+sum(**M**), which avoids the (large and dense) intermediate result of the element-wise matrix addition. The HADAD rewriting enables Morpheus to execute sum(**M**) by pushing sum to **R** and **S**, for up to 20× speed-up (see Figure 9(c)).

Morpheus evaluates **P1.12**: sum(rowSums(**M**)) by pushing the rowSums operator to **R** and **S**. HADAD finds the rewriting sum(**M**), which enables Morpheus to push the sum operation instead, achieving up to 4.5× speed-up (see Figure 9(d)).

Since Morpheus does not exploit the associative property of matrix multiplication, it cannot reorder multiplication chains to avoid large intermediate results, which lead to runtime exception in R (Morpheus' backend). For example, for the chain $N^T N\mathbf{M}$, when the size of **M** is 1M×20 and the size of N is 20×1M, the size of the $N^T N$ intermediate result is 1M×1M, which R cannot handle. HADAD exploits associativity and selects the rewriting $N^T(N\mathbf{M})$ of intermediate result size (100×20).

[3] We select a set of pipelines where Morpheus can rewrite (factorize) them. This includes pipelines that contain aggregate operations, transpose and matrix multiplication. For the inverse, we only evaluated P2.21 (OLS), which is the realistic one.



(a) **N matrix: 2M rows** (b) **N matrix: 1M rows** (c) **N matrix : 0.5 rows**

Figure 10: Micro-Hybrid Benchmark Twitter Dataset

Interestingly, for pipelines that involve transpose operator, Morpheus applies its special rewrite rules that replace an operation on $\mathbf{M}^T$ with an operation on $\mathbf{M}$ before pushing the operation to the base tables. For example, for **P1.14**: $\texttt{sum}(\texttt{colSums}(\mathbf{M}^T N^T))$ and **P2.12**: $\texttt{sum}(\texttt{rowSums}(\mathbf{M}^T N^T))$, Morpheus rewrites the pipelines to $\texttt{sum}((\texttt{rowSums}(N\mathbf{M})))$ and $\texttt{sum}((\texttt{colSums}(N\mathbf{M})))$, respectively, and then applies its factorized matrix-multiplication rewrite rule on $N\mathbf{M}$. HADAD rewrites both pipelines to $\texttt{sum}(N\mathbf{M})$, enabling again Morpheus to apply its factorized rewrite rule on $N\mathbf{M}$ and achieving speed-up ranging from 1.3 up to 1.5×. The reason is that for both rewritings (found by Morpheus and HADAD), Morpheus's factorized matrix multiplication rewrite rule dominates the execution time.

9.2.2 ***Micro-hybrid Benchmark Experiments.*** In this experiment, we create a *micro-hybrid benchmark* on Twitter[12] and MIMIC [33] datasets to empirically study HADAD's rewriting benefits in a hybrid setting. The benchmark comprises ten different queries combining relational and linear algebra expressions.

Twitter Dataset Preparation. We obtain from Twitter API [12] 16GB of tweets (in JSON). We extract the structural parts of the dataset, which include user and tweet information, and store them in tables **User (U)** and **Tweet (T)**, linked via PK-FK relationships. The dataset is detailed in §2. The tables **User** and **Tweet** as well as **TweetJSON (TJ)** are stored in Parquet format.

Twitter Queries and Views. Queries consist of two parts: (*i*) RA preprocessing (Q_{RA}) and (*ii*) LA analysis (Q_{LA}). In the Q_{RA} part, queries construct two matrices: **M** and **N**. The matrix **M** (2M×12;*dense*) is the output of joining **T** and **U**. The construction of matrix **N** is described in §2[4]. We fix the Q_{RA} part across all queries and vary the Q_{LA} part using a set of LA pipelines detailed below.

In addition to the views defined in §2, we define three hybrid RA-LA materialized views: V_3, V_4 and V_5, which store the result of applying rowSums, colSums and matrix multiplication operations over base tables **T** and **U** (viewed as matrices).

Importantly, rewritings based on these views can only be found by *exploiting together LA properties* and *Morpheus' rewrites rules* (we incorporated them in our framework as a set of integrity constraints). The full list of queries and views is in Appendix H.

Discussion. For the first run of all queries (Figure 10(a)), matrix **N** (2M×1000) is constructed from **TJ** for all tweets that mention "covid" and are posted from "USA", and **M** (2M× 12) is constructed as described above. Both matrices are then loaded in SystemML to be used for the analysis part (varied using the set of pipelines in Table 7). Before evaluating an LA pipeline, the queries filter **N** 's rows with *filter-level* less than 4 (medium). Now, we analyze HADAD's rewrites and optimizations for each query in detail in the following paragraphs:

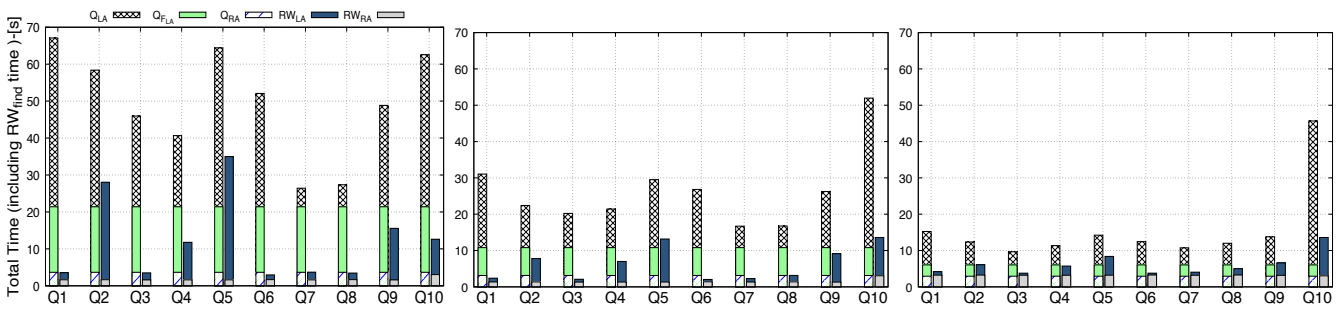
No.	Expression
P3.1	$\texttt{rowSums}(X\mathbf{M}) + (uv^T + \mathbf{N}^T)v$
P3.2	$\texttt{ucolSums}((X\mathbf{M})^T)+\mathbf{N}$
P3.3	$((\mathbf{N}+X)v)\texttt{colSums}(\mathbf{M})$
P3.4	$\texttt{sum}(C+\mathbf{N}\texttt{rowSums}(X\mathbf{M})v)$
P3.5	$u\texttt{colSums}(\mathbf{M}X)+\mathbf{N}$
P3.6	$\texttt{rowSums}((\mathbf{M}X)^T) + (uv^T + \mathbf{N})v$
P3.7	$X\mathbf{N}u+\texttt{rowSums}(\mathbf{M})^T$
P3.8	$\mathbf{N}\odot\texttt{trace}(C + v\texttt{colSums}(\mathbf{M}X)C)$
P3.9	$X\odot\texttt{sum}(\texttt{colSums}(C)^T+\texttt{rowSums}(\mathbf{M}))+\mathbf{N}$
P3.10	$\mathbf{N}\odot\ \texttt{sum}((X + C)\mathbf{M})$

Table 7: LA pipelines used in micro-hybrid benchmark

Q1: The rewriting *modifies the pre-processing* of **N** by introducing V_1 and V_2; it also *pushes the filter-level selection* into the pre-processing part. Now, for the LA pipeline **P3.1** (see Table 7), HADAD applies several optimizations: (*i*) it rewrites $(uv^T + \mathbf{N}^T)v$ to $uv^Tv+\mathbf{N}^Tv$, where u and v are synthetic vectors of size 12×1 and 2M×1. First, **N** is ultra sparse, which makes the computation of $\mathbf{N}^Tv$ extremely efficient. Second, SystemML evaluates uv^Tv efficiently in one go without intermediates, taking advantage of **tsmm** operator (discussed earlier) and **mmchain** for matrix multiply chains, where the best way to evaluate it computes v^Tv first, which results in a scalar, instead of computing uv^T, which results in a dense matrix of size 1000×2M. Alone, SystemML is unable to exploit its own efficient operations for lack of awareness of the LA property $Av + Bv = (A + B)v$; (*ii*) HADAD also rewrites $\texttt{rowSums}(X\mathbf{M})$ into XV_4, where V_4 =$\texttt{rowSums}(\mathbf{T})$ +$K\texttt{rowSums}(\mathbf{U})$, by exploiting the property $\texttt{rowSums}(X\mathbf{M}) = X\texttt{rowSums}(\mathbf{M})$ together with Morpheus's rewrite rule: $\texttt{rowSums}(\mathbf{M}) \rightarrow \texttt{rowSums}(\mathbf{T}) + K\texttt{rowSums}(\mathbf{U})$, where **M** is the join's output table (matrix) of **T** and **U** base tables (feature matrices). The rewriting achieves up to 16.5× speed-up.

Q2: From hereafter, the rewriting of the pre-processing part of **N** is similar to the one in **Q1**. The speed-up of 2.5× in this query comes from rewriting the pre-processing part and turning **P3.2**:

[4]Matrix N is represented in Matrix Market Format (MTX) since it is sparse.



(a) **N matrix: 40K rows** (b) **N matrix: 20K rows** (c) **N matrix : 10K rows**

Figure 11: Micro-Hybrid Benchmark MIMIC

$u$colSums$((X\mathbf{M})^T)$+**N** to $u(XV_4)^T$+**N**, where u and X are synthetic matrices of size 2M×1 and 1000×2M, respectively. The rewriting is achieved by exploiting : colSums$((X\mathbf{M})^T)$ =rowSums$(X\mathbf{M})^T$ and rowSums$(X\mathbf{M})$ = XrowSums$(\mathbf{M})$ along with Morpheus's rewrite rule: rowSums$(\mathbf{M})$ → rowSums$(\mathbf{T})$ + KrowSums$(\mathbf{U})$. Both the rewriting and the original LA pipeline still introduce an unavoidable large dense intermediate of size 2M×1000.

Q3: In addition to the rewriting of the pre-processing part, for **P3.3**, HADAD (i) avoids introducing the large dense intermediate (**N**+X; 2M×1000) by distributing the multiplication of v (1000 × 1) and consider the sparsity of **N**, which again makes the multiplication with v more efficient; (ii) directly rewrites colSums$(\mathbf{M})$ to V_3 = [colSums$(\mathbf{T})$,colSums$(K)\mathbf{U}$] by utilizing Morpheus's rewrite rule: colSums$(\mathbf{M})$ → [colSums$(\mathbf{T})$,colSums$(K)\mathbf{U}$]. The obtained rewriting achieves 9.2× speed-up.

Q4: For this query, the speed-up of 3.6× attributes to rewriting the pre-processing part and optimizing **P3.4** by distributing the sum operation (sum$(A+B)$ =sum(A)+sum(B)), which prevents materializing a dense intermediate. The matrix C here is a synthetic dense matrix of size 2M×1000. Interestingly, SystemML includes this rewrite rule but did not apply it during optimization. Moreover, HADAD rewrites rowSums$(X\mathbf{M})$ to XV_4 (see **Q1**), which then enables SystemML to further optimize the chain of $\mathbf{N}XV_4v$ to , where the sizes of X and v are 1000×2M and 1×1000, respectively.

Q5: HADAD's rewriting speeds-up this query by 2.3×. It rewrites $u$colSums$(\mathbf{M}X)$ in **P3.5** to uV_3X (see **Q3** for V_3's definition). The view is exploited by HADAD due to utilizing colSums$(\mathbf{M}X)$ = colSums$(\mathbf{M})X$ together with the Morpheus's rewrite rule (shown in **Q3**) for pushing the colSums to the base tables (matrices) **U** and **T**. The found rewriting enables SystemML to optimize the matrix-chain multiplication by computing V_3X first, which results in 1×1000 matrix instead of computing uV_3 which results in 2M×12. The rewriting and the original pipeline still introduce unprevenbted dense intermediate of size 2M×1000.

Q6: The LA pipeline (**P3.6**) in this query is a variation of **P3.1**. In addition to distributing the multiplication of v, HADAD rewrites rowSums$((\mathbf{M}\ X)^T)$ to $(V_3X)^T$ (see **Q3** for V_3's definition) by exploiting rowSums$((\mathbf{M}X)^T)$ = colSums$(\mathbf{M}X)^T$ and colSums$(\mathbf{M}X)$ = colSums$(\mathbf{M})X$, all together with Morpheus's rewrite rule as illustrated in **Q3**. The obtained rewriting (along with the rewriting of the pre-processing part) achieves speed-up of 13.4× .

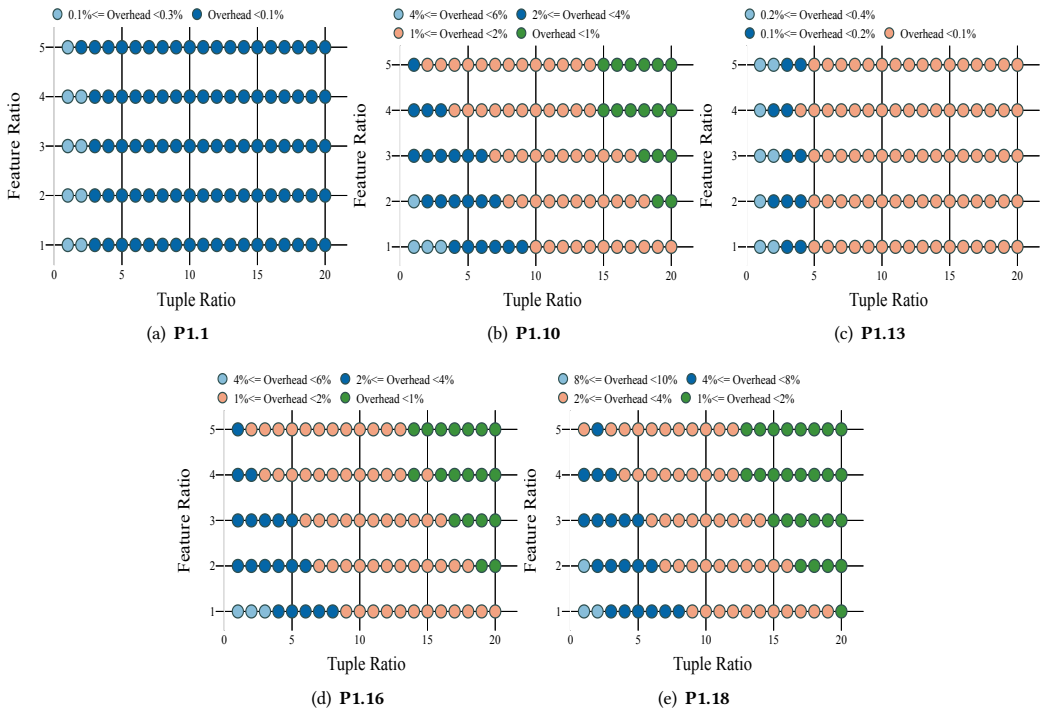
Q8: In addition to rewriting the pre-processing part, HADAD applies several optimizations for the analysis part (**P3.8**). First, it distributes the trace operation to prevent materializing a dense intermediate, which SystemML does not apply. Second, it exploits the view V_3 (see **Q3**) by utilizing colSums$(\mathbf{M}X)$ = colSums$(\mathbf{M})X$, which then enables SystemML to further optimize the matrix chain multiplication by computing $v((V_3X)C)$, where the size of v, X and C are 20K×1, 12×20K and 20K×20K, receptively. The element-wise multiplication with **N** in the rewiring and the original pipeline is extremely efficient since **N** is ultra-sparse. The rewriting speeds-up the query by 5.9×.

Q9: Beside rewriting the pre-processing part, the speed-up of 3× also attributes to turning sum(colSums$(C)^T$ +rowSums$(\mathbf{M})$) in **P3.9** to sum(V_5), where V_5 = $[C\mathbf{T}, CK\mathbf{U}]$. The view V_5 is utilized by exploiting the property sum$(C\mathbf{M})$ =sum(colSums$(C)^T \odot$ rowSums$(\mathbf{M})$) together with Morpheus's rewrite rule: $C\mathbf{M}$→ $[C\mathbf{T}, CK\mathbf{U}]$. The result of an element-wise multiplication with the dense matrix X in the rewriting and the original pipelines is a dene intermediate.

Q10: HADAD's rewrite leads to a speed-up of 3.9×. In addition to rewriting the pre-processing part, HADAD rewrites sum$((X+C)\mathbf{M})$ **P3.10** to sum$(X\mathbf{M})$+sum(V_5), where V_5 = $[C\mathbf{T}, CK\mathbf{U}]$. The view V_5 is utilized by exploiting the property $(X+C)\mathbf{M}$= $X\mathbf{M}$+$C\mathbf{M}$ together with Morpheus's rewrite rule: $C\mathbf{M}$→ $[C\mathbf{T}, CK\mathbf{U}]$ (see **Q9**). This optimization goes beyond SystemML's optimization since it does not consider distributing the multiplication of **M**, which then enables exploiting the view and distributing the sum operation to avoid introducing a large dense intermediate.

For the second and third runs of the benchmark (see Figures 10(b) and 10(c)), we vary the text-search selection condition ("Trump" and "US election") in the pre-processing part where the number of **N**'s rows are 1M and 0.5M, respectively. Accordingly, we change the size of synthetic matrices to ensure matrices' dimensions compatibility across the pipelines. Importantly, the rewriting time across all queries in the benchmark is very negligible compared to the original queries execution time.

MIMIC Dataset Preparation. MIMIC dataset [33] comprises health data for patients. The total size of the dataset is 46.6 GB, and it consists of : (i) all charted data for all patients and their hospital admission information, ICU stays, laboratory measurements, caregivers' notes, and prescriptions; (ii) the role of caregivers (e.g., MD stands for "medical doctor"), (iii) lab measurements (e.g., ABG stands for "arterial blood gas") and (iv) diagnosis related groups (DRG) codes



(a) **P1.1** (b) **P1.10** (c) **P1.13**

(d) **P1.16** (e) **P1.18**

Figure 12: HADAD RW_{find} overhead as a percentage (%) of the total time ($Q_{exec} + RW_{fond}$) for pipelines P1.1, P1.10, P1.13, P1.16 and P1.18 running on Morpheus

descriptions. We use subset of the dataset, which includes **Patients (P)**, **Admission (A)**, **Service (S)**, and **Callout (C)** tables. We convert tables' categorical features (columns) to numeric using one-hot encoding [**?**].

MIMIC Queries and Views. Similar to the Twitter's dataset benchmark, queries consist of two parts: (*i*) preprocessing (Q_{RA}) and (*ii*) analysis (Q_{LA}). In the Q_{RA} part, the queries construct two main matrices: **M** and **N**. The matrix **M** (40K×82;*dense*) is the join's output table (matrix) of **P** and **A**. The matrix **N** (40K×30K;*ultra-sparse*) is *patient-service* outcome (e.g., cancelled (1), serving (2), etc) matrix, constructed from joining **C** and **S** for all patients who are in "CCU" care unit. We fix the Q_{RA} part across all queries and vary the Q_{LA} part using a set of LA pipelines in Table 7, a well as **N** and **M** matrices. For views, we define three cross RA-LA materialized views: V_1, V_2 and V_3, which store the result of applying `rowSums`, `colSums` and matrix multiplication operations over **P** and **A** base tables (matrices), respectively. These views can only be found by *exploiting together LA properties* and *Morpheus rewrites' rules* in similar fashion as we detailed in Twitter's benchmark experiment.

Discussion The results exhibit similar trends to the Twitter's benchmark. The first run of the benchmark is shown in Figure 11(a); both matrices **M** and **N** are loaded in SystemML to be used for the analysis part (varied using the set of pipelines in Table 7). Before evaluating an LA pipeline, all queries filter **N** 's rows with *outcome* is equal to 2. HADAD applies the same set of optimizations as described in Twitter's benchmark. For the second and third runs of the queries (see Figures 11(b) and 11(c)), we construct the **N** matrix for patients who are in "TSICU" and "MICU" care units, where **N**'s rows are 20K and 10K, respectively.

9.2.3 ***Rewriting Time Overhead.*** For Morpheus, using the same experiment setup illustrated in §9.2.1, the rewriting time's overhead for a set $\mathcal{P}_M^{opt} \subset \mathcal{P}^{Opt}$ of 5 pipelines (running on Morpheus) is very negligible compared to the pipelines' execution time. For pipelines that contain matrix multiplication expressions such as **P1.1** and **P1.13**, the rewriting time is generally less than 0.1% of the total time, as shown in Figure 12. However, for the other pipelines **P1.10**, **P1.16**, and **P1.18**, which contain only aggregate operations, the rewriting time is up to 9% of the total time (when the data size is very small (0.32GB) and the computation is extremely efficient) and less than 1% (when the data size is large (19.2GB) and the computation is expensive) as shown in Figure 12. As for the micro-hybrid benchmark, the RW_{find} reaches 78ms and up to 145 ms.

9.3 Experiments Summary

We have shown that HADAD brings significant performance-saving across LA-oriented and cross RA-LA platforms without the need to modify their internals. It improves their performance by order of magnitudes on typical LA-based and hybrid pipelines. Moreover, as we confirm experimentally, the time spent searching for rewritings is a small fraction of the query execution time hence a worthwhile investment. In addition, the rewriting overhead of pipelines that are already in the optimal form is very negligible compared to the original pipeline execution time in the presence of sparse/dense matrices and using naïve and MNC-based cost models.

10 RELATED WORK AND CONCLUSION

LA Systems/Libraries. SystemML [21] offers high-level R-like linear algebra language and applies some logical LA pattern-based rewrites and physical execution optimizations, based on cost estimates for the latter. SparkMLlib [42] provides LA operations and built-in function implementations of popular ML algorithms, such as linear regression, etc. on Spark RDDs. R [8] and NumPy [7] are two of the most popular computing environments for statistical data analysis, widely used in academia and industry. They provide a high-level abstraction that can simplify the programming of numerical and statistical computations, by treating matrices as first-class citizens and by providing a rich set of built-in LA operations. However, LA properties in most of these systems remain unexploited, which makes them *miss opportunities to use their own highly efficient operators* (recall hybrid-scenario in §2). Our experiments (§9.1) show that LA pipelines evaluation in these systems can be sped up, by more than 10×, by our rewriting using (*i*) LA properties and (*ii*) materialized views.

Bridging the Gap: RA and LA. There has been a recent increase in research for unifying the execution of RA and LA expressions [9, 27, 35, 40]. A key limitation of these approaches is that *the semantics of LA operations remains hidden* behind built-in functions or UDFs, preventing performance-enhancing rewrites; as shown in §9.2.1.

Closer to our work, SPORES [49], SPOOF [23], LARA [38] and RAVEN [34]. SPORES and SPOOF optimize LA expressions, by converting them into RA, optimizing the latter, and then converting the result back to an (optimized) LA expression. They are restricted to a small set of selected LA operations (the ones that can be expressed in RA), while we support significantly more (§6.1), and model properties allowing to optimize with them. LARA relies on a declarative domain-specific language for collections and matrices, which can enable optimization (i.e., *selection pushdown*) across the two algebraic abstractions. It heavily focuses on low-level optimization such as exploiting the choice of data layouts for physical LA operators' optimization. RAVEN [34] takes a step forward by providing intermediate representation to enhance in-database model inferencing performance. It transforms classical ML models into equivalent neural networks (NN) to leverage highly optimized ML engines on CPU/GPU. [19, 25] study the expressive power of corss RA-LA [19] / LA [25] query languages. In contrast to HADAD, as all aforementioned solutions do not reason with constraints, they provide no capabilities for *holistic* semantic query optimizations including RA/LA views-based and LA pure rewritings; such optimizations can bring large performance saving as shown in (§9)

We see HADAD as complementary to all of these works, where it can can be naturally and portably applied on top of these platforms; while their existing code need not be rewritten.

Conclusion. HADAD is an extensible lightweight framework for optimizing hybrid analytics queries, based on the powerful intermediate abstraction of a *a relational model with integrity constraints*. HADAD extends the capability of [15] with a reduction from LA (or LA view)-based rewriting to relational rewriting under constraints. It enables a full exploration of rewrites using a large set of LA operations, with no modification to the execution platform. Our experiments show significant performance gains on various LA and hybrid workloads across popular LA and cross RA-LA platforms.

REFERENCES

[1] Amazon Review Data. https://nijianmo.github.io/amazon/index.html, Accessed August, 2020.
[2] Breeze Wiki. https://github.com/scalanlp/breeze/wiki, Accessed June, 2020.
[3] Kaggle Survey. https://www.kaggle.com/kaggle-survey-2019, Accessed June, 2020.
[4] Morpheus. https://github.com/lchen001/Morpheus, Accessed December, 2020.
[5] Native Blas in SystemDS. https://apache.github.io/systemds/native-backend, Accessed June, 2020.
[6] Netflix Movie Rating. https://www.kaggle.com/netflix-inc/netflix-prize-data, Accessed August, 2020.
[7] NumPy. https://numpy.org/, Accessed June, 2020.
[8] Project R. https://www.r-project.org/other-docs.html, Accessed June, 2020.
[9] Python/NumPy in Monetdb. https://tinyurl.com/11ljy21v, Accessed June, 2020.
[10] SparkMLlib. https://spark.apache.org/mllib, Accessed June, 2020.
[11] Technical Report. https://github.com/hadad-paper/HADAD_SIGMOD2021.
[12] Twitter API. https://developer.twitter.com/en/docs, Accessed January, 2021.
[13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A System for Large-Scale Machine Learning. In *USENIX*, pages 265–283, 2016.
[14] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[15] R. Alotaibi, D. Bursztyn, A. Deutsch, I. Manolescu, and S. Zampetakis. Towards Scalable Hybrid Stores: Constraint-based Rewriting to the Rescue. In *SIGMOD*, pages 1660–1677, 2019.
[16] R. Alotaibi, B. Cautis, A. Deutsch, M. Latrache, I. Manolescu, and Y. Yang. ESTOCADA: Towards Scalable Polystore Systems. *PVLDB*, pages 2949–2952, 2020.
[17] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. SparkSQl: Relational Data Processing in Spark. In *SIGMOD*, pages 1383–1394, 2015.
[18] S. Axler. *Linear Algebra Done Right*. Springer, 2015.
[19] P. Barceló, N. Higuera, J. Pérez, and B. Subercaseaux. On the Expressiveness of LARA: A Unified Language for Linear and Relational Algebra. In *ICDT*, pages 6:1–6:20, 2020.
[20] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, et al. Tfx: A Tnsorflow-based Production-Scale Machine Learning Platform. In *SIGKDD*, pages 1387–1395, 2017.
[21] M. Boehm, M. W. Dusenberry, D. Eriksson, A. V. Evfimievski, F. M. Manshadi, N. Pansare, B. Reinwald, F. R. Reiss, P. Sen, A. C. Surve, et al. SystemML: Declarative Machine Learning on Spark. *PVLDB*, pages 1425–1436, 2016.
[22] M. Boehm, A. V. Evfimievski, N. Pansare, and B. Reinwald. Declarative Machine Learning Classification of Basic Properties and Types. *arXiv:1605.05826*, 2016.
[23] M. Boehm, B. Reinwald, D. Hutchison, P. Sen, A. V. Evfimievski, and N. Pansare. On Optimizing Operator Fusion Plans for Large-Scale Machine Learning in SystemML. *PVLDB*, pages 1755–1768, 2018.
[24] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski, D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang. Probabilistic Demand Forecasting at Scale. *PVLDB*, pages 1694–1705, 2017.
[25] R. Brijder, F. Geerts, J. V. den Bussche, and T. Weerwag. On the Expressive Power of Query Languages for Matrices. *ACM Trans. Database Syst.*, pages 15:1–15:31, 2019.
[26] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Databases. In *ACM symposium on Theory of computing*, pages 77–90, 1977.
[27] L. Chen, A. Kumar, J. Naughton, and J. M. Patel. Towards Linear Algebra Over Normalized Data. *PVLDB*, pages 1214–1225, 2017.
[28] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *TOMS*, 2011.
[29] A. Deutsch. FOL Modeling of Integrity Constraints (Dependencies). In *Encyclopedia of Database Systems, Second Edition*. 2018.
[30] A. Deutsch, L. Popa, and V. Tannen. Query Reformulation with Constraints. In *SIGMOD*, pages 65–73, 2006.
[31] I. Ileana. *Query Rewriting Using Views : a Theoretical and Practical Perspective*. Theses, Télécom ParisTech, Oct. 2014.
[32] I. Ileana, B. Cautis, A. Deutsch, and Y. Katsis. Complete Yet Practical Search for Minimal Query Reformulations Under Constraints. In *SIGMOD*, pages 1015–1026, 2014.
[33] A. Johnson et al. MIMIC-III. *http://www.nature.com/articles/sdata201635*, 2016.
[34] K. Karanasos, M. Interlandi, D. Xin, F. Psallidas, R. Sen, K. Park, I. Popivanov, S. Nakandal, S. Krishnan, M. Weimer, et al. Extending Relational Query Processing with ML Inference. In *CIDR*, 2020.
[35] D. Kernert, F. Köhler, and W. Lehner. Bringing Linear Algebra Objects to Life in a Column-Criented in-Memory Database. In *In Memory Data Management and Analysis*, pages 44–55. Springer, 2013.
[36] A. Kumar, M. Boehm, and J. Yang. Data Management in Machine Learning: Challenges, Techniques, and Systems. In *SIGMOD*, pages 1717–1722, 2017.

[37] A. Kumar, R. McCann, J. Naughton, and J. M. Patel. Model Selection Management Systems: The Next Frontier of Advanced Analytics. In *SIGMOD*, pages 17–22, 2016.

[38] A. Kunft, A. Katsifodimos, S. Schelter, S. Breß, T. Rabl, and V. Markl. An Intermediate Representation for Optimizing Machine Learning Pipelines. *PVLDB*, pages 1553–1567, 2019.

[39] K. Kuttler. *Linear Algebra: Theory and Applications.* The Saylor Foundation, 2012.

[40] S. Luo, Z. J. Gao, M. Gubanov, L. L. Perez, and C. Jermaine. Scalable Linear Algebra on a Relational Database System. In *ICDE*, pages 1224–1238, 2018.

[41] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pages 281–297, 1967.

[42] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. MLlib: Machine Learning in Apache Spark. *The Journal of Machine Learning Research*, pages 1235–1241, 2016.

[43] M. Milani, S. Hosseinpour, and H. Pehlivan. Rule-based Production of Mathematical Expressions. *Mathematics*, 6:254, 11 2018.

[44] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[45] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden Technical Debt in Machine Learning Systems. In *NeurIPS*, pages 2503–2511, 2015.

[46] J. Sommer, M. Boehm, A. V. Evfimievski, B. Reinwald, and P. J. Haas. MNC: Structure-Exploiting Sparsity Estimation for Matrix Expressions. In *SIGMOD*, pages 1607–1623, 2019.

[47] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska. Automating Model Search for Large Scale Machine Learning. In *ACM Symposium on Cloud Computing*, pages 368–380, 2015.

[48] A. Thomas and A. Kumar. A Comparative Evaluation of Systems for Scalable Linear Algebra-based Analytics. *PVLDB*, pages 2168–2182, 2018.

[49] Y. R. Wang, S. Hutchison, J. Leang, B. Howe, and D. Suciu. SPORES: Sum-Product Optimization via Relational Equality Saturation for Large Scale Linear Algebra. *PVLDB*, pages 1919–1932, 2020.

A L_{ops} OPERATIONS PROPERTIES (LA_{prop}) CAPTURED AS INTEGRITY CONSTRAINTS

Table 8: L_{ops} Operations Properties (LA_{prop}) Captured as Integrity Constraints

LA Property	Relational Encoding as Integrity Constraints
Addition of Matrices	
$M + N = N + M$	$\forall M, N, R\ \mathsf{add}_M(M, N, R) \rightarrow \mathsf{add}_M(N, M, R)$
$(M + N) + D = M + (N + D)$	$\forall M, N, D, R_1, R_2\ \mathsf{add}_M(M, N, R_1) \wedge \mathsf{add}_M(R_1, D, R_2) \rightarrow$ $\exists R_3\ \mathsf{add}_M(N, D, R_3) \wedge \mathsf{add}_M(M, R_3, R_2)$
$c(M + N) = cM + cN$	$\forall c, M, N, R_1, R_2\ \mathsf{add}_M(M, N, R_1) \wedge \mathtt{multi}_{MS}(c, R_1, R_2) \rightarrow$ $\exists R_3, R_4 \mathtt{multi}_{MS}(c, M, R_3) \wedge \mathtt{multi}_{MS}(c, N, R_4) \wedge \mathsf{add}_M(R_3, R_4, R_2)$
$(c + d)M = cM + dM$	$\forall c, d, s, M, R_1 add_S(c, d, s) \wedge \mathtt{multi}_{MS}(s, M, R_1) \rightarrow$ $\exists R_2, R_3 \mathtt{multi}_{MS}(c, M, R_2) \wedge \mathtt{multi}_{MS}(d, M, R_3) \wedge \mathsf{add}_M(R_2, R_3, R_1)$
$M + 0 = M$	$\rightarrow \exists Zero(O)$ $\forall M, n, O\ name(M, n) \wedge Zero(O) \rightarrow \mathsf{add}_M(M, O, M)$ $\forall O\ Zero(O) \rightarrow \mathsf{add}_M(O, O, O)$
Product of Matrices	
$(MN)D = M(ND)$	$\forall M, N, D, R_1, R_2\ \mathtt{multi}_M(M, N, R_1) \wedge \mathtt{multi}_M(R_1, D, R_2) \rightarrow$ $\exists R_3 \mathtt{multi}_M(N, D, R_3) \wedge \mathtt{multi}_M(M, R_3, R_2)$
$M(N + D) = MN + MD$	$\forall M, N, D, R_1, R_2\ \mathsf{add}_M(N, D, R_1) \wedge \mathtt{multi}_M(M, R_1, R_2) \rightarrow$ $\exists R_3, R_4\ \mathtt{multi}_M(M, N, R_3) \wedge \mathtt{multi}_M(M, D, R_4) \wedge \mathsf{add}_M(R_3, R_4, R_2)$
$(M + N)D = MD + MD$	$\forall M, N, D, R_1, R_2 \mathsf{add}_M(M, N, R_1) \wedge \mathtt{multi}_M(R_1, D, R_2) \rightarrow$ $\exists R_3, R_4 \mathtt{multi}_M(M, D, R_3) \wedge \mathtt{multi}_M(N, D, R_4) \wedge \mathsf{add}_M(R_3, R_4, R_2)$
$d(MN) = (dM)N$	$\forall d, M, N, R_1, R_2\ \mathtt{multi}_M(M, N, R_1) \wedge \mathtt{multi}_{MS}(d, R_1, R_2) \rightarrow$ $\exists R_3 \mathtt{multi}_{MS}(d, M, R_3) \wedge \mathtt{multi}_M(R_3, N, R_2)$
$c(dM) = (cd)M$	$\forall c, dM, R_1, R_2 \mathtt{multi}_{MS}(d, M, R_1) \wedge \mathtt{multi}_{MS}(c, R_1, R_2) \rightarrow$ $\exists s\ multi_S(c, d, s) \wedge \mathtt{multi}_{MS}(s, M, R_2)$
$I_k M = M = M I_z$	$\forall M, n, k, z\ name(M, n), size(M, k, z) \rightarrow \exists I_1\ Identity(I_1), size(I_1, k, k)$ $\forall M, n, k, z\ name(M, n), size(M, k, z) \rightarrow \exists I_1\ Identity(I_1), size(I_1, z, z)$ $\forall M, I_1, n, k, z\ name(M, n), size(M, k, z), Identity(I_1), size(I_1, k, k) \rightarrow \mathtt{multi}_M(I_1, M, M)$ $\forall M, I_1, n, k, z\ name(M, n), size(M, k, z), Identity(I_1), size(I_1, z, z) \rightarrow \mathtt{multi}_M(M, I_1, M)$
Transposition of Matrices	
$(MN)^T = (N)^T (M)^T$	$\forall M, N, R_1, R_2 \mathtt{multi}_M(M, N, R_1) \wedge \mathtt{tr}(R_1, R_2) \rightarrow$ $\exists R_3, R_4 \mathtt{tr}(M, R_3) \wedge \mathtt{tr}(N, R_4) \wedge \mathtt{multi}_M(R_4, R_3, R_2)$
$(M + N)^T = (M)^T + (N)^T$	$\forall M, N, R_1, R_2 \mathsf{add}_M(M, N, R_1) \wedge \mathtt{tr}(R_1, R_2) \rightarrow$ $\exists R_3, R_4 \mathtt{tr}(M, R_3) \wedge \mathtt{tr}(N, R_4) \wedge \mathsf{add}_M(R_3, R_4, R_2)$
$(cM)^T = c(M)^T$	$\forall c, M, R_1, R_2 \mathtt{multi}_{MS}(c, M, R_1) \wedge \mathtt{tr}(R_1, R_2) \rightarrow$ $\exists R_3 \mathtt{tr}(M, R_3) \wedge \mathtt{multi}_{MS}(c, R_3, R_2)$
$((M)^T)^T = M$	$\forall n, Mname(M, n) \rightarrow \exists R_1 \mathtt{tr}(M, R_1) \wedge \mathtt{tr}(R_1, M)$
$(I)^T = I$, where I is identity matrix $(O)^T = O$, where O is zero matrix	$\forall I_1 Identity(I_1) \rightarrow \mathtt{tr}(I_1, I_1)$ $\forall O_1^i Zero(O_1^i) \rightarrow \mathtt{tr}(O_1^i, O_1^i)$
Inverses of Matrices	
$((M)^{-1})^{-1} = M$	$\forall n, Mname(M, n) \rightarrow \exists R_1 \mathtt{inv}_M(M, R_1) \wedge \mathtt{inv}_M(R_1, M)$
$(MN)^{-1} = (N)^{-1}(M)^{-1}$	$\forall M, N, R_1, R_2 \mathtt{multi}_M(M, N, R_1) \wedge \mathtt{inv}_M(R_1, R_2) \rightarrow$ $\exists R_3, R_4 \mathtt{inv}_M(M, R_3) \wedge \mathtt{inv}_M(N, R_4) \wedge \mathtt{multi}_M(R_4, R_3, R_2)$
$((M)^T)^{-1} = ((M)^{-1})^T$	$\forall M, R_1, R_2 \mathtt{tr}(M, R_1) \wedge \mathtt{inv}_M(R_1, R_2) \rightarrow$ $\exists R_3 \mathtt{inv}_M(M, R_3) \wedge \mathtt{tr}(R_3, R_2)$
$((kM))^{-1} = k^{-1}(M)^{-1}$	$\forall k, M, R_1, R_2 \mathtt{multi}_{MS}(k, M, R_1) \wedge \mathtt{inv}_M(R_1, R_2) \rightarrow$ $\exists R_3, s\ inv_S(k, s) \wedge \mathtt{inv}_M(M, R_3) \wedge \mathtt{multi}_{MS}(s, R_3, R_2)$
$M^{-1}M = I = MM^{-1}$	$\forall M, R_1, R_2\ \mathtt{inv}_M(M, R_1) \wedge \mathtt{multi}_M(R_1, M, R_2) \rightarrow Identity(R_2)$ $\forall M, R_1, R_2\ \mathtt{inv}_M(M, R_1) \wedge \mathtt{multi}_M(M, R_1, R_2) \rightarrow Identity(R_2)$

Table 9: L_{ops} Operations Properties (LA_{prop}) Captured as Integrity Constraints

LA Property	Relational Encoding as Integrity Constraints
Determinant of Matrices	
$\mathtt{det}(MN) = \mathtt{det}(M) * \mathtt{det}(N)$	$\forall M, N, R_1, d\ \mathtt{multi}_M(M, N, R_1) \wedge \mathtt{det}(R_1, d) \rightarrow$ $\exists d_1, d_2 \mathtt{det}(M, d_1) \wedge \mathtt{det}(N, d_2) \wedge multi_S(d_1, d_2, d)$
$\mathtt{det}((M)^T) = \mathtt{det}(M)$	$\forall M, R_1, d\ \mathtt{tr}(M, R_1) \wedge \mathtt{det}(R_1, d) \rightarrow \mathtt{det}(M, d)$
$\mathtt{det}((M)^{-1}) = (\mathtt{det}(M))^{-1}$	$\forall M, R_1, d\ \mathtt{inv}_M(M, R_1) \wedge \mathtt{det}(R_1, d) \rightarrow \exists d_1 \mathtt{det}(M, d_1) \wedge inv_S(d_1, d)$
$\mathtt{det}((cM)) = c^k \mathtt{det}(M)$	$\forall M, c, k, d^O\ size(M, k, k) \wedge \mathtt{multi}_{MS}(c, M, d^O) \rightarrow \exists s_1, s_2$ $pow(c, k, s_1) \wedge \mathtt{det}(M, s_2) \wedge multi_S(s_1, s_2, d^O)$
$\mathtt{det}((I)) = 1$	$\forall I_1, d^O Identity(I_1) \wedge \mathtt{det}(I_1, d^O) \rightarrow d^O = 1$
Adjoint of Matrices	
$\mathtt{adj}(M) = R^T$	$\forall M, R_1\ \mathtt{adj}(M, R_1) \rightarrow \exists R_2\ cof(M, R_2) \wedge \mathtt{tr}(R_2, R_1)$
$\mathtt{adj}(M)^T = \mathtt{adj}(M^T)$	$\forall M, R_1, R_2\ \mathtt{adj}(M, R_1) \wedge \mathtt{tr}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{tr}(M, R_3) \wedge \mathtt{adj}(R_3, R_2)$
$\mathtt{adj}(M)^{-1} = \mathtt{adj}(M^{-1})$	$\forall M, R_1, R_2\ \mathtt{adj}(M, R_1) \wedge \mathtt{inv}_M(R_1, R_2) \rightarrow \exists R_3\ \mathtt{inv}_M(M, R_3) \wedge \mathtt{adj}(R_3, R_2)$
$\mathtt{adj}(MN) = \mathtt{adj}(NM)$	$\forall M, R_1, R_2\ \mathtt{multi}_M(M, N, R_1) \wedge \mathtt{adj}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{multi}_M(N, M, R_3) \wedge \mathtt{adj}(R_3, R_2)$
Trace of Matrices	
$\mathtt{trace}(M + N) = \mathtt{trace}(M) + \mathtt{trace}(N)$	$\forall M, N, R_1, s_1\ \mathtt{add}_M(M, N, R_1) \wedge$ $\mathtt{trace}(R_1, s_1) \rightarrow \exists s_2, s_3\ \mathtt{trace}(M, s_2) \wedge \mathtt{trace}(N, s_3) \wedge add_s(s_2, s_3, s_1)$
$\mathtt{trace}(MN) = \mathtt{trace}(NM)$	$\forall M, N, R_1, s_1\ \mathtt{multi}_M(M, N, R_1) \wedge$ $\mathtt{trace}(R_1, s_1) \rightarrow \exists R_2\ \mathtt{multi}_M(N, M, R_2) \wedge \mathtt{trace}(R_2, s_1)$
$\mathtt{trace}(M^T) = \mathtt{trace}(M)$	$\forall M, R_1, s_1\ \mathtt{tr}(M, R_1) \wedge \mathtt{trace}(R_1, s_1) \rightarrow \mathtt{trace}(M, s_1)$
$\mathtt{trace}(cM) = c\mathtt{trace}(M)$	$\forall M, R_1, c, s_1\ \mathtt{multi}_{MS}(c, M, R_1) \wedge \mathtt{trace}(R_1, s_1) \rightarrow \exists s_2 \mathtt{trace}(M, s_2) \wedge multi_s(c, s_1, s_2)$
$\mathtt{trace}(I_k) = k$	$\forall I_O, k, s_1\ Identity(I_O) \wedge size(I_O, k, k) \wedge \mathtt{trace}(I_O, s_1) \rightarrow s_1 = k$
Direct Sum	
$(M \oplus N) + (C \oplus D) = (M + C) \oplus (N + D)$	$\forall M, N, R_1, C, D, R_2, R_3\ \mathtt{sum}_D(M, N, R_1)$ $\wedge \mathtt{sum}_D(C, D, R_2) \wedge \mathtt{add}_M(R_1, R_2, R_3) \rightarrow \exists R_4, R_5 \mathtt{add}_M(M, C, R_4) \wedge \mathtt{add}_M(N, D, R_5)$
$(M \oplus N) + (C \oplus D) = (MC) \oplus (ND)$	$\forall M, N, R_1, C, D, R_2, R_3\ \mathtt{sum}_D(M, N, R_1)$ $\wedge \mathtt{sum}_D(C, D, R_2) \wedge \mathtt{multi}_M(R_1, R_2, R_3) \rightarrow \exists R_4, R_5 \mathtt{multi}_M(M, C, R_4) \wedge \mathtt{multi}_M(N, D, R_5)$
$c(M \oplus N) = (cM \oplus cN)$	$\forall M, N, R_1, c, R_2\ \mathtt{sum}_D(M, N, R_1)$ $\wedge \mathtt{multi}_{MS}(c, R_1, R_2) \rightarrow \exists R_3, R_4 \mathtt{multi}_{MS}(M, c, R_3) \wedge \mathtt{multi}_{MS}(N, c, R_4) \wedge \mathtt{sum}_D(R_3, R_4, R_2)$
Exponential of Matrices	
$\mathtt{exp}(0) = I$	$\forall O, R_1\ Zero(O) \wedge \mathtt{exp}(O, R_1) \rightarrow Identity(R_1)$
$\mathtt{exp}(M^T) = \mathtt{exp}(M)^T$	$\forall M, R_1, R_2 \mathtt{tr}(M, R_1) \wedge \mathtt{exp}(R_1, R_2) \rightarrow \exists R_3 \mathtt{exp}(M, R_3) \wedge \mathtt{exp}(R_3, R_2)$

Table 10: Matrix Decompositions Properties Captured as Integrity Constraints

Decomposition Property	Relational Encoding as Integrity Constraints
Cholesky Decomposition (CD)	
$\texttt{CHO}(M) = L$ such that $M = LL^T$, where M is symmetric positive definite	$\forall M\ type(M, \text{“}S\text{”}) \rightarrow \exists\ L_1 \exists L_2\ \texttt{CHO}(M, L_1) \wedge type(L_1, \text{“}L\text{”}) \wedge$ $\texttt{tr}(L_1, L_2) \wedge \texttt{multi}_M(L_1, L_2, M)$
QR Decomposition	
$QR(M) = [Q, R]$ such that $M = QR$	$\forall M \forall n \forall k\ name(M, n) \wedge size(M, k, z) \rightarrow\ \exists Q_1, R_1$ $QR(M, Q_1, R_1) \wedge type(Q_1, \text{“}O\text{”}) \wedge size(Q_1, k, k) \wedge type(R_1, \text{“}U\text{”}) \wedge$ $size(R_1, k, z) \wedge \texttt{multi}_M(Q_1, R_1, M)$ $\forall Q_1\ type(Q_1, \text{“}O\text{”}) \wedge size(Q_1, k, k) \rightarrow \exists I_1\ QR(Q_1, Q_1, I_1) \wedge identity(I_1)$ $\wedge size(I_1, k, k) \wedge \texttt{multi}_M(Q_1, I_1, Q_1)$ $\forall R_1\ type(R_1, \text{“}U\text{”}) \wedge size(Q_1, k, z) \rightarrow \exists I_1\ QR(R_1, I_1, R_1) \wedge identity(I_1)$ $\wedge size(I_1, k, k) \wedge \texttt{multi}_M(I_1, R_1, R_1)$ $\forall I_1\ identity(I_1) \rightarrow QR(I_1, I_1, I_1)$
LU Decomposition	
$\texttt{LU}(M) = [L, U]$ such that $M = LU$	$\forall M \forall n \forall k\ name(M, n) \wedge size(M, k, z) \rightarrow\ \exists L_1, U_1$ $\texttt{LU}(M, L_1, U_1) \wedge type(L_1, \text{“}L\text{”}) \wedge size(L_1, k, z) \wedge type(U_1, \text{“}U\text{”}) \wedge$ $size(U_1, z, z) \wedge \texttt{multi}_M(L_1, U_1, M)$ $\forall L_1\ type(L_1, \text{“}L\text{”}) \wedge size(L_1, k, z) \rightarrow \exists I_1\ \texttt{LU}(L_1, L_1, I_1) \wedge identity(I_1)$ $\wedge \texttt{multi}_M(L_1, I_1, L_1) \wedge size(I_1, z, z)$ $\forall U_1\ type(U_1, \text{“}U\text{”}) \wedge size(U_1 z, z) \rightarrow \exists I_1\ \texttt{LU}(U_1, I_1, U_1) \wedge identity(I_1)$ $\wedge size(I_1 z, z) \wedge \texttt{multi}_M(I_1, U_1, U_1)$ $\forall I_1\ identity(I_1) \rightarrow \texttt{LU}(I_1, I_1, I_1)$
Pivoted LU Decomposition	
$\texttt{LUP}(M) = [L, U, P]$ such that $PM = LU$, where M is a square matrix	$\forall M \forall n \forall k\ name(M, n) \wedge size(M, k, z) \rightarrow\ \exists L_1, U_1, P_1, R_1$ $\texttt{LUP}(M, L_1, U_1, P_1) \wedge type(L_1, \text{“}L\text{”}) \wedge type(U_1, \text{“}U\text{”}) \wedge type(P_1, \text{“}P\text{”})$ $\wedge \texttt{multi}_M(L_1, U_1, R_1) \wedge \texttt{multi}_M(P_1, M, R_1)$ $\forall L_1\ type(L_1, \text{“}L\text{”}) \wedge size(L_1, k, z) \rightarrow \exists I_1, I_2\ \texttt{LUP}(L_1, L_1, I_1, I_2) \wedge$ $identity(I_1) \wedge identity(I_2) \wedge size(I_1, z, z) \wedge size(I_2, k, k)$ $\wedge \texttt{multi}_M(L_1, I_1, L_1) \wedge \texttt{multi}_M(I_2, L_1, L_1)$ $\forall U_1\ type(U_1, \text{“}U\text{”}) \rightarrow \exists I_1\ \texttt{LUP}(U_1, I_1, U_1, I_1) \wedge$ $identity(I_1) \wedge \texttt{multi}_M(I_1, U_1, U_1)$ $\forall I_1\ identity(I_1) \rightarrow \texttt{LU}(I_1, I_1, I_1)$

B SYSTEMML REWRITE RULES ENCODED AS INTEGRITY CONSTRAINTS

Table 11: SystemML Algebraic Aggregate Rewrite Rules Captured as Integrity Constraints

SystemML Algebraic Simplification Rule	Integrity Constraints $\mathcal{MMC}_{StatAgg}$
UnnecessaryAggregates	
sum(t(M))-> sum(M)	$\forall M, R_1, s\ \mathtt{tr}(M, R_1), \mathtt{sum}(R_1, s) \rightarrow \mathtt{sum}(M, s)$
sum(rev(M))-> sum(M)	$\forall M, R_1, s\ rev(M, R_1), \mathtt{sum}(R_1, s) \rightarrow \mathtt{sum}(M, s)$
sum(rowSums(M))-> sum(M)	$\forall M, R_1, s\ \mathtt{rowSums}(M, R_1), \mathtt{sum}(R_1, s) \rightarrow \mathtt{sum}(M, s)$
sum(colSums(M))-> sum(M)	$\forall M, R_1, s\ \mathtt{colSums}(M, R_1), \mathtt{sum}(R_1, s) \rightarrow \mathtt{sum}(M, s)$
min(rowMin(M))-> min(M)	$\forall M, R_1, s\ \mathtt{rowMin}(M, R_1), \mathtt{min}(R_1, s) \rightarrow \mathtt{min}(M, s)$
min(colMin(M))-> min(M)	$\forall M, R_1, s\ \mathtt{colMin}(M, R_1), \mathtt{min}(R_1, s) \rightarrow \mathtt{min}(M, s)$
max(colmax(M))-> max(M)	$\forall M, R_1, s\ \mathtt{colMax}(M, R_1), \mathtt{max}(R_1, s) \rightarrow \mathtt{max}(M, s)$
max(rowmax(M))-> max(M)	$\forall M, R_1, s\ \mathtt{rowMax}(M, R_1), \mathtt{max}(R_1, s) \rightarrow \mathtt{max}(M, s)$
pushdownUnaryAggTransposeOp	
rowSums(t(M))->t(colSums(M))	$\forall M, R_1, R_2\ \mathtt{tr}(M, R_1) \wedge \mathtt{rowSums}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{colSums}(M, R_3) \wedge \mathtt{tr}(R_3, R_2)$
colSums(t(M))->t(rowSums(M))	$\forall M, R_1, R_2\ \mathtt{tr}(M, R_1) \wedge \mathtt{colSums}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{rowSums}(M, R_3) \wedge \mathtt{tr}(R_3, R_2)$
rowMean(t(M))->t(colMean(M))	$\forall M, R_1, R_2\ \mathtt{tr}(M, R_1) \wedge \mathtt{colMean}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{rowMean}(M, R_3) \wedge \mathtt{tr}(R_3, R_2)$
colMean(t(M))->t(rowMean(M))	$\forall M, R_1, R_2\ \mathtt{tr}(M, R_1) \wedge \mathtt{rowMean}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{colMean}(M, R_3 \wedge \mathtt{tr}(R_3, R_2)$
rowVar(t(M))->t(colVar(M))	$\forall M, R_1, R_2\ \mathtt{tr}(M, R_1) \wedge \mathtt{rowVar}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{colVar}(M, R_3) \wedge \mathtt{tr}(R_3, R_2)$
colVar(t(X))->t(rowVar(X))	$\forall M, R_1, R_2\ \mathtt{tr}(M, R_1) \wedge \mathtt{colVar}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{rowVar}(M, R_3) \wedge \mathtt{tr}(R_3, R_2)$
rowMax(t(M))->t(colMax(M))	$\forall M, R_1, R_2\ \mathtt{tr}(M, R_1) \wedge \mathtt{rowMax}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{colMax}(M, R_3) \wedge \mathtt{tr}(R_3, R_2)$
colMax(t(M))->t(rowMax(M))	$\forall M, R_1, R_2\ \mathtt{tr}(M, R_1) \wedge \mathtt{colMax}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{rowMax}(M, R_3) \wedge \mathtt{tr}(R_3, R_2)$
rowMin(t(M))->t(colMin(M))	$\forall M, R_1, R_2\ \mathtt{tr}(M, R_1) \wedge \mathtt{rowMin}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{colMin}(M, R_3) \wedge \mathtt{tr}(R_3, R_2)$
colMin(t(M))->t(rowMin(M))	$\forall M, R_1, R_2\ \mathtt{tr}(M, R_1) \wedge \mathtt{colMin}(R_1, R_2) \rightarrow \exists R_3\ \mathtt{rowMin}(M, R_3) \wedge \mathtt{tr}(R_3, R_2)$
simplifyTraceMatrixMult	
trace(MN)->sum(M⊙t(N))	$\forall M, N, R_1, r\ \mathtt{multi}_M(M, N, R_1) \wedge \mathtt{trace}(R_1, r) \rightarrow$ $\exists R_3, R_4 \mathtt{tr}(N, R_3) \wedge \mathtt{multi}_E(M, R_3, R_4) \wedge \mathtt{sum}(R_4, r)$
simplifySumMatrixMult	
sum(MN) -> sum(t(colSums(M))⊙rowSums(N))	$\forall M, N, R_1, r\ \mathtt{multi}_M(M, N, R_1) \wedge \mathtt{sum}(R_1, r) \rightarrow$ $\exists R_2, R_3, R_4, R_5\ \mathtt{colSums}(M, R_2) \wedge \mathtt{tr}(R_2, R_3) \wedge \mathtt{rowSums}(N, R_4) \wedge$ $\mathtt{multi}_E(R_3, R_4, R_5), \wedge \mathtt{sum}(R_5, r)$
colSums(MN) -> colSums(M)N	$\forall M, N, R_1, R_2\ \mathtt{multi}_M(M, N, R_1) \wedge \mathtt{colSums}(R_1, R_2) \rightarrow$ $\exists R_3\ \mathtt{colSums}(M, R_3) \wedge \mathtt{multi}_M(R_3, N, R_2)$
rowSums(MN) -> MrowSums(N)	$\forall M, N, R_1, R_2\ \mathtt{multi}_M(M, N, R_1) \wedge \mathtt{rowSums}(R_1, R_2) \rightarrow$ $\exists R_3\ \mathtt{rowSums}(N, R_3) \wedge \mathtt{multi}_M(M, R_3, R_2)$
colSums(M)->M if x is row vector	$\forall M, n, i\ name(M, n) \wedge size(M, \text{“1”}, j) \rightarrow \mathtt{colSums}(M, M)$
colMean(M)->M if x is row vector	$\forall M, n, j\ name(M, n) \wedge size(M, \text{“1”}, j) \rightarrow \mathtt{colSums}(M, M)$
colVar(M)->M if x is row vector	$\forall M, n, j\ name(M, n) \wedge size(v\text{“1”}, j) \rightarrow \mathtt{colVar}(M, M)$
colMax(M)->M if x is row vector	$\forall M, n, j\ name(M, n) \wedge size(M, \text{“1”}, j) \rightarrow \mathtt{colMax}(M, M)$
colMin(M)->M if x is row vector	$\forall M, n, j\ name(M, n) \wedge size(M, \text{“1”}, j) \rightarrow \mathtt{colMin}(M, M)$
colSums(M)->sum(M) if x is col vector	$\forall M, i, R_1\ \mathtt{colSums}(M, R_1) \wedge size(M, i, \text{“1”}) \rightarrow \mathtt{sum}(M, R_1)$
colMean(M)->mean(M) if x is col vector	$\forall M, i, R_1\ \mathtt{colMean}(M, R_1) \wedge size(M, i, \text{“1”}) \rightarrow \mathtt{mean}(M, R_1)$
colMax(X)->max(M) if x is col vector	$\forall M, i, R_1\ \mathtt{colMax}(M, R_1) \wedge size(M, i, \text{“1”}) \rightarrow \mathtt{max}(M, R_1)$
colMin(M)->min(X) if x is col vector	$\forall M, i, R_1\ \mathtt{colMin}(M, R_1) \wedge size(M, i, \text{“1”}) \rightarrow \mathtt{min}(M, R_1)$
colVar(M)->var(M) if x is col vector	$\forall M, i, R_1\ \mathtt{colVar}(M, R_1) \wedge size(M, i, \text{“1”}) \rightarrow \mathtt{var}(M, R_1)$

SystemML Algebraic Simplification Rule	Integrity Constraints $\mathcal{MMC}_{StatAgg}$
simplifyRowWiseAgg	
rowSums(M)->M if x is col vector	$\forall M, n, i\ name(M, n) \wedge size(M, i, \text{“1”}) \rightarrow \texttt{rowSums}(M, M)$
rowMean(M)->M if x is col vector	$\forall M, n, i\ name(M, n) \wedge size(M, i, \text{“1”}) \rightarrow \texttt{rowMean}(M, M)$
rowVar(M)->M if x is col vector	$\forall M, n, i\ name(M, n) \wedge size(M, i, \text{“1”}) \rightarrow \texttt{rowVar}(M, M)$
rowMax(M)->M if x is col vector	$\forall M, n, i\ name(M, n) \wedge size(M, i, \text{“1”}) \rightarrow \texttt{rowMax}(M, M)$
rowMin(M)->M if x is col vector	$\forall M, n, i\ name(M, n) \wedge size(M, i, \text{“1”}) \rightarrow \texttt{rowMax}(M, M)$
rowSums(M)->sum(M) if x is row vector	$\forall M, j, R_1\ \texttt{rowSums}(M, R_1) \wedge size(M, \text{“1”}, j) \rightarrow \texttt{sum}(M, R_1)$
rowMean(M)->mean(M) if x is row vector	$\forall M, j, R_1\ \texttt{rowMean}(M, R_1) \wedge size(M, \text{“1”}, j) \rightarrow \texttt{mean}(M, R_1)$
rowMax(M)->max(M) if x is row vector	$\forall M, j, R_1\ \texttt{rowMax}(M, R_1) \wedge size(M, \text{“1”}, j) \rightarrow \texttt{max}(M, R_1)$
rowMin(X)->min(M) if x is row vector	$\forall M, j, R_1\ \texttt{rowMin}(M, R_1) \wedge size(M, \text{“1”}, j) \rightarrow \texttt{min}(M, R_1)$
rowVar(X)->var(M) if x is row vector	$\forall M, j, R_1\ \texttt{rowVar}(M, R_1) \wedge size(M, \text{“1”}, j) \rightarrow \texttt{var}(M, R_1)$
pushdownSumOnAdd	
sum(M+N) -> sum(A)+sum(B)	$\forall M, N, s\ \texttt{add}_M M, N, s_1) \wedge \texttt{sum}(M, s_1) \rightarrow$ $\exists s_2, s_3\ \texttt{sum}(M, s_2) \wedge \texttt{sum}(N, s_3) \wedge \wedge add_s(s_2, s_3, s_1)$
ColSumsMVMult	
colSums(M*N) -> t(M)N	$\forall M, N, R_1, R_2, i\ size(N, i, \text{“1”}) \wedge \texttt{multi}_E(M, N, R_1) \wedge \texttt{colSums}(R_1, R_2)$ $\exists R_3 \texttt{tr}(M, R_3) \wedge \texttt{multi}_M(R_3, N, R_2)$
rowSums(M*M) -> Mt(N)	$\forall M, N, R_1, R_2, j\ size(N, \text{“1”}, j) \wedge \texttt{multi}_E(M, N, R_1) \wedge \texttt{rowSums}(R_1, R_2)$ $\exists R_3 \texttt{tr}(N, R_3) \wedge \texttt{multi}_M(M, R_3, R_2)$

C $\mathcal{P}^{\neg Opt}$ AND $\mathcal{P}^{Views}$ PIPELINES REWRITES

No.	Rewrite	No.	Rewrite	No.	Rewrite
P1.1	$N^T M^T$	P1.2	$(A+B)^T$	P1.3	$(DC)^{-1}$
P1.4	$Av_1 + Bv_1$	P1.5	D	P1.6	$s_1\texttt{trace}(D)$
P1.7	A	P1.8	$(s_1+s_2)A$	P1.9	$\texttt{det}(D)$
P1.10	$\texttt{colSums}(A)^T$	P1.11	$\texttt{colSums}(A+B)^T$	P1.12	$\texttt{colSums}(M)N$
P1.13	$\texttt{sum}(\texttt{colSums}(M)^T * \texttt{rowSums}(N))$	P1.14	$\texttt{sum}(\texttt{colSums}(M)^T * \texttt{rowSums}(N))$	P1.15	$M(NM)$
P1.16	$\texttt{sum}(A)$	P1.17	$\texttt{det}(C) * \texttt{det}(D) * \texttt{det}(C)$	P1.18	$\texttt{sum}(A)$
P1.25	$M \odot (N^T/(M(NN^T)))$				

Table 12: $\mathcal{P}^{\neg Opt}$ Pipelines (Part 1) Rewrites

No.	Rewrite	No.	Rewrite	No.	Rewrite
P2.1	$\texttt{trace}(C) + \texttt{trace}(D)$	P2.2	$1/\texttt{det}(D)$	P2.3	$\texttt{trace}(D)$
P2.4	$s_1(A+B)$	P2.5	$1/\texttt{det}((C+D))$	P2.6	$(D^{-1}C)^T$
P2.7	C	P2.8	$\texttt{det}(C) * \texttt{det}(D)$	P2.9	$\texttt{trace}(DC) + \texttt{trace}(D)$
P2.10	$MrowSumsN)$	P2.11	$\texttt{sum}(A) + \texttt{sum}(B)$	P2.12	$\texttt{sum}(\texttt{colSums}(M)^T * \texttt{rowSums}(N)$
P2.13	$(M(NM))^T$	P2.14	$(M(NM))N$	P2.15	$\texttt{sum}(A)$
P2.16	$\texttt{trace}((DC)^{-1}) + traceD)$	P2.17	$((((C+D)^{-1})^T)D$	P2.18	$\texttt{rowSums}(A+B)^T$
P2.25	$u_1 v_2^T v_2 - Xv_2$				

Table 13: $\mathcal{P}^{\neg Opt}$ Pipelines (Part 2) Rewrites

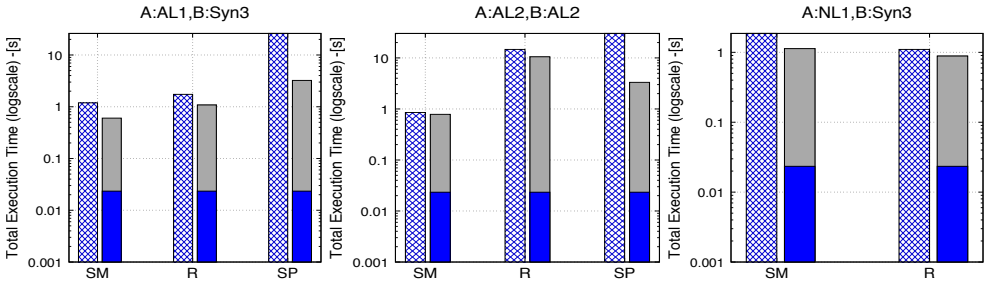
No.	Expression	No.	Expression	No.	Expression
V_1	$(D)^{-1}$	V_2	$(C^T)^{-1}$	V_3	NM
V_4	$u_1 v_2^T$	V_5	DC	V_6	$A+B$
V_7	C^{-1}	V_8	$C^T D$	V_9	$(D+C)^{-1}$
V_{10}	$\texttt{det}(CD)$	V_{11}	$\texttt{det}(DC)$	V_{12}	$(DC)^T$

Table 14: The set of views V_{exp}

No.	Rewrite	No.	Rewrite	No.	Rewrite
P1.2	$(V_6)^T$	P1.3	$V_7 V_1$	P1.4	$(V_6)v_1$
P1.11	$\texttt{colSums}(V_6)^T$	P1.15	$M(V_3)$	P1.17	$V_{10} * \texttt{det}(C)$
P1.19	V_2	P1.20	$\texttt{trace}(V_7)$	P1.21	$(C+V_1)^T$
P1.22	$\texttt{trace}(V_9)$	P1.24	$\texttt{trace}(V_1 V_7) + \texttt{trace}(D)$	P1.29	$V_5 CCC$
P1.30	$V_3 \odot V_3 R^T$	P2.2	$\texttt{det}(V_1)$	P2.4	$s_1(V6)$
P2.5	$\texttt{det}(V_9)$	P2.6	$(V_1 C)^T$	P2.9	$\texttt{trace}(V_{12}) + \texttt{trace}(D)$
P2.11	$\texttt{sum}(V_6)$	P2.13	$(MV_3)^T$	P2.14	$MV_3 N$
P2.16	$\texttt{trace}(V_7 V_1) + traceD)$	P2.17	$(V_9^T)D$	P2.18	$\texttt{rowSums}(V_6)^T$
P2.20	$(MV_3)^T$	P2.21	$V_1(V_1^T(D^T v_1))$	P2.25	$V_4 v_1 - Xv_1$
P1.23	$\texttt{det}((V_7 V_1) + D)$	P2.26	$exp(V_9)$	P2.27	$V_9^T V_5$

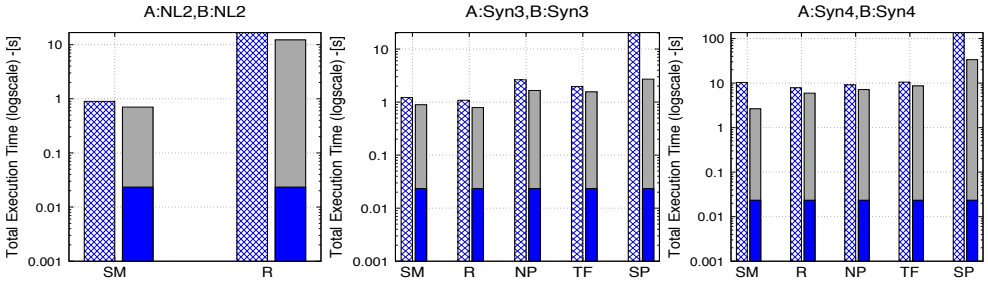
Table 15: $\mathcal{P}^{Views}$ Pipelines Rewrites

D ADDITIONAL RESULTS: $\mathcal{P}^{\neg Opt}$ PIPELINES - NAÏVE-BASED COST MODEL



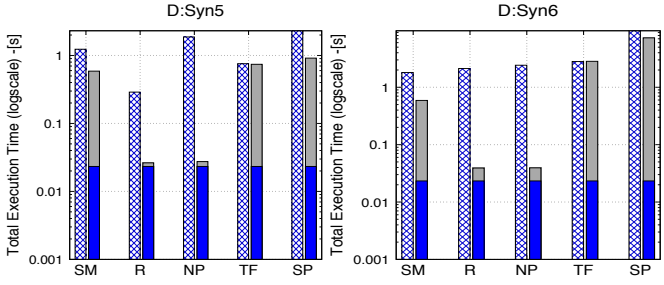
(a) **P1.2** (b) **P1.2** (c) **P1.2**

Figure 13: P1.2 evaluation time with and without rewriting



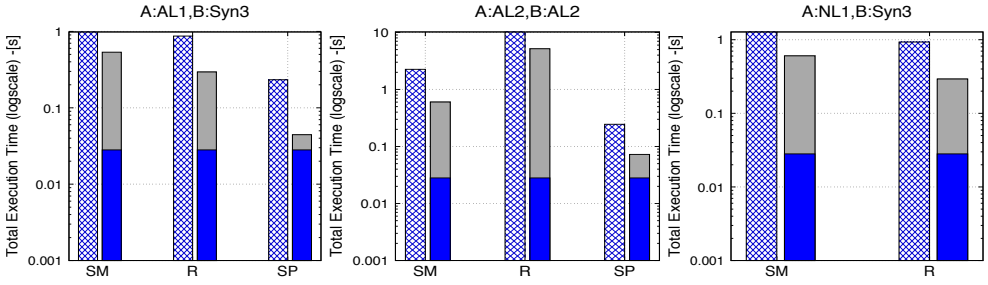
(a) **P1.2** (b) **P1.2** (c) **P1.2**

Figure 14: P1.2 evaluation time with and without rewriting



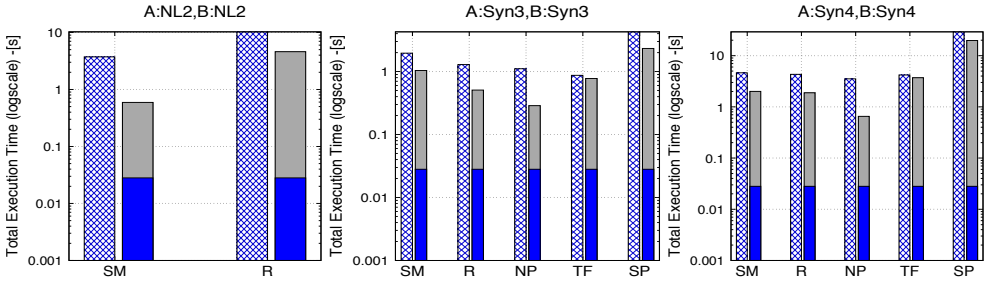
(a) **P1.6** (b) **P1.6**

Figure 15: P1.6 evaluation time with and without rewriting



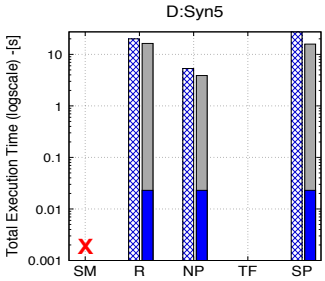
(a) **P1.8** (b) **P1.8** (c) **P1.8**

Figure 16: P1.8 evaluation time with and without rewriting



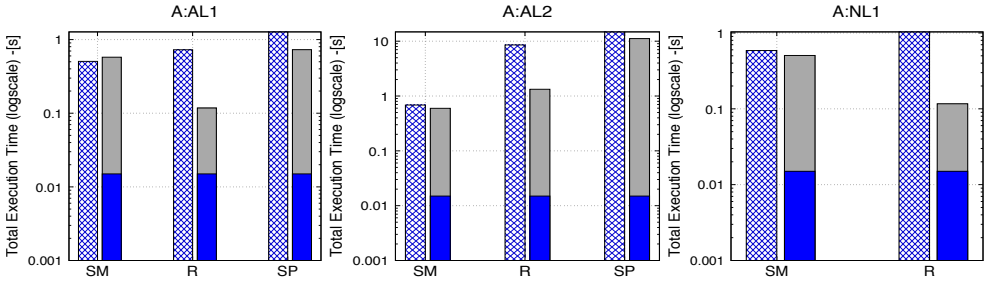
(a) **P1.8** (b) **P1.8** (c) **P1.8**

Figure 17: P1.8 evaluation time with and without rewriting



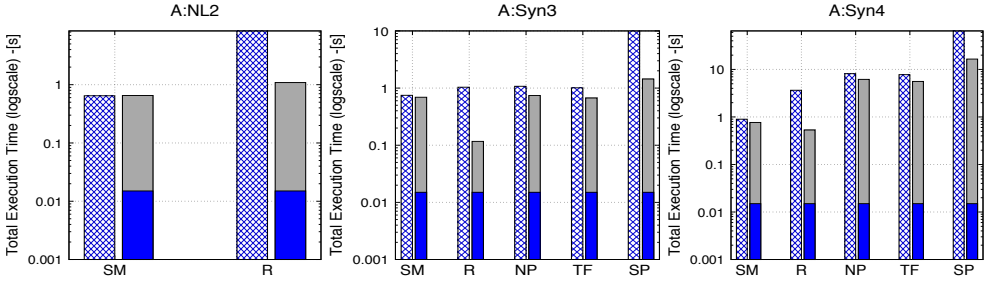
(a) **P1.9**

Figure 18: P1.9 evaluation time with and without rewriting



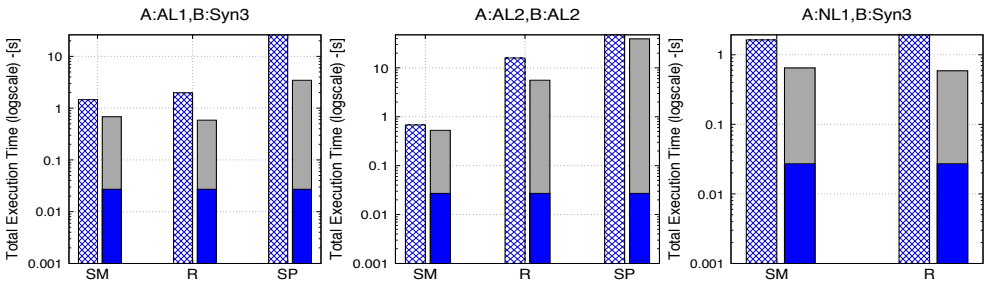
(a) **P1.10** (b) **P1.10** (c) **P1.10**

Figure 19: P1.10 evaluation time with and without rewriting



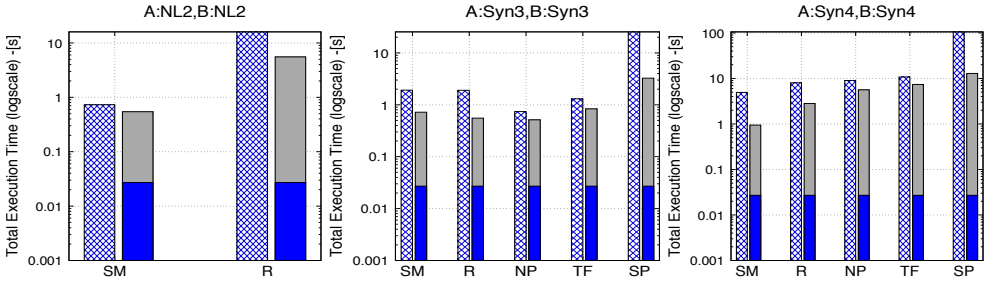
(a) **P1.10** (b) **P1.10** (c) **P1.10**

Figure 20: P1.10 evaluation time with and without rewriting



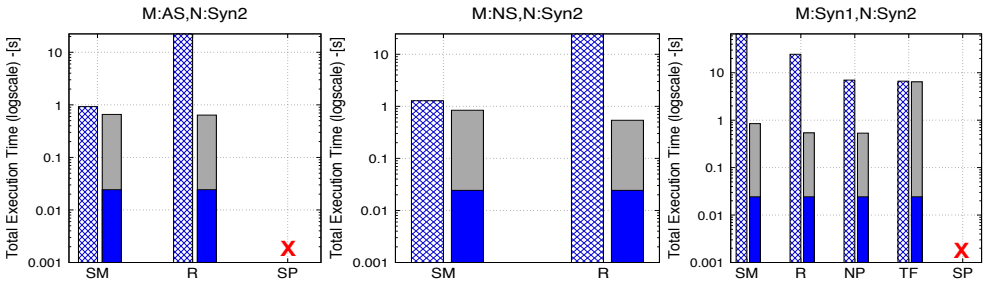
(a) **P1.11** (b) **P1.11** (c) **P1.11**

Figure 21: P1.11 evaluation time with and without rewriting



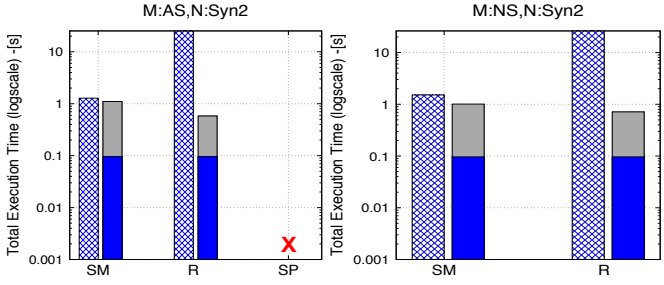
(a) **P1.11** (b) **P1.11** (c) **P1.11**

Figure 22: P1.11 evaluation time with and without rewriting



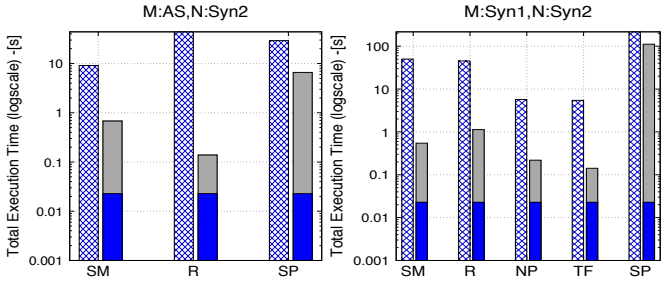
(a) **P1.12** (b) **P1.12** (c) **P1.12**

Figure 23: P1.12 evaluation time with and without rewriting



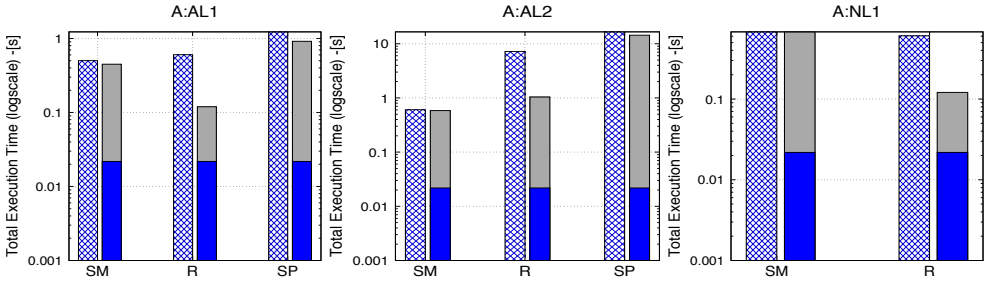
(a) **P1.14** (b) **P1.14**

Figure 24: P1.14 evaluation time with and without rewriting



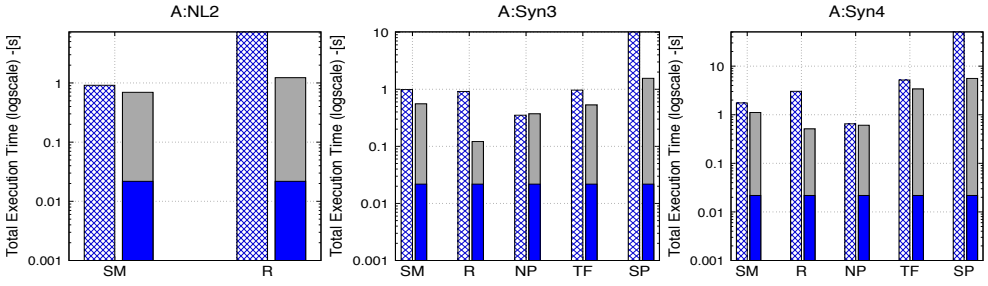
(a) **P1.15** (b) **P1.15**

Figure 25: P1.15 evaluation time with and without rewriting



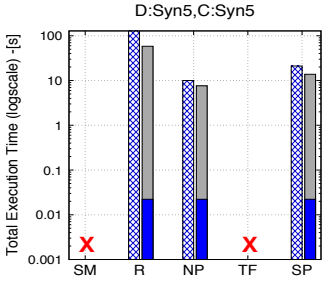
(a) **P1.16** (b) **P1.16** (c) **P1.16**

Figure 26: P1.16 evaluation time with and without rewriting



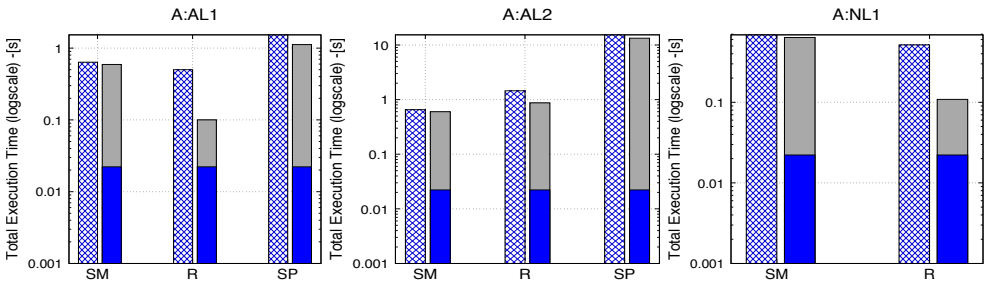
(a) **P1.16** (b) **P1.16** (c) **P1.16**

Figure 27: P1.16 evaluation time with and without rewriting



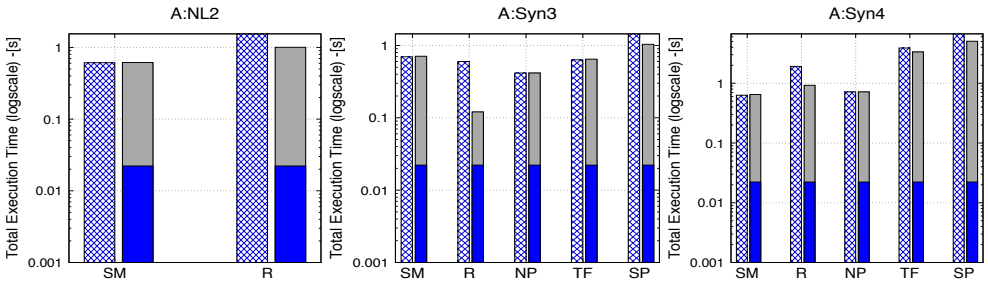
(a) **P1.17**

Figure 28: P1.17 evaluation time with and without rewriting



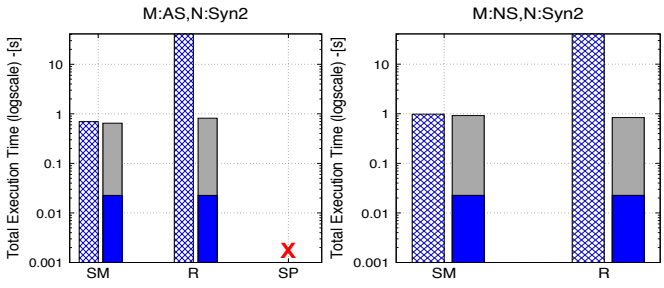
(a) **P1.18** (b) **P1.18** (c) **P1.18**

Figure 29: P1.18 evaluation time with and without rewriting



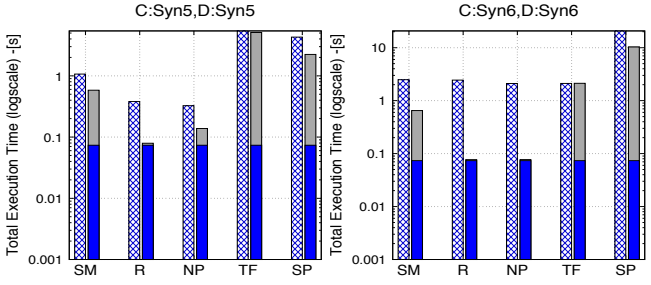
(a) **P1.18** (b) **P1.18** (c) **P1.18**

Figure 30: P1.18 evaluation time with and without rewriting



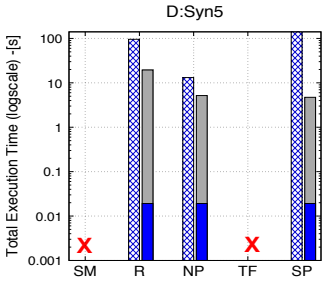
(a) **P1.25** (b) **P1.25**

Figure 31: P1.25 evaluation time with and without rewriting



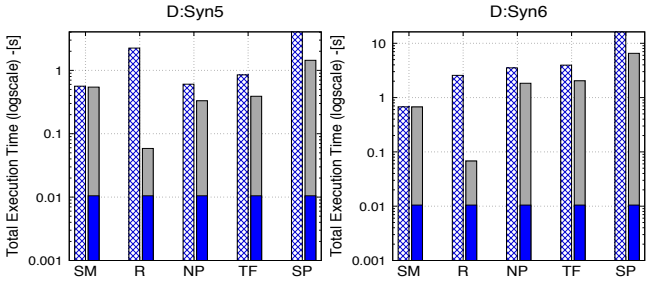
(a) **P2.1** (b) **P2.1**

Figure 32: P2.1 evaluation time with and without rewriting



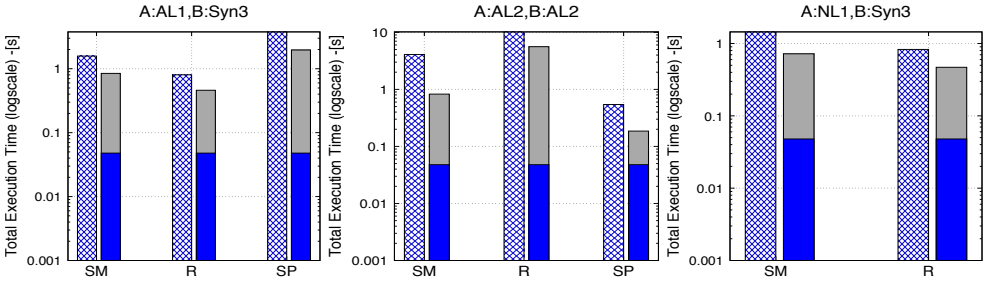
(a) **P2.2**

Figure 33: P2.2 evaluation time with and without rewriting



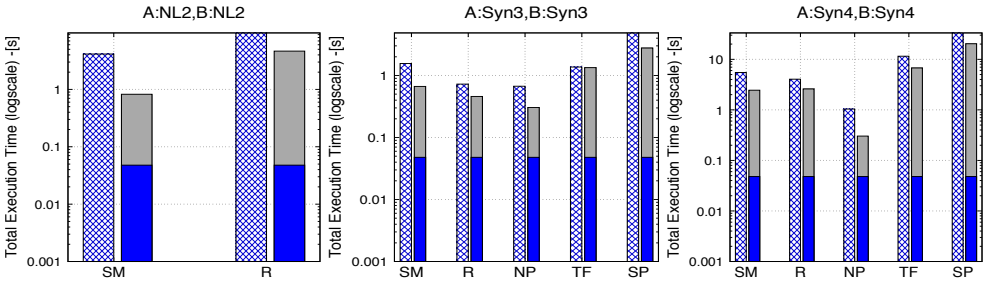
(a) **P2.3** (b) **P2.3**

Figure 34: P2.3 evaluation time with and without rewriting



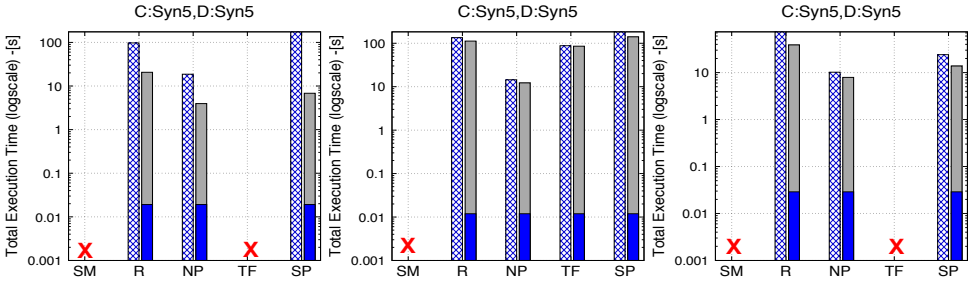
(a) **P2.4** (b) **P2.4** (c) **P2.4**

Figure 35: P2.4 evaluation time with and without rewriting



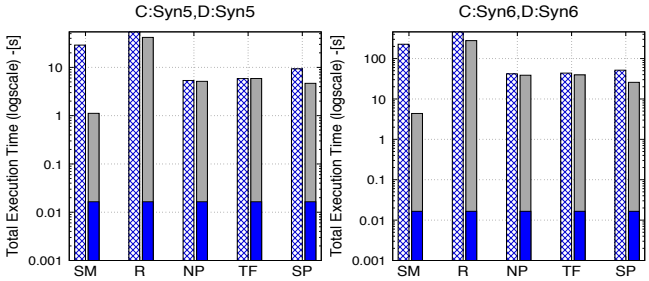
(a) **P2.4** (b) **P2.4** (c) **P2.4**

Figure 36: P2.4 evaluation time with and without rewriting



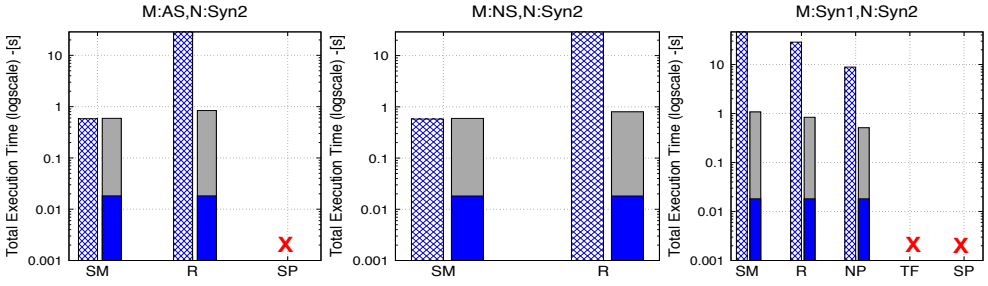
(a) **P2.5** (b) **P2.6** (c) **P2.8**

Figure 37: P2.5, P2.6 and P2.8 evaluation time with and without rewriting



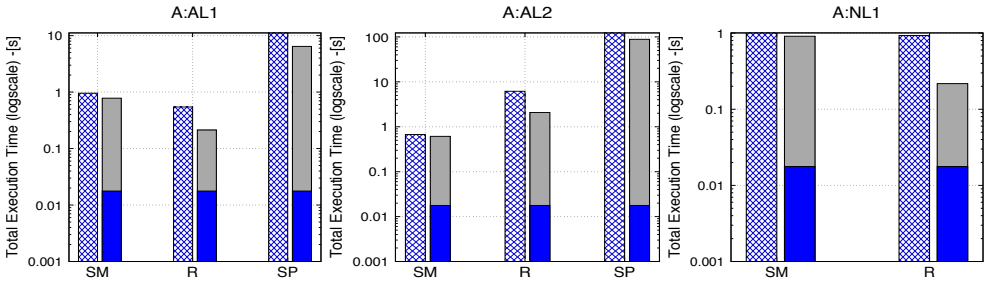
(a) **P2.9** (b) **P2.9S**

Figure 38: P2.9 evaluation time with and without rewriting



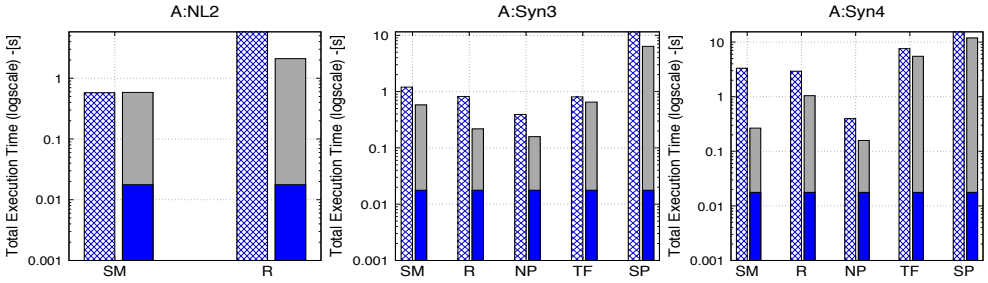
(a) **P2.10** (b) **P2.10** (c) **P2.10**

Figure 39: P2.10 evaluation time with and without rewriting



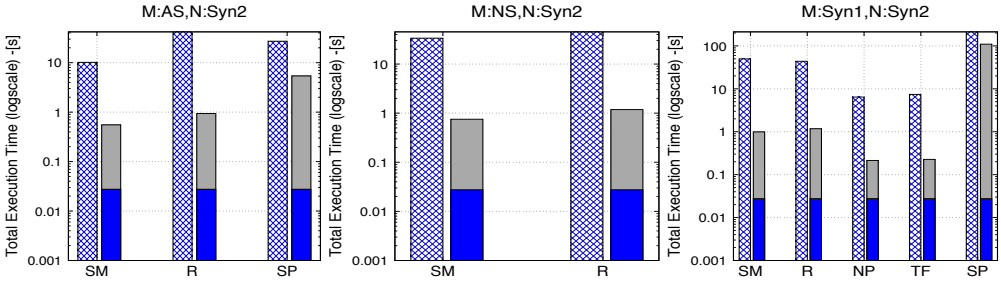
(a) **P2.11** (b) **P2.11** (c) **P2.11**

Figure 40: P2.11 evaluation time with and without rewriting



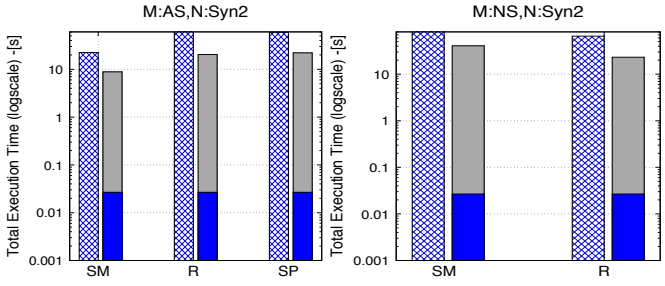
(a) **P2.11** (b) **P2.11** (c) **P2.11**

Figure 41: P2.11 evaluation time with and without rewriting



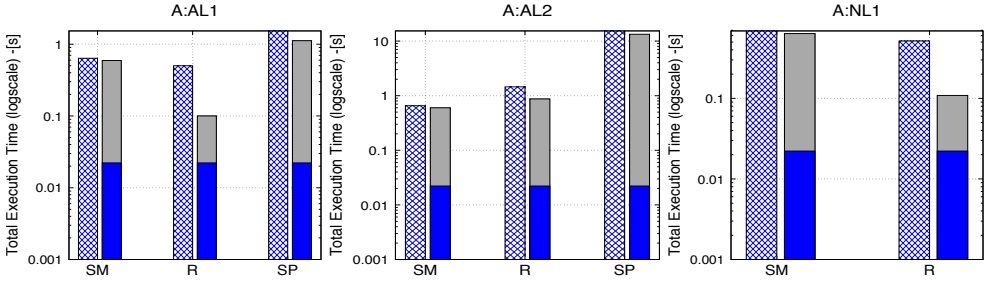
(a) **P2.13** (b) **P2.13** (c) **P2.13**

Figure 42: P2.13 evaluation time with and without rewriting



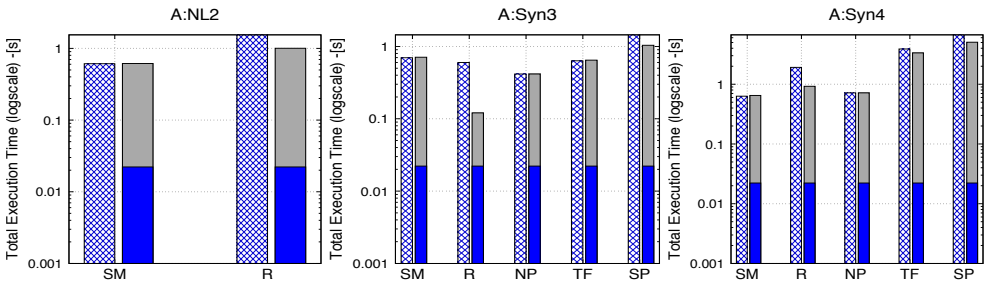
(a) **P2.14** (b) **P2.14**

Figure 43: P2.14 evaluation time with and without rewriting



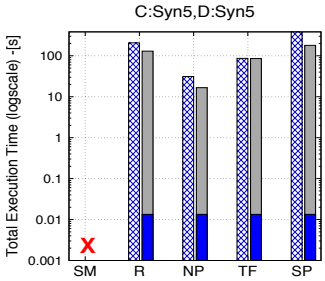
(a) **P2.15** (b) **P2.15** (c) **P2.15**

Figure 44: P2.15 evaluation time with and without rewriting



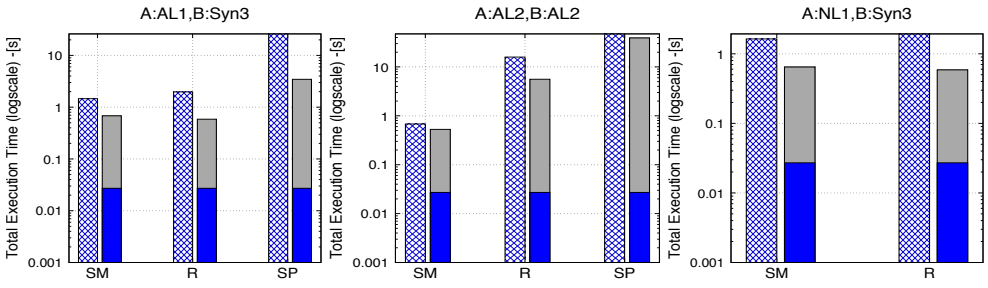
(a) **P2.15** (b) **P2.15** (c) **P2.15**

Figure 45: P2.15 evaluation time with and without rewriting



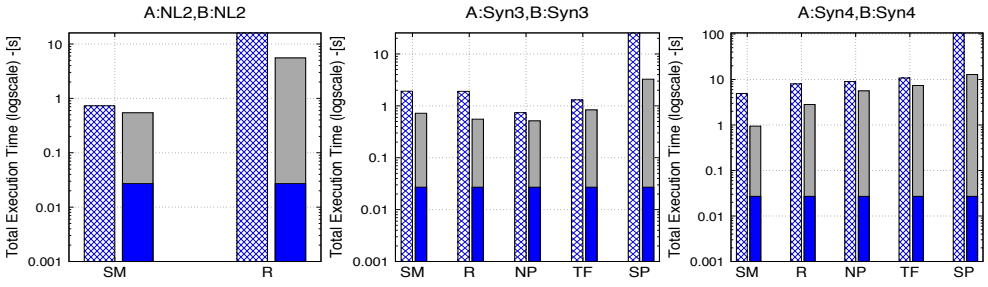
(a) **P2.16**

Figure 46: P2.16 evaluation time with and without rewriting



(a) **P2.18** (b) **P2.18** (c) **P2.18**

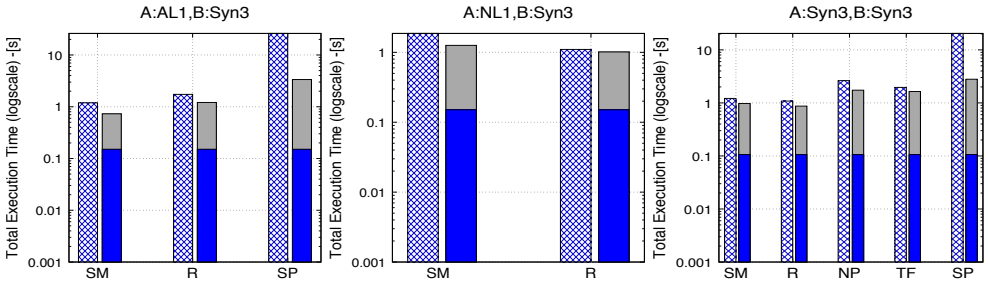
Figure 47: P2.18 evaluation time with and without rewriting



(a) **P2.18** (b) **P2.18** (c) **P2.18**

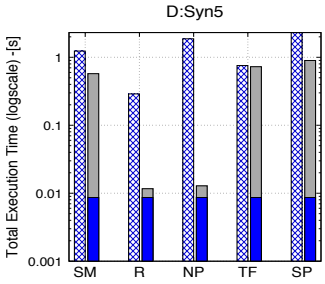
Figure 48: P2.18 evaluation time with and without rewriting

E ADDITIONAL RESULTS: $\mathcal{P}^{\neg Opt}$ PIPELINES - MNC-BASED COST MODEL



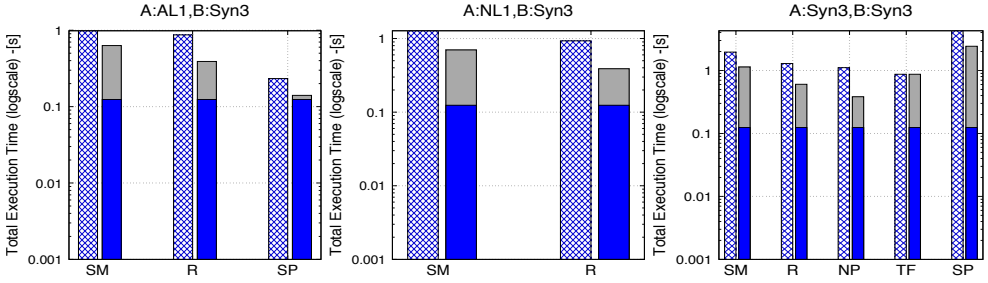
(a) **P1.2** (b) **P1.2** (c) **P1.2**

Figure 49: P1.2 evaluation time with and without rewriting



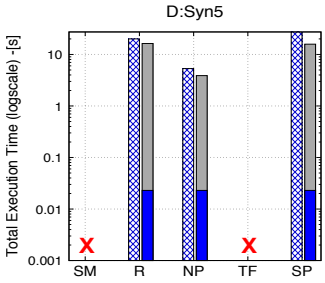
(a) **P1.6**

Figure 50: P1.6 evaluation time with and without rewriting



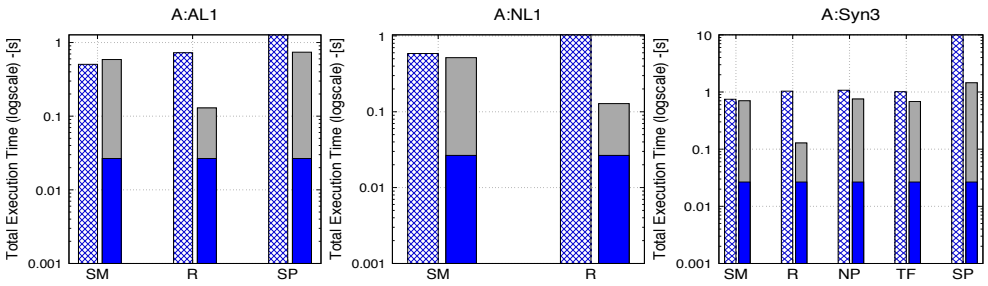
(a) **P1.8** (b) **P1.8** (c) **P1.8**

Figure 51: P1.8 evaluation time with and without rewriting



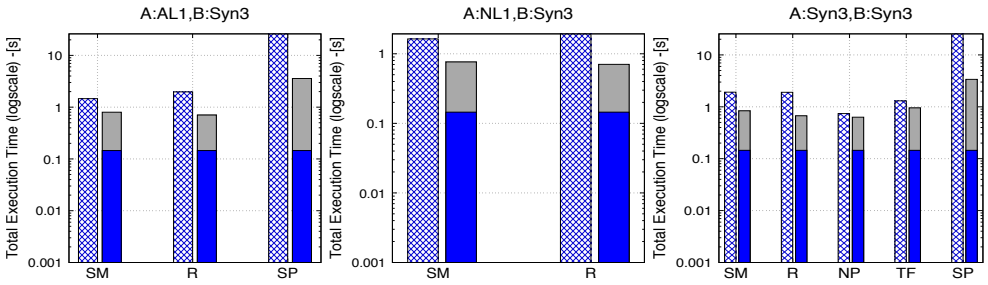
(a) **P1.9**

Figure 52: P1.9 evaluation time with and without rewriting



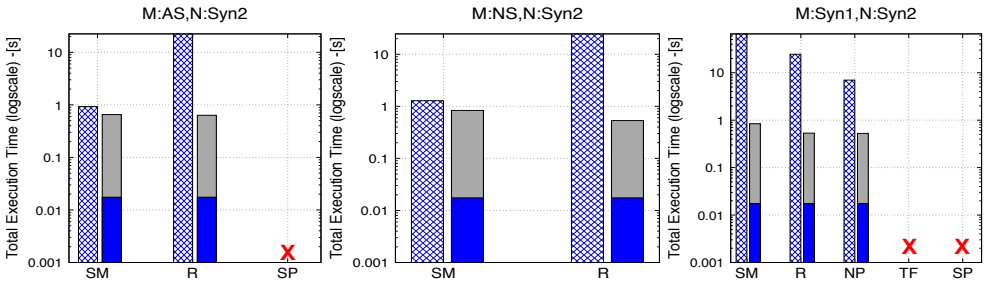
(a) **P1.10** (b) **P1.10** (c) **P1.10**

Figure 53: P1.10 evaluation time with and without rewriting



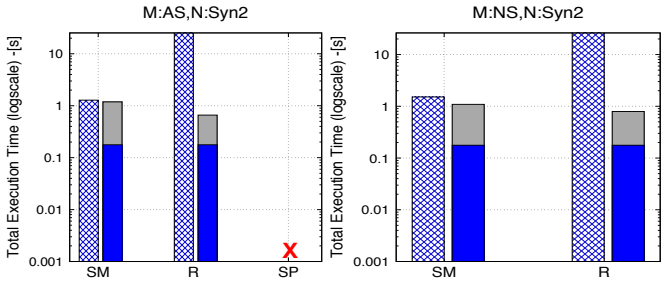
(a) **P1.11** (b) **P1.11** (c) **P1.11**

Figure 54: P1.11 evaluation time with and without rewriting



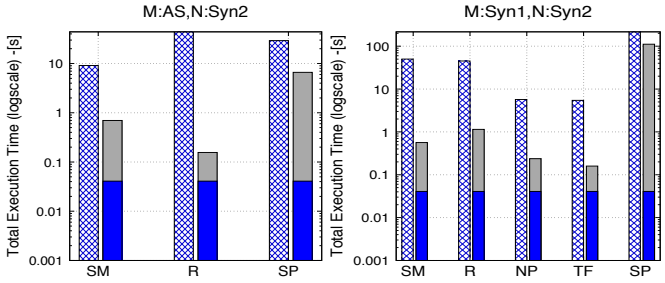
(a) **P1.12** (b) **P1.12** (c) **P1.12**

Figure 55: P1.12 evaluation time with and without rewriting



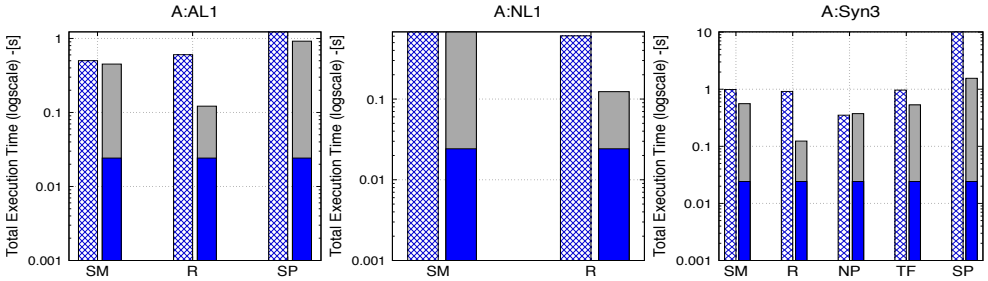
(a) **P1.14** (b) **P1.14**

Figure 56: P1.14 evaluation time with and without rewriting



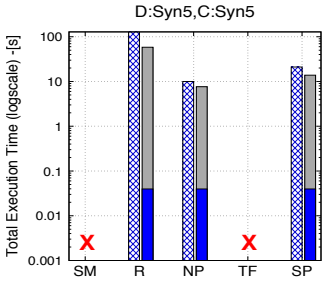
(a) **P1.15** (b) **P1.15**

Figure 57: P1.15 evaluation time with and without rewriting



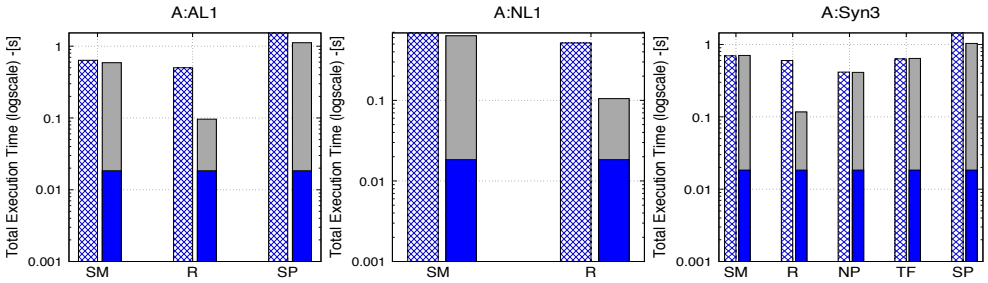
(a) **P1.16** (b) **P1.16** (c) **P1.16**

Figure 58: P1.16 evaluation time with and without rewriting



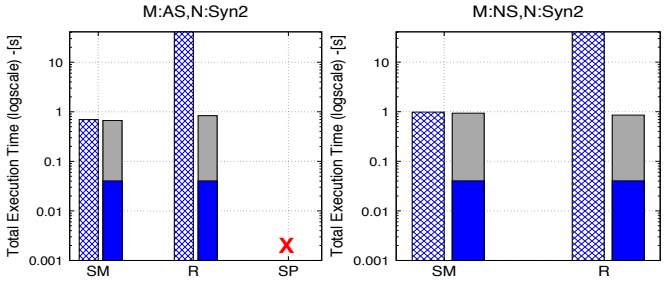
(a) **P1.17**

Figure 59: P1.17 evaluation time with and without rewriting



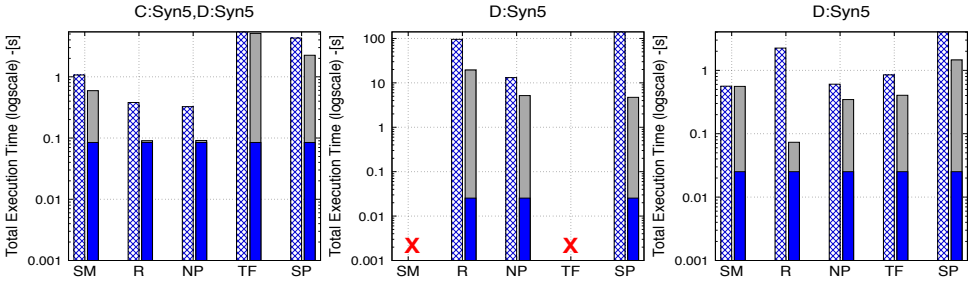
(a) **P1.18** (b) **P1.18** (c) **P1.18**

Figure 60: P1.18 evaluation time with and without rewriting



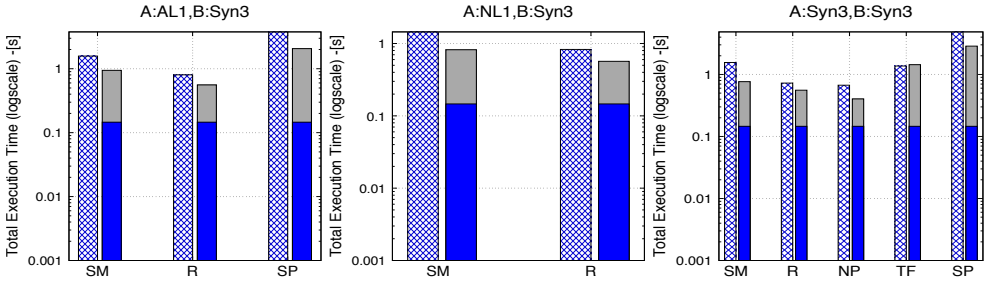
(a) **P1.25** (b) **P1.25**

Figure 61: P1.25 evaluation time with and without rewriting



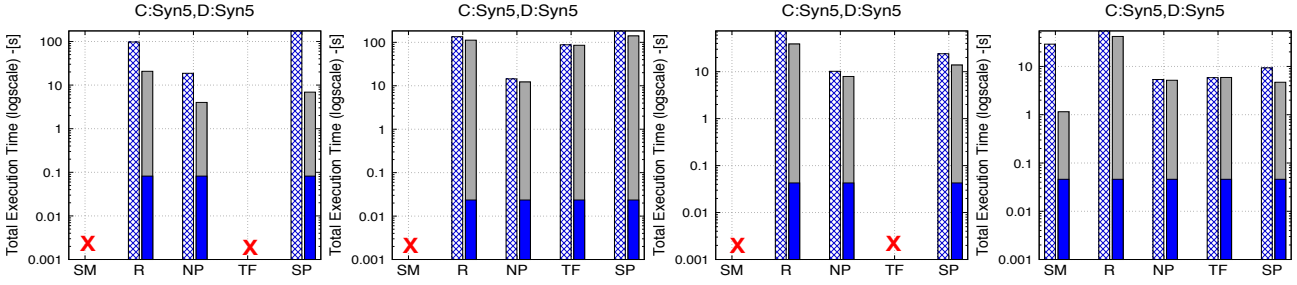
(a) **P2.1** (b) **P2.2** (c) **P2.3**

Figure 62: P2.1 evaluation time with and without rewriting



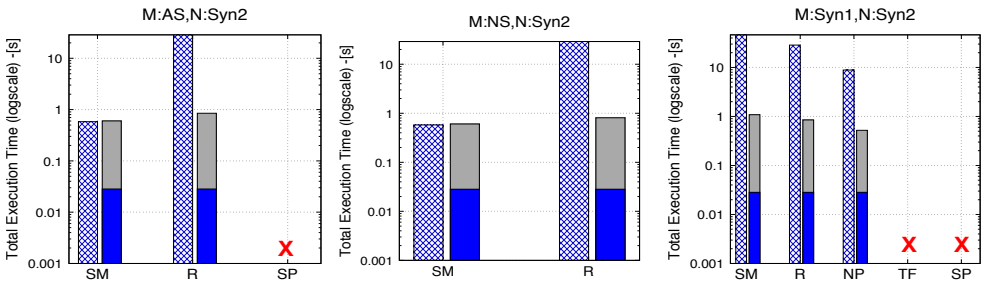
(a) **P2.4** (b) **P2.4** (c) **P2.4**

Figure 63: P2.4 evaluation time with and without rewriting



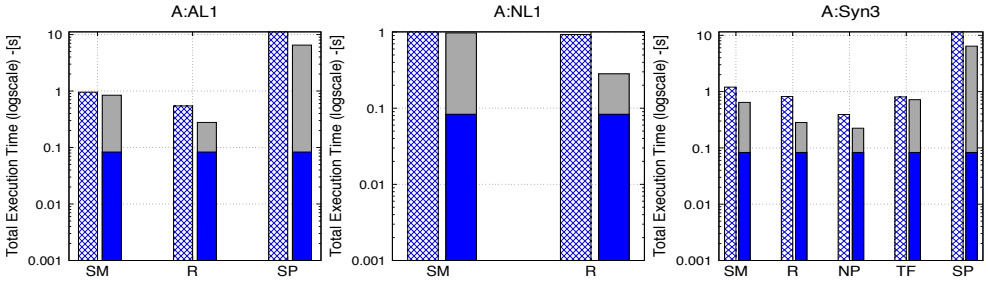
(a) **P2.5** (b) **P2.6** (c) **P2.8** (d) **P2.9**

Figure 64: P2.5, P2.6, P2.8 and P2.9evaluation time with and without rewriting



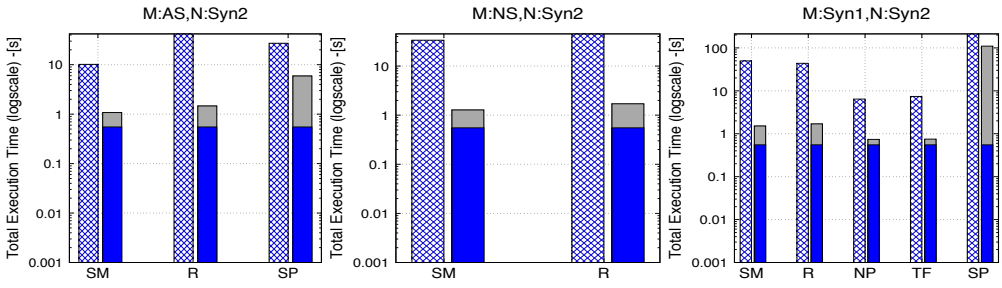
(a) **P2.10** (b) **P2.10** (c) **P2.10**

Figure 65: P2.10 evaluation time with and without rewriting



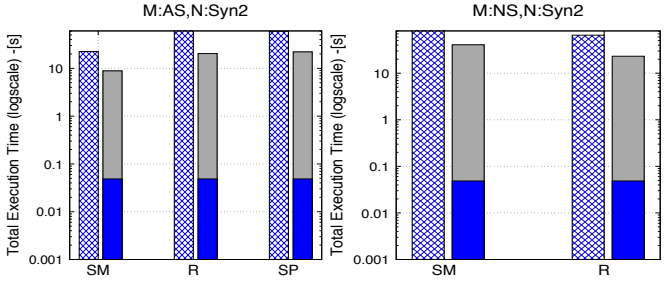
(a) **P2.11** (b) **P2.11** (c) **P2.11**

Figure 66: P2.11 evaluation time with and without rewriting



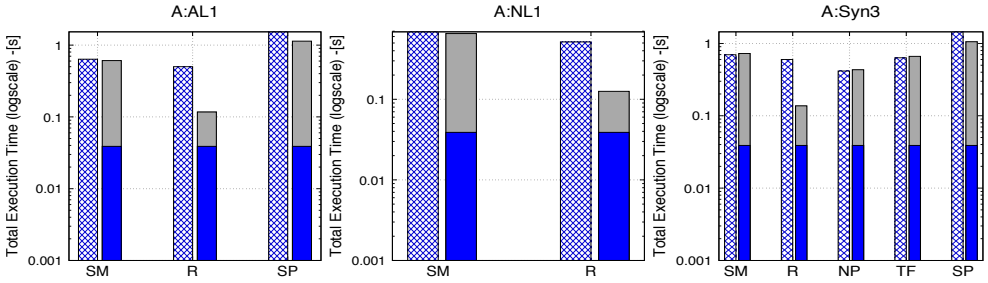
(a) **P2.13** (b) **P2.13** (c) **P2.13**

Figure 67: P2.13 evaluation time with and without rewriting



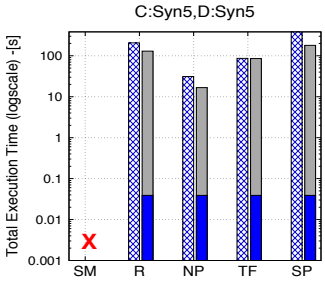
(a) **P2.14** (b) **P2.14**

Figure 68: P2.14 evaluation time with and without rewriting



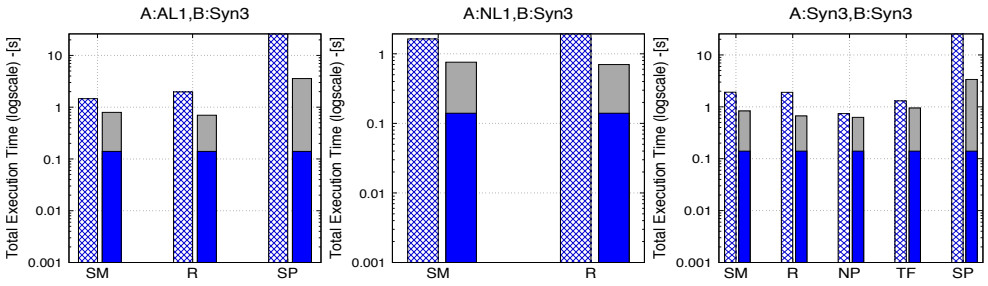
(a) **P2.15** (b) **P2.15** (c) **P2.15**

Figure 69: P2.15 evaluation time with and without rewriting



(a) **P2.16**

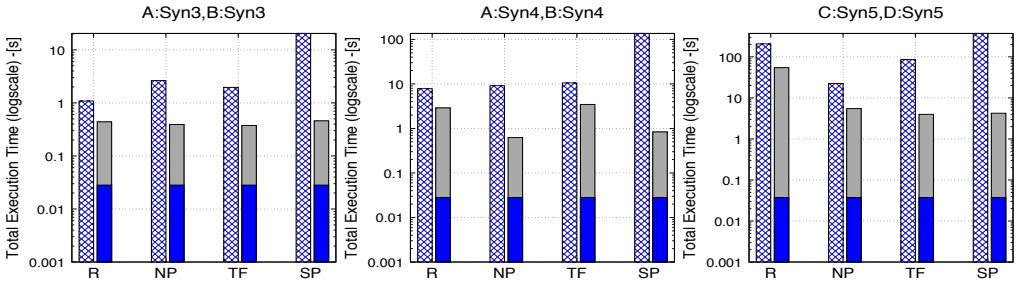
Figure 70: P2.16 evaluation time with and without rewriting



(a) **P2.18** (b) **P2.18** (c) **P2.18**

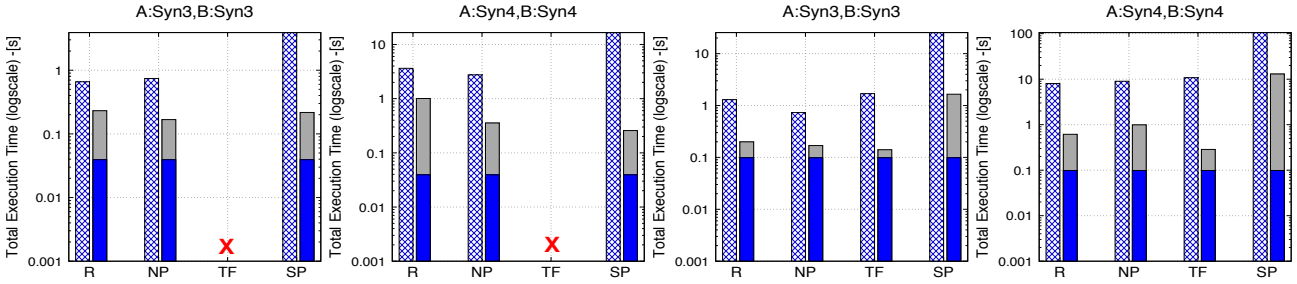
Figure 71: P2.18 evaluation time with and without rewriting

F ADDITIONAL RESULTS: $\mathcal{P}^{Views}$ PIPELINES



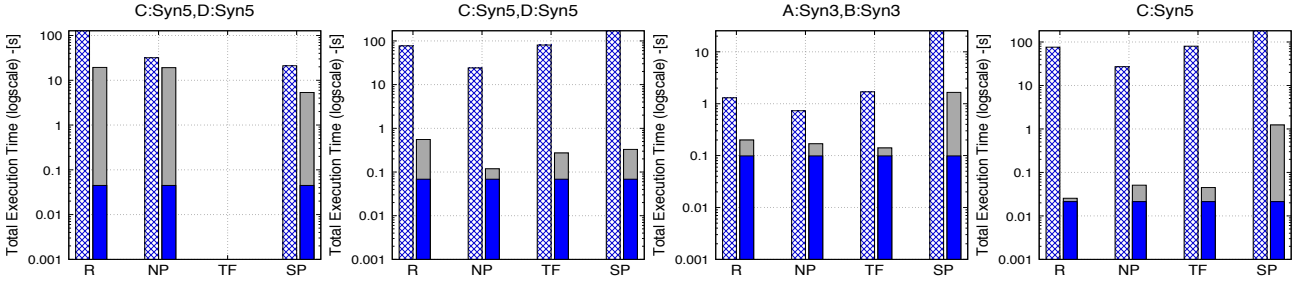
(a) **P1.2** (b) **P1.2** (c) **P1.3**

Figure 72: P1.2 and P1.3 evaluation time with and without rewriting



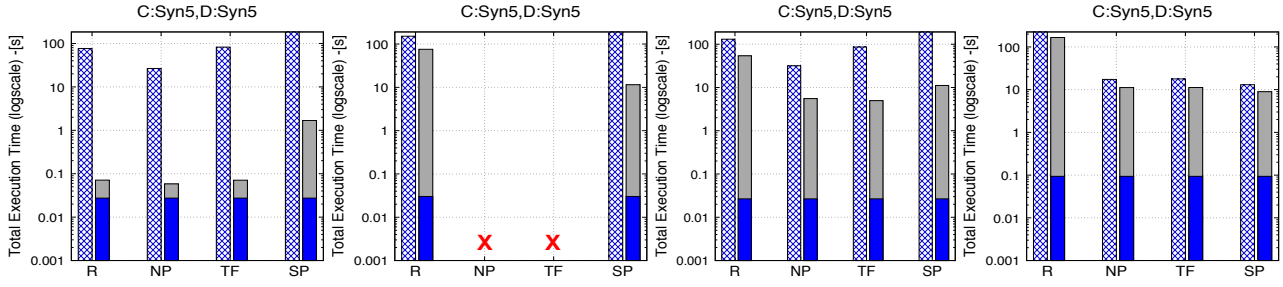
(a) **P1.4** (b) **P1.4** (c) **P1.11** (d) **P1.11**

Figure 73: P1.4 and P1.11 evaluation time with and without rewriting



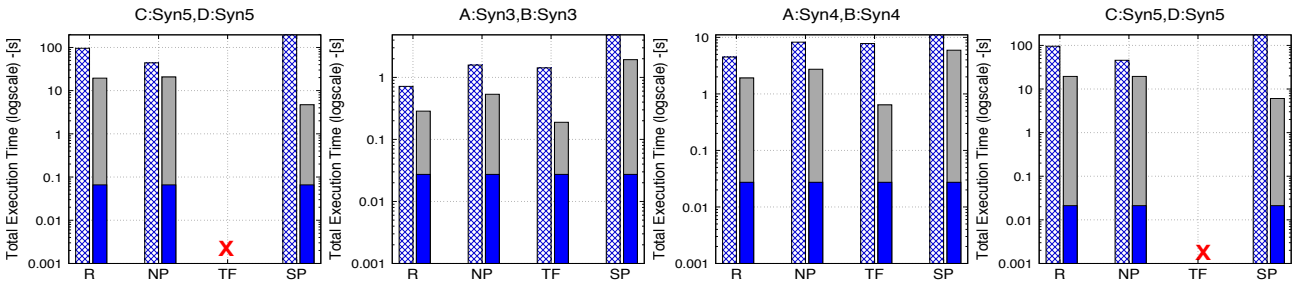
(a) **P1.17** (b) **P1.19** (c) **P1.20** (d) **P1.21**

Figure 74: P1.17,P1.19,P1.20 and P1.21 evaluation time with and without rewriting



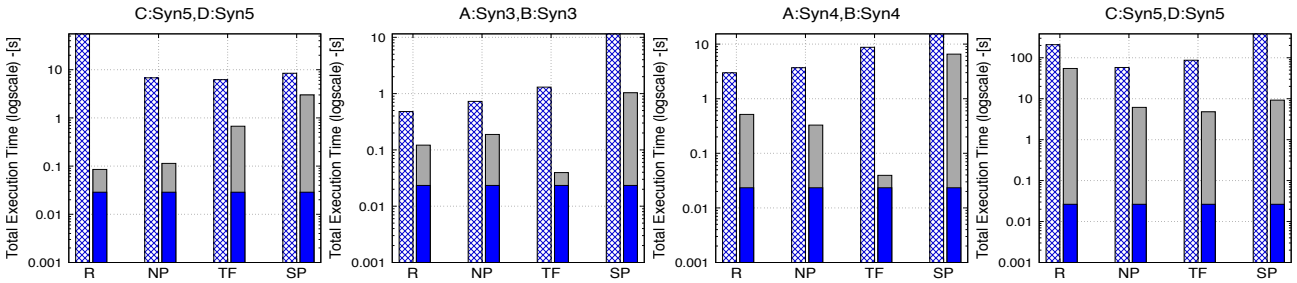
(a) **P1.22** (b) **P1.23** (c) **P1.24** (d) **P1.29**

Figure 75: P1.22,P1.23,P1.24 and P1.29 evaluation time with and without rewriting



(a) **P2.2** (b) **P2.4** (c) **P2.4** (d) **P2.5**

Figure 76: P2.2,P2.4 and P2.5 evaluation time with and without rewriting



(a) **P2.9** (b) **P2.11** (c) **P2.11** (d) **P2.16**

Figure 77: P2.9,P2.11 and P2.16 evaluation time with and without rewriting

G HYBRID TWITTER BENCHMARK CONSTRAINTS

```
#EGDs
name(id1, n1), name(id2, n1)− > id2 = id1;
tr(id1, idt1), tr(id1, idt2)− > idt2 = idt1;
in(id1, idt1), in(id1, idt2)− > idt2 = idt1;
ins(id1, idt1), ins(id1, idt2)− > idt1 = idt2;
in(id1, idr1), in(idr1, idr3)− > idr3 = id1;
multi(idA, idB, idr1), multi(idA, idB, idr2)− > idr2 = idr1;
multi_S(idA, idB, idr1), multi_S(idA, idB, idr2)− > idr2 = idr1;
det(idA, d1), det(idA, d2)− > d2 = d1;
trace(idA, id1), trace(idA, id2)− > id2 = id1;
add(idA, idB, idr1), add(idA, idB, idr2)− > idr1 = idr2;
size(idA, i1, j1), size(idA, i2, j2)− > j2 = j1, i2 = i1;
name(id1, n1), name(id2, n1)− > id2 = id1;
nameI(mID1, n1), nameI(mID1, n2)− > n1 = n2;
add_S(idA, idB, idr1), add_S(idA, idB, idr2)− > idr1 = idr2;
sum(id1, idt1), sum(id1, idt2)− > idt2 = idt1;
rowSums(idA, id1), rowSums(idA, id2)− > id2 = id1;
colSums(idA, id1), colSums(idA, id2)− > id2 = id1;
user(EID, FOLC1, FRIC1, LC1, P1, V1), user(EID, FOLC2, FRIC2, LC2, P2, V2)− > FOLC2 = FOLC1, FRIC2 = FRIC1, LC2 = LC1, P2 = P1, V2 = V1;
constructMatrix_2(rID, mID, v11, v12), constructMatrix_2(rID, mID, v21, v22)− > v21 = v11, v22 = v12;
constructMatrix_4(rID, mID, v11, v12, v13, v14), constructMatrix_4(rID, mID, v21, v22, v23, v24)− > v21 = v11, v22 = v12, v23 = v13, v24 = v14;
constructMatrix_11(rID, mID, v11, v12, v13, v14, v15, v16, v17, v18, v19, v110, v111), constructMatrix_11(rID, mID, v21, v22, v23, v24, v25, v26, v27, v28, v29, v210, v211)− >
v11 = v21, v12 = v22, v13 = vv23, v14 = v24, v15 = v25, v16 = v26, v17 = v27, v18 = v28, v19 = v29, v110 = v210, v111 = v211;
constructMtxMatrix(i, j, v, id)(i1, j1, v1, id), constructMtxMatrix(i2, j2, v2, id)− > i2 = i1, j2 = j1, v2 = v1;
indicator(id1, id2, idr31), indicator(id1, id2, idr32)− > idr32 = idr31;
deriveFrom(id1, id2, idr31), deriveFrom(id1, id2, idr32)− > idr32 = idr31;
concat(id1, id2, idr1), concat(id1, id2, idr2)− > idr2 = idr1;
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr1),
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr2)− > idr2 = idr1;
subset_user(id1, id2, id3, id4, id5, id6, idr1),
subset_user(id1, id2, id3, id4, id5, id6, idr2)− > idr2 = idr1;
tweet(id1), tweet(id2)− > id2 = id1;
child_sp(x, y1, z, d), child_sp(x, y2, z, d)− > y2 = y1;
val_sp(x, v1), val_sp(x, v2)− > v2 = v1;
child_sjV2(x, y1, z, d), child_sjV2(x, y2, z, d)− > y2 = y1;

#TGDs
add(idA, idB, idr1)− > add(idB, idA, idr1);
multi(id1, id2, idr1), rowSums(idr1, idr2)− > rowSums(id2, idr3), multi(id1, idr3, idr2);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);
multi(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idB, idC, idr3), multi(idA, idr3, idr2);
add(idA, idB, idr1), tr(idr1, idT)− > tr(idA, idAT), tr(idB, idBT), add(idAT, idBT, idT);
tr(idA, idAT), tr(idB, idBT), multi(idBT, idAT, idr2)− > multi(idA, idB, idr1), tr(idr1, idr2);
construcMtxMatrix(i, j, v, id), name(id, n), matrixFilterLT(id, x, idr1)− >
name(idr1, n), construcMtxMatrix(i, j, v, idr1), LT(v, x);

subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idK), deriveFrom(idS1, idR1, id), rowSums(id, idr1)− >
rowSums(idS1, idr2), rowSums(idR1, idr3), multi(idK, idr3, idr4), add(idr2, idr4, idr1);

user(EID, FOLC, FRIC, LC, P, V),
tweetR(EID, FC, QC, RC, F, PS, R),
constructMatrix_11(mID1, rwID, FOLC, FRIC, LC, P, V, FC, QC, RC, F, PS, R)− >
nameI(mID1, n),
name(mID2, "tweet"),
name(mID3, "user"),
subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idr3),
deriveFrom(idS1, idR1, mID1);
name(idS, "tweet"),
name(idR, "user"),
indicator(idS, idR, idK),
subset_tweet(idS, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(idR, "FOLC", "FRIC", "LC", "P", "V", idR1),
rowSums(idS1, idr1),
rowSums(idR1, idr2),
multi(idK, idr2, idr3),
add(idr1, idr3, idr4)− > name(idr4, "VV1");

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "flevel", "o", fID),
child_sp(id, "entities", "o", eID),
child_sp(eID, "hashtag", "o", hID),
child_sp(hID, "hID", "o", hcID),
child_sp(id, "place", "o", pID),
child_sp(pID, "code", "o", cID)− > V1(tID, hcID, fID, cID);

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "text", "o", text)− >
V2(idV2),
child_sjV2(idV2, "tweetID", "o", tID),
child_sjV2(idV2, "text", "o", text);
```

Listing G1: Forward Constraints Used for Q1

```
#EGDs
name(id1, n1), name(id2, n1)− > id2 = id1;
tr(id1, idt1), tr(id1, idt2)− > idt2 = idt1;
in(id1, idt1), in(id1, idt2)− > idt2 = idt1;
ins(id1, idt1), ins(id1, idt2)− > idt1 = idt2;
in(id1, idr1), in(idr1, idr3)− > idr3 = id1;
multi(idA, idB, idr1), multi(idA, idB, idr2)− > idr2 = idr1;
multi_S(idA, idB, idr1), multi_S(idA, idB, idr2)− > idr2 = idr1;
det(idA, d1), det(idA, d2)− > d2 = d1;
trace(idA, id1), trace(idA, id2)− > id2 = id1;
add(idA, idB, idr1), add(idA, idB, idr2)− > idr1 = idr2;
size(idA, i1, j1), size(idA, i2, j2)− > j2 = j1, i2 = i1;
name(id1, n1), name(id2, n1)− > id2 = id1;
nameI(mID1, n1), nameI(mID1, n2)− > n1 = n2;
add_S(idA, idB, idr1), add_S(idA, idB, idr2)− > idr1 = idr2;
sum(id1, idt1), sum(id1, idt2)− > idt2 = idt1;
rowSums(idA, id1), rowSums(idA, id2)− > id2 = id1;
colSums(idA, id1), colSums(idA, id2)− > id2 = id1;
user(EID, FOLC1, FRIC1, LC1, P1, V1), user(EID, FOLC2, FRIC2, LC2, P2, V2)− > FOLC2 = FOLC1, FRIC2 = FRIC1, LC2 = LC1, P2 = P1, V2 = V1;
constructMatrix_2(rID, mID, v11, v12), constructMatrix_2(rID, mID, v21, v22)− > v21 = v11, v22 = v12;
constructMatrix_4(rID, mID, v11, v12, v13, v14), constructMatrix_4(rID, mID, v21, v22, v23, v24)− > v21 = v11, v22 = v12, v23 = v13, v24 = v14;
constructMatrix_11(rID, mID, v11, v12, v13, v14, v15, v16, v17, v18, v19, v110, v111), constructMatrix_11(rID, mID, v21, v22, v23, v24, v25, v26, v27, v28, v29, v210, v211)− >
v11 = v21, v12 = v22, v13 = vv23, v14 = v24, v15 = v25, v16 = v26, v17 = v27, v18 = v28, v19 = v29, v110 = v210, v111 = v211;
constructMtxMatrix(i, j, v, id)(i1, j1, v1, id), constructMtxMatrix(i2, j2, v2, id)− > i2 = i1, j2 = j1, v2 = v1;
indicator(id1, id2, idr31), indicator(id1, id2, idr32)− > idr32 = idr31;
deriveFrom(id1, id2, idr31), deriveFrom(id1, id2, idr32)− > idr32 = idr31;
concat(id1, id2, idr1), concat(id1, id2, idr2)− > idr2 = idr1;
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr1),
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr2)− > idr2 = idr1;
subset_user(id1, id2, id3, id4, id5, id6, idr1),
subset_user(id1, id2, id3, id4, id5, id6, idr2)− > idr2 = idr1;
tweet(id1), tweet(id2)− > id2 = id1;
child_sp(x, y1, z, d), child_sp(x, y2, z, d)− > y2 = y1;
val_sp(x, v1), val_sp(x, v2)− > v2 = v1;
child_sj_V2(x, y1, z, d), child_sj_V2(x, y2, z, d)− > y2 = y1;

#TGDs
add(idA, idB, idr1)− > add(idB, idA, idr1);
multi(id1, id2, idr1), rowSums(idr1, idr2)− > rowSums(id2, idr3), multi(id1, idr3, idr2);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);
multi(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idB, idC, idr3), multi(idA, idr3, idr2);
add(idA, idB, idr1), tr(idr1, idT)− > tr(idA, idAT), tr(idB, idBT), add(idAT, idBT, idT);
tr(idA, idAT), tr(idB, idBT), multi(idBT, idAT, idr2)− > multi(idA, idB, idr1), tr(idr1, idr2);
construcMtxMatrix(i, j, v, id), name(id, n), matrixFilterLT(id, x, idr1)− >
name(idr1, n), construcMtxMatrix(i, j, v, idr1), LT(v, x);
tr(id, idr1), colSums(idr1, idr2)− > rowSums(id, idr3), tr(idr3, idr2);

subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idK), deriveFrom(idS1, idR1, id), rowSums(id, idr1)− >
rowSums(idS1, idr2), rowSums(idR1, idr3), multi(idK, idr3, idr4), add(idr2, idr4, idr1);

user(EID, FOLC, FRIC, LC, P, V),
tweetR(EID, FC, QC, RC, F, PS, R),
constructMatrix_11(mID1, rwID, FOLC, FRIC, LC, P, V, FC, QC, RC, F, PS, R)− >
nameI(mID1, n),
name(mID2, "tweet"),
name(mID3, "user"),
subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idr3),
deriveFrom(idS1, idR1, mID1);
name(idS, "tweet"),
name(idR, "user"),
indicator(idS, idR, idK),
subset_tweet(idS, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(idR, "FOLC", "FRIC", "LC", "P", "V", idR1),
rowSums(idS1, idr1),
rowSums(idR1, idr2),
multi(idK, idr2, idr3),
add(idr1, idr3, idr4)− > name(idr4, "VV1");

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "flevel", "o", fID),
child_sp(id, "entities", "o", eID),
child_sp(eID, "hashtag", "o", hID),
child_sp(hID, "hID", "o", hcID),
child_sp(id, "place", "o", pID),
child_sp(pID, "code", "o", cID)− > V1(tID, hcID, fID, cID);

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "text", "o", text)− >
V2(idV2),
child_sj_V2(idV2, "tweetID", "o", tID),
child_sj_V2(idV2, "text", "o", text);
```

Listing G2: Forward Constraints Used for Q2

```
#EGDs
name(id1, n1), name(id2, n1)− > id2 = id1;
tr(id1, idt1), tr(id1, idt2)− > idt2 = idt1;
in(id1, idt1), in(id1, idt2)− > idt2 = idt1;
ins(id1, idt1), ins(id1, idt2)− > idt1 = idt2;
in(id1, idr1), in(idr1, idr3)− > idr3 = id1;
multi(idA, idB, idr1), multi(idA, idB, idr2)− > idr2 = idr1;
multi_S(idA, idB, idr1), multi_S(idA, idB, idr2)− > idr2 = idr1;
det(idA, d1), det(idA, d2)− > d2 = d1;
trace(idA, id1), trace(idA, id2)− > id2 = id1;
add(idA, idB, idr1), add(idA, idB, idr2)− > idr1 = idr2;
size(idA, i1, j1), size(idA, i2, j2)− > j2 = j1, i2 = i1;
name(id1, n1), name(id2, n1)− > id2 = id1;
nameI(mID1, n1), nameI(mID1, n2)− > n1 = n2;
add_S(idA, idB, idr1), add_S(idA, idB, idr2)− > idr1 = idr2;
sum(id1, idt1), sum(id1, idt2)− > idt2 = idt1;
rowSums(idA, id1), rowSums(idA, id2)− > id2 = id1;
colSums(idA, id1), colSums(idA, id2)− > id2 = id1;
user(EID, FOLC1, FRIC1, LC1, P1, V1), user(EID, FOLC2, FRIC2, LC2, P2, V2)− > FOLC2 = FOLC1, FRIC2 = FRIC1, LC2 = LC1, P2 = P1, V2 = V1;
constructMatrix_2(rID, mID, v11, v12), constructMatrix_2(rID, mID, v21, v22)− > v21 = v11, v22 = v12;
constructMatrix_4(rID, mID, v11, v12, v13, v14), constructMatrix_4(rID, mID, v21, v22, v23, v24)− > v21 = v11, v22 = v12, v23 = v13, v24 = v14;
constructMatrix_11(rID, mID, v11, v12, v13, v14, v15, v16, v17, v18, v19, v110, v111), constructMatrix_11(rID, mID, v21, v22, v23, v24, v25, v26, v27, v28, v29, v210, v211)− >
v11 = v21, v12 = v22, v13 = vv23, v14 = v24, v15 = v25, v16 = v26, v17 = v27, v18 = v28, v19 = v29, v110 = v210, v111 = v211;
constructMtxMatrix(i, j, v, id)(i1, j1, v1, id), constructMtxMatrix(i2, j2, v2, id)− > i2 = i1, j2 = j1, v2 = v1;
indicator(id1, id2, idr31), indicator(id1, id2, idr32)− > idr32 = idr31;
deriveFrom(id1, id2, idr31), deriveFrom(id1, id2, idr32)− > idr32 = idr31;
concat(id1, id2, idr1), concat(id1, id2, idr2)− > idr2 = idr1;
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr1),
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr2)− > idr2 = idr1;
subset_user(id1, id2, id3, id4, id5, id6, idr1),
subset_user(id1, id2, id3, id4, id5, id6, idr2)− > idr2 = idr1;
tweet(id1), tweet(id2)− > id2 = id1;
child_sp(x, y1, z, d), child_sp(x, y2, z, d)− > y2 = y1;
val_sp(x, v1), val_sp(x, v2)− > v2 = v1;
child_sj_V2(x, y1, z, d), child_sj_V2(x, y2, z, d)− > y2 = y1;

#TGDs
add(idA, idB, idr1)− > add(idB, idA, idr1);
multi(id1, id2, idr1), rowSums(idr1, idr2)− > rowSums(id2, idr3), multi(id1, idr3, idr2);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);
multi(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idB, idC, idr3), multi(idA, idr3, idr2);
construcMtxMatrix(i, j, v, id), name(id, n), matrixFilterLT(id, x, idr1)− >
name(idr1, n), construcMtxMatrix(i, j, v, idr1), LT(v, x);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);

subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idK), deriveFrom(idS1, idR1, id), colSums(id, idr1)− >
colSums(idK, idr2), multi(idr2, idR1, idr3), colSums(idS1, idr4), concat(idr4, idr3, idr1);


user(EID, FOLC, FRIC, LC, P, V),
tweetR(EID, FC, QC, RC, F, PS, R),
constructMatrix_11(mID1, rwID, FOLC, FRIC, LC, P, V, FC, QC, RC, F, PS, R)− >
nameI(mID1, n),
name(mID2, "tweet"),
name(mID3, "user"),
subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idr3),
deriveFrom(idS1, idR1, mID1);
name(idS, "tweet"),
name(idR, "user"),
indicator(idS, idR, idK),
subset_tweet(idS, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(idR, "FOLC", "FRIC", "LC", "P", "V", idR1),
rowSums(idS1, idr1),
rowSums(idR1, idr2),
multi(idK, idr2, idr3),
add(idr1, idr3, idr4)− > name(idr4, "VV1");

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "flevel", "o", fID),
child_sp(id, "entities", "o", eID),
child_sp(eID, "hashtag", "o", hID),
child_sp(hID, "hID", "o", hcID),
child_sp(id, "place", "o", pID),
child_sp(pID, "code", "o", cID)− > V1(tID, hcID, fID, cID);

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "text", "o", text)− >
V2(idV2),
child_sj_V2(idV2, "tweetID", "o", tID),
child_sj_V2(idV2, "text", "o", text);
```

Listing G3: Forward Constraints Used for Q3

```
#EGDs
name(id1, n1), name(id2, n1)-> id2 = id1;
tr(id1, idt1), tr(id1, idt2)-> idt2 = idt1;
in(id1, idt1), in(id1, idt2)-> idt2 = idt1;
ins(id1, idt1), ins(id1, idt2)-> idt1 = idt2;
in(id1, idr1), in(idr1, idr3)-> idr3 = id1;
multi(idA, idB, idr1), multi(idA, idB, idr2)-> idr2 = idr1;
multi_s(idA, idB, idr1), multi_s(idA, idB, idr2)-> idr2 = idr1;
det(idA, d1), det(idA, d2)-> d2 = d1;
trace(idA, id1), trace(idA, id2)-> id2 = id1;
add(idA, idB, idr1), add(idA, idB, idr2)-> idr1 = idr2;
size(idA, i1, j1), size(idA, i2, j2)-> j2 = j1, i2 = i1;
name(id1, n1), name(id2, n1)-> id2 = id1;
nameI(mID1, n1), nameI(mID1, n2)-> n1 = n2;
add_s(idA, idB, idr1), add_s(idA, idB, idr2)-> idr1 = idr2;
sum(id1, idt1), sum(id1, idt2)-> idt2 = idt1;
rowSums(idA, id1), rowSums(idA, id2)-> id2 = id1;
colSums(idA, id1), colSums(idA, id2)-> id2 = id1;
user(EID, FOLC1, FRIC1, LC1, P1, V1), user(EID, FOLC2, FRIC2, LC2, P2, V2)-> FOLC2 = FOLC1, FRIC2 = FRIC1, LC2 = LC1, P2 = P1, V2 = V1;
constructMatrix_2(rID, mID, v11, v12), constructMatrix_2(rID, mID, v21, v22)-> v21 = v11, v22 = v12;
constructMatrix_4(rID, mID, v11, v12, v13, v14), constructMatrix_4(rID, mID, v21, v22, v23, v24)-> v21 = v11, v22 = v12, v23 = v13, v24 = v14;
constructMatrix_11(rID, mID, v11, v12, v13, v14, v15, v16, v17, v18, v19, v110, v111), constructMatrix_11(rID, mID, v21, v22, v23, v24, v25, v26, v27, v28, v29, v210, v211)->
v11 = v21, v12 = v22, v13 = vv23, v14 = v24, v15 = v25, v16 = v26, v17 = v27, v18 = v28, v19 = v29, v110 = v210, v111 = v211;
constructMtxMatrix(i, j, v, id)(i1, j1, v1, id), constructMtxMatrix(i2, j2, v2, id)-> i2 = i1, j2 = j1, v2 = v1;
indicator(id1, id2, idr31), indicator(id1, id2, idr32)-> idr32 = idr31;
deriveFrom(id1, id2, idr31), deriveFrom(id1, id2, idr32)-> idr32 = idr31;
concat(id1, id2, idr1), concat(id1, id2, idr2)-> idr2 = idr1;
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr1),
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr2)-> idr2 = idr1;
subset_user(id1, id2, id3, id4, id5, id6, idr1),
subset_user(id1, id2, id3, id4, id5, id6, idr2)-> idr2 = idr1;
tweet(id1), tweet(id2)-> id2 = id1;
child_sp(x, y1, z, d), child_sp(x, y2, z, d)-> y2 = y1;
val_sp(x, v1), val_sp(x, v2)-> v2 = v1;
child_sj_V2(x, y1, z, d), child_sj_V2(x, y2, z, d)-> y2 = y1;

#TGDs
add(idA, idB, idr1)-> add(idB, idA, idr1);
multi(id1, id2, idr1), rowSums(idr1, idr2)-> rowSums(id2, idr3), multi(id1, idr3, idr2);
add(idA, idB, idr1), multi(idr1, idC, idr2)-> multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);
multi(idA, idB, idr1), multi(idr1, idC, idr2)-> multi(idB, idC, idr3), multi(idA, idr3, idr2);
construcMtxMatrix(i, j, v, id), name(id, n), matrixFilterLT(id, x, idr1)->
name(idr1, n), construcMtxMatrix(i, j, v, idr1), LT(v, x);
add(idA, idB, idr1), multi(idr1, idC, idr2)-> multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);


subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idK), deriveFrom(idS1, idR1, id), rowSums(id, idr1)->
rowSums(idS1, idr2), rowSums(idR1, idr3), multi(idK, idr3, idr4), add(idr2, idr4, idr1);


user(EID, FOLC, FRIC, LC, P, V),
tweetR(EID, FC, QC, RC, F, PS, R),
constructMatrix_11(mID1, rwID, FOLC, FRIC, LC, P, V, FC, QC, RC, F, PS, R)->
nameI(mID1, n),
name(mID2, "tweet"),
name(mID3, "user"),
subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idr3),
deriveFrom(idS1, idR1, mID1);

name(idS, "tweet"),
name(idR, "user"),
indicator(idS, idR, idK),
subset_tweet(idS, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(idR, "FOLC", "FRIC", "LC", "P", "V", idR1),
rowSums(idS1, idr1),
rowSums(idR1, idr2),
multi(idK, idr2, idr3),
add(idr1, idr3, idr4)-> name(idr4, "VV1");

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "flevel", "o", f1D),
child_sp(id, "entities", "o", e1D),
child_sp(e1D, "hashtag", "o", h1D),
child_sp(h1D, "h1D", "o", hcID),
child_sp(id, "place", "o", pID),
child_sp(pID, "code", "o", c1D)-> V1(tID, hcID, fID, cID);

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "text", "o", text)->
V2(idV2),
child_sj_V2(idV2, "tweetID", "o", tID),
child_sj_V2(idV2, "text", "o", text);
```

Listing G4: Forward Constraints Used for Q4

```
#EGDs
name(id1, n1), name(id2, n1)− > id2 = id1;
tr(id1, idt1), tr(id1, idt2)− > idt2 = idt1;
in(id1, idt1), in(id1, idt2)− > idt2 = idt1;
ins(id1, idt1), ins(id1, idt2)− > idt1 = idt2;
in(id1, idr1), in(idr1, idr3)− > idr3 = id1;
multi(idA, idB, idr1), multi(idA, idB, idr2)− > idr2 = idr1;
multi_s(idA, idB, idr1), multi_s(idA, idB, idr2)− > idr2 = idr1;
det(idA, d1), det(idA, d2)− > d2 = d1;
trace(idA, id1), trace(idA, id2)− > id2 = id1;
add(idA, idB, idr1), add(idA, idB, idr2)− > idr1 = idr2;
size(idA, i1, j1), size(idA, i2, j2)− > j2 = j1, i2 = i1;
name(id1, n1), name(id2, n1)− > id2 = id1;
nameI(mID1, n1), nameI(mID1, n2)− > n1 = n2;
add_s(idA, idB, idr1), add_s(idA, idB, idr2)− > idr1 = idr2;
sum(id1, idt1), sum(id1, idt2)− > idt2 = idt1;
rowSums(idA, id1), rowSums(idA, id2)− > id2 = id1;
colSums(idA, id1), colSums(idA, id2)− > id2 = id1;
user(EID, FOLC1, FRIC1, LC1, P1, V1), user(EID, FOLC2, FRIC2, LC2, P2, V2)− > FOLC2 = FOLC1, FRIC2 = FRIC1, LC2 = LC1, P2 = P1, V2 = V1;
constructMatrix_2(rID, mID, v11, v12), constructMatrix_2(rID, mID, v21, v22)− > v21 = v11, v22 = v12;
constructMatrix_4(rID, mID, v11, v12, v13, v14), constructMatrix_4(rID, mID, v21, v22, v23, v24)− > v21 = v11, v22 = v12, v23 = v13, v24 = v14;
constructMatrix_11(rID, mID, v11, v12, v13, v14, v15, v16, v17, v18, v19, v110, v111), constructMatrix_11(rID, mID, v21, v22, v23, v24, v25, v26, v27, v28, v29, v210, v211)− >
v11 = v21, v12 = v22, v13 = vv23, v14 = v24, v15 = v25, v16 = v26, v17 = v27, v18 = v28, v19 = v29, v110 = v210, v111 = v211;
constructMtxMatrix(i, j, v, id)(i1, j1, v1, id), constructMtxMatrix(i2, j2, v2, id)− > i2 = i1, j2 = j1, v2 = v1;
indicator(id1, id2, idr31), indicator(id1, id2, idr32)− > idr32 = idr31;
deriveFrom(id1, id2, idr31), deriveFrom(id1, id2, idr32)− > idr32 = idr31;
concat(id1, id2, idr1), concat(id1, id2, idr2)− > idr2 = idr1;
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr1),
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr2)− > idr2 = idr1;
subset_user(id1, id2, id3, id4, id5, id6, idr1),
subset_user(id1, id2, id3, id4, id5, id6, idr2)− > idr2 = idr1;
tweet(id1), tweet(id2)− > id2 = id1;
child_sp(x, y1, z, d), child_sp(x, y2, z, d)− > y2 = y1;
val_sp(x, v1), val_sp(x, v2)− > v2 = v1;
child_sj_V2(x, y1, z, d), child_sj_V2(x, y2, z, d)− > y2 = y1;

#TGDs
add(idA, idB, idr1)− > add(idB, idA, idr1);
multi(id1, id2, idr1), colSums(idr1, idr2)− > colSums(id1, idr3), multi(idr3, id2, idr2);
multi(id1, id2, idr1), rowSums(idr1, idr2)− > rowSums(id2, idr3), multi(id1, idr3, idr2);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);
multi(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idB, idC, idr3), multi(idA, idr3, idr2);
constructMtxMatrix(i, j, v, id), name(id, n), matrixFilterLT(id, x, idr1)− >
name(idr1, n), construcMtxMatrix(i, j, v, idr1), LT(v, x);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);

subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idK), deriveFrom(idS1, idR1, id), colSums(id, idr1)− >
colSums(idK, idr2), multi(idr2, idR1, idr3), colSums(idS1, idr4), concat(idr4, idr3, idr1);


user(EID, FOLC, FRIC, LC, P, V),
tweetR(EID, FC, QC, RC, F, PS, R),
constructMatrix_11(mID1, rwID, FOLC, FRIC, LC, P, V, FC, QC, RC, F, PS, R)− >
nameI(mID1, n),
name(mID2, "tweet"),
name(mID3, "user"),
subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idr3),
deriveFrom(idS1, idR1, mID1);
name(idS, "tweet"),
name(idR, "user"),
indicator(idS, idR, idK),
subset_tweet(idS, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(idR, "FOLC", "FRIC", "LC", "P", "V", idR1),
rowSums(idS1, idr1),
rowSums(idR1, idr2),
multi(idK, idr2, idr3),
add(idr1, idr3, idr4)− > name(idr4, "VV1");

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "flevel", "o", fID),
child_sp(id, "entities", "o", eID),
child_sp(eID, "hashtag", "o", hID),
child_sp(hID, "hID", "o", hcID),
child_sp(id, "place", "o", pID),
child_sp(pID, "code", "o", cID)− > V1(tID, hcID, fID, cID);

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "text", "o", text)− >
V2(idV2),
child_sj_V2(idV2, "tweetID", "o", tID),
child_sj_V2(idV2, "text", "o", text);
```

Listing G5: Forward Constraints Used for Q5

```
#EGDs
name(id1, n1), name(id2, n1)− > id2 = id1;
tr(id1, idt1), tr(id1, idt2)− > idt2 = idt1;
in(id1, idt1), in(id1, idt2)− > idt2 = idt1;
ins(id1, idt1), ins(id1, idt2)− > idt1 = idt2;
in(id1, idr1), in(idr1, idr3)− > idr3 = id1;
multi(idA, idB, idr1), multi(idA, idB, idr2)− > idr2 = idr1;
multi_s(idA, idB, idr1), multi_s(idA, idB, idr2)− > idr2 = idr1;
det(idA, d1), det(idA, d2)− > d2 = d1;
trace(idA, id1), trace(idA, id2)− > id2 = id1;
add(idA, idB, idr1), add(idA, idB, idr2)− > idr1 = idr2;
size(idA, i1, j1), size(idA, i2, j2)− > j2 = j1, i2 = i1;
name(id1, n1), name(id2, n1)− > id2 = id1;
nameI(mID1, n1), nameI(mID1, n2)− > n1 = n2;
add_s(idA, idB, idr1), add_s(idA, idB, idr2)− > idr1 = idr2;
sum(id1, idt1), sum(id1, idt2)− > idt2 = idt1;
rowSums(idA, id1), rowSums(idA, id2)− > id2 = id1;
colSums(idA, id1), colSums(idA, id2)− > id2 = id1;
user(EID, FOLC1, FRIC1, LC1, P1, V1), user(EID, FOLC2, FRIC2, LC2, P2, V2)− > FOLC2 = FOLC1, FRIC2 = FRIC1, LC2 = LC1, P2 = P1, V2 = V1;
constructMatrix_2(rID, mID, v11, v12), constructMatrix_2(rID, mID, v21, v22)− > v21 = v11, v22 = v12;
constructMatrix_4(rID, mID, v11, v12, v13, v14), constructMatrix_4(rID, mID, v21, v22, v23, v24)− > v21 = v11, v22 = v12, v23 = v13, v24 = v14;
constructMatrix_11(rID, mID, v11, v12, v13, v14, v15, v16, v17, v18, v19, v110, v111), constructMatrix_11(rID, mID, v21, v22, v23, v24, v25, v26, v27, v28, v29, v210, v211)− >
v11 = v21, v12 = v22, v13 = vv23, v14 = v24, v15 = v25, v16 = v26, v17 = v27, v18 = v28, v19 = v29, v110 = v210, v111 = v211;
constructMtxMatrix(i, j, v, id)(i1, j1, v1, id), constructMtxMatrix(i2, j2, v2, id)− > i2 = i1, j2 = j1, v2 = v1;
indicator(id1, id2, idr31), indicator(id1, id2, idr32)− > idr32 = idr31;
deriveFrom(id1, id2, idr31), deriveFrom(id1, id2, idr32)− > idr32 = idr31;
concat(id1, id2, idr1), concat(id1, id2, idr2)− > idr2 = idr1;
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr1),
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr2)− > idr2 = idr1;
subset_user(id1, id2, id3, id4, id5, id6, idr1),
subset_user(id1, id2, id3, id4, id5, id6, idr2)− > idr2 = idr1;
tweet(id1), tweet(id2)− > id2 = id1;
child_sp(x, y1, z, d), child_sp(x, y2, z, d)− > y2 = y1;
val_sp(x, v1), val_sp(x, v2)− > v2 = v1;
child_sj_V2(x, y1, z, d), child_sj_V2(x, y2, z, d)− > y2 = y1;

#TGDs
add(idA, idB, idr1)− > add(idB, idA, idr1);
multi(id1, id2, idr1), colSums(idr1, idr2)− > colSums(id1, idr3), multi(idr3, id2, idr2);
multi(id1, id2, idr1), rowSums(idr1, idr2)− > rowSums(id2, idr3), multi(id1, idr3, idr2);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);
multi(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idB, idC, idr3), multi(idA, idr3, idr2);
constructMtxMatrix(i, j, v, id), name(id, n), matrixFilterLT(id, x, idr1)− >
name(idr1, n), construcMtxMatrix(i, j, v, idr1), LT(v, x);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);

subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idK), deriveFrom(idS1, idR1, id), colSums(id, idr1)− >
colSums(idK, idr2), multi(idr2, idR1, idr3), colSums(idS1, idr4), concat(idr4, idr3, idr1);


user(EID, FOLC, FRIC, LC, P, V),
tweetR(EID, FC, QC, RC, F, PS, R),
constructMatrix_11(mID1, rwID, FOLC, FRIC, LC, P, V, FC, QC, RC, F, PS, R)− >
nameI(mID1, n),
name(mID2, "tweet"),
name(mID3, "user"),
subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idr3),
deriveFrom(idS1, idR1, mID1);
name(idS, "tweet"),
name(idR, "user"),
indicator(idS, idR, idK),
subset_tweet(idS, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(idR, "FOLC", "FRIC", "LC", "P", "V", idR1),
rowSums(idS1, idr1),
rowSums(idR1, idr2),
multi(idK, idr2, idr3),
add(idr1, idr3, idr4)− > name(idr4, "VV1");

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "flevel", "o", fID),
child_sp(id, "entities", "o", eID),
child_sp(eID, "hashtag", "o", hID),
child_sp(hID, "hID", "o", hcID),
child_sp(id, "place", "o", pID),
child_sp(pID, "code", "o", cID)− > V1(tID, hcID, fID, cID);

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "text", "o", text)− >
V2(idV2),
child_sj_V2(idV2, "tweetID", "o", tID),
child_sj_V2(idV2, "text", "o", text);
```

Listing G6: Forward Constraints Used for Q6

```
#EGDs
name(id1, n1), name(id2, n1)-> id2 = id1;
tr(id1, idt1), tr(id1, idt2)-> idt2 = idt1;
in(id1, idt1), in(id1, idt2)-> idt2 = idt1;
ins(id1, idt1), ins(id1, idt2)-> idt1 = idt2;
in(id1, idr1), in(idr1, idr3)-> idr3 = id1;
multi(idA, idB, idr1), multi(idA, idB, idr2)-> idr2 = idr1;
multi_S(idA, idB, idr1), multi_S(idA, idB, idr2)-> idr2 = idr1;
det(idA, d1), det(idA, d2)-> d2 = d1;
trace(idA, id1), trace(idA, id2)-> id2 = id1;
add(idA, idB, idr1), add(idA, idB, idr2)-> idr1 = idr2;
size(idA, i1, j1), size(idA, i2, j2)-> j2 = j1, i2 = i1;
name(id1, n1), name(id2, n1)-> id2 = id1;
nameI(mID1, n1), nameI(mID1, n2)-> n1 = n2;
add_S(idA, idB, idr1), add_S(idA, idB, idr2)-> idr1 = idr2;
sum(id1, idt1), sum(id1, idt2)-> idt2 = idt1;
rowSums(idA, id1), rowSums(idA, id2)-> id2 = id1;
colSums(idA, id1), colSums(idA, id2)-> id2 = id1;
user(EID, FOLC1, FRIC1, LC1, P1, V1), user(EID, FOLC2, FRIC2, LC2, P2, V2)-> FOLC2 = FOLC1, FRIC2 = FRIC1, LC2 = LC1, P2 = P1, V2 = V1;
constructMatrix_2(rID, mID, v11, v12), constructMatrix_2(rID, mID, v21, v22)-> v21 = v11, v22 = v12;
constructMatrix_4(rID, mID, v11, v12, v13, v14), constructMatrix_4(rID, mID, v21, v22, v23, v24)-> v21 = v11, v22 = v12, v23 = v13, v24 = v14;
constructMatrix_11(rID, mID, v11, v12, v13, v14, v15, v16, v17, v18, v19, v110, v111), constructMatrix_11(rID, mID, v21, v22, v23, v24, v25, v26, v27, v28, v29, v210, v211)->
v11 = v21, v12 = v22, v13 = vv23, v14 = v24, v15 = v25, v16 = v26, v17 = v27, v18 = v28, v19 = v29, v110 = v210, v111 = v211;
constructMtxMatrix(i, j, v, id)(i1, j1, v1, id), constructMtxMatrix(i2, j2, v2, id)-> i2 = i1, j2 = j1, v2 = v1;
indicator(id1, id2, idr31), indicator(id1, id2, idr32)-> idr32 = idr31;
deriveFrom(id1, id2, idr31), deriveFrom(id1, id2, idr32)-> idr32 = idr31;
concat(id1, id2, idr1), concat(id1, id2, idr2)-> idr2 = idr1;
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr1),
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr2)-> idr2 = idr1;
subset_user(id1, id2, id3, id4, id5, id6, idr1),
subset_user(id1, id2, id3, id4, id5, id6, idr2)-> idr2 = idr1;
tweet(id1), tweet(id2)-> id2 = id1;
child_sp(x, y1, z, d), child_sp(x, y2, z, d)-> y2 = y1;
val_sp(x, v1), val_sp(x, v2)-> v2 = v1;
child_sj_V2(x, y1, z, d), child_sj_V2(x, y2, z, d)-> y2 = y1;

#TGDs
add(idA, idB, idr1)-> add(idB, idA, idr1);
multi(idA, idB, idr1), multi(idr1, idC, idr2)-> multi(idB, idC, idr3), multi(idA, idr3, idr2);
multi(id1, id2, idr1), colSums(idr1, idr2)-> colSums(id1, idr3), multi(idr3, id2, idr2);
multi(id1, id2, idr1), rowSums(idr1, idr2)-> rowSums(id2, idr3), multi(id1, idr3, idr2);
add(idA, idB, idr1), multi(idr1, idC, idr2)-> multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);
constructMtxMatrix(i, j, v, id), name(id, n), matrixFilterLT(id, x, idr1)->
name(idr1, n), constructMtxMatrix(i, j, v, idr1), LT(v, x);
add(idA, idB, idr1), multi(idr1, idC, idr2)-> multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);

subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idK), deriveFrom(idS1, idR1, id), colSums(id, idr1)->
colSums(idK, idr2), multi(idr2, idR1, idr3), colSums(idS1, idr4), concat(idr4, idr3, idr1);


user(EID, FOLC, FRIC, LC, P, V),
tweetR(EID, FC, QC, RC, F, PS, R),
constructMatrix_11(mID1, rwID, FOLC, FRIC, LC, P, V, FC, QC, RC, F, PS, R)->
nameI(mID1, n),
name(mID2, "tweet"),
name(mID3, "user"),
subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idr3),
deriveFrom(idS1, idR1, mID1);

name(idS, "tweet"),
name(idR, "user"),
indicator(idS, idR, idK),
subset_tweet(idS, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(idR, "FOLC", "FRIC", "LC", "P", "V", idR1),
rowSums(idS1, idr1),
rowSums(idR1, idr2),
multi(idK, idr2, idr3),
add(idr1, idr3, idr4)-> name(idr4, "VV1");

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "flevel", "o", f1D),
child_sp(id, "entities", "o", e1D),
child_sp(e1D, "hashtag", "o", h1D),
child_sp(h1D, "h1D", "o", hcID),
child_sp(id, "place", "o", pID),
child_sp(pID, "code", "o", c1D)-> V1(tID, hcID, fID, cID);

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "text", "o", text)->
V2(idV2),
child_sj_V2(idV2, "tweetID", "o", tID),
child_sj_V2(idV2, "text", "o", text);
```

Listing G7: Forward Constraints Used for Q7

```
#EGDs
name(id1, n1), name(id2, n1)− > id2 = id1;
tr(id1, idt1), tr(id1, idt2)− > idt2 = idt1;
in(id1, idt1), in(id1, idt2)− > idt2 = idt1;
ins(id1, idt1), ins(id1, idt2)− > idt1 = idt2;
in(id1, idr1), in(idr1, idr3)− > idr3 = id1;
multi(idA, idB, idr1), multi(idA, idB, idr2)− > idr2 = idr1;
multi_s(idA, idB, idr1), multi_s(idA, idB, idr2)− > idr2 = idr1;
det(idA, d1), det(idA, d2)− > d2 = d1;
trace(idA, id1), trace(idA, id2)− > id2 = id1;
add(idA, idB, idr1), add(idA, idB, idr2)− > idr1 = idr2;
size(idA, i1, j1), size(idA, i2, j2)− > j2 = j1, i2 = i1;
name(id1, n1), name(id2, n1)− > id2 = id1;
nameI(mID1, n1), nameI(mID1, n2)− > n1 = n2;
add_s(idA, idB, idr1), add_s(idA, idB, idr2)− > idr1 = idr2;
sum(id1, idt1), sum(id1, idt2)− > idt2 = idt1;
rowSums(idA, id1), rowSums(idA, id2)− > id2 = id1;
colSums(idA, id1), colSums(idA, id2)− > id2 = id1;
user(EID, FOLC1, FRIC1, LC1, P1, V1), user(EID, FOLC2, FRIC2, LC2, P2, V2)− > FOLC2 = FOLC1, FRIC2 = FRIC1, LC2 = LC1, P2 = P1, V2 = V1;
constructMatrix_2(rID, mID, v11, v12), constructMatrix_2(rID, mID, v21, v22)− > v21 = v11, v22 = v12;
constructMatrix_4(rID, mID, v11, v12, v13, v14), constructMatrix_4(rID, mID, v21, v22, v23, v24)− > v21 = v11, v22 = v12, v23 = v13, v24 = v14;
constructMatrix_11(rID, mID, v11, v12, v13, v14, v15, v16, v17, v18, v19, v110, v111), constructMatrix_11(rID, mID, v21, v22, v23, v24, v25, v26, v27, v28, v29, v210, v211)− >
v11 = v21, v12 = v22, v13 = vv23, v14 = v24, v15 = v25, v16 = v26, v17 = v27, v18 = v28, v19 = v29, v110 = v210, v111 = v211;
constructMtxMatrix(i, j, v, id)(i1, j1, v1, id), constructMtxMatrix(i2, j2, v2, id)− > i2 = i1, j2 = j1, v2 = v1;
indicator(id1, id2, idr31), indicator(id1, id2, idr32)− > idr32 = idr31;
deriveFrom(id1, id2, idr31), deriveFrom(id1, id2, idr32)− > idr32 = idr31;
concat(id1, id2, idr1), concat(id1, id2, idr2)− > idr2 = idr1;
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr1),
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr2)− > idr2 = idr1;
subset_user(id1, id2, id3, id4, id5, id6, idr1),
subset_user(id1, id2, id3, id4, id5, id6, idr2)− > idr2 = idr1;
tweet(id1), tweet(id2)− > id2 = id1;
child_sp(x, y1, z, d), child_sp(x, y2, z, d)− > y2 = y1;
val_sp(x, v1), val_sp(x, v2)− > v2 = v1;
child_sj_V2(x, y1, z, d), child_sj_V2(x, y2, z, d)− > y2 = y1;

#TGDs
add(idA, idB, idr1)− > add(idB, idA, idr1);
multi(id1, id2, idr1), colSums(idr1, idr2)− > colSums(id1, idr3), multi(idr3, id2, idr2);
multi(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idB, idC, idr3), multi(idA, idr3, idr2);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);
add(idA, idB, idr1), trace(idr1, t)− > trace(idA, idr2), trace(idB, idr3), add_s(idr2, idr3, t);

construcMtxMatrix(i, j, v, id), name(id, n), matrixFilterLT(id, x, idr1)− >
name(idr1, n), construcMtxMatrix(i, j, v, idr1), LT(v, x);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);

subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idK), deriveFrom(idS1, idR1, id), colSums(id, idr1)− >
colSums(idK, idr2), multi(idr2, idR1, idr3), colSums(idS1, idr4), concat(idr4, idr3, idr1);


user(EID, FOLC, FRIC, LC, P, V),
tweetR(EID, FC, QC, RC, F, PS, R),
constructMatrix_11(mID1, rwID, FOLC, FRIC, LC, P, V, FC, QC, RC, F, PS, R)− >
nameI(mID1, n),
name(mID2, "tweet"),
name(mID3, "user"),
subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idr3),
deriveFrom(idS1, idR1, mID1);

name(idS, "tweet"),
name(idR, "user"),
indicator(idS, idR, idK),
subset_tweet(idS, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(idR, "FOLC", "FRIC", "LC", "P", "V", idR1),
rowSums(idS1, idr1),
rowSums(idR1, idr2),
multi(idK, idr2, idr3),
add(idr1, idr3, idr4)− > name(idr4, "VV1");

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "flevel", "o", fID),
child_sp(id, "entities", "o", eID),
child_sp(eID, "hashtag", "o", hID),
child_sp(hID, "hID", "o", hcID),
child_sp(id, "place", "o", pID),
child_sp(pID, "code", "o", cID)− > V1(tID, hcID, fID, cID);

tweet(id),
child_sp(id, "tweetID", "o", tID),
child_sp(id, "text", "o", text)− >
V2(idV2),
child_sj_V2(idV2, "tweetID", "o", tID),
child_sj_V2(idV2, "text", "o", text);
```

Listing G8: Forward Constraints Used for Q8

```
#EGDs
name(id1, n1), name(id2, n1)-> id2 = id1;
tr(id1, idt1), tr(id1, idt2)-> idt2 = idt1;
in(id1, idt1), in(id1, idt2)-> idt2 = idt1;
ins(id1, idt1), ins(id1, idt2)-> idt1 = idt2;
in(id1, idr1), in(idr1, idr3)-> idr3 = id1;
multi(idA, idB, idr1), multi(idA, idB, idr2)-> idr2 = idr1;
multi_s(idA, idB, idr1), multi_s(idA, idB, idr2)-> idr2 = idr1;
det(idA, d1), det(idA, d2)-> d2 = d1;
trace(idA, id1), trace(idA, id2)-> id2 = id1;
add(idA, idB, idr1), add(idA, idB, idr2)-> idr1 = idr2;
size(idA, i1, j1), size(idA, i2, j2)-> j2 = j1, i2 = i1;
name(id1, n1), name(id2, n1)-> id2 = id1;
nameI(mID1, n1), nameI(mID1, n2)-> n1 = n2;
add_s(idA, idB, idr1), add_s(idA, idB, idr2)-> idr1 = idr2;
sum(id1, idt1), sum(id1, idt2)-> idt2 = idt1;
rowSums(idA, id1), rowSums(idA, id2)-> id2 = id1;
colSums(idA, id1), colSums(idA, id2)-> id2 = id1;
user(EID, FOLC1, FRIC1, LC1, P1, V1), user(EID, FOLC2, FRIC2, LC2, P2, V2)-> FOLC2 = FOLC1, FRIC2 = FRIC1, LC2 = LC1, P2 = P1, V2 = V1;
constructMatrix_2(rID, mID, v11, v12), constructMatrix_2(rID, mID, v21, v22)-> v21 = v11, v22 = v12;
constructMatrix_4(rID, mID, v11, v12, v13, v14), constructMatrix_4(rID, mID, v21, v22, v23, v24)-> v21 = v11, v22 = v12, v23 = v13, v24 = v14;
constructMatrix_11(rID, mID, v11, v12, v13, v14, v15, v16, v17, v18, v19, v110, v111), constructMatrix_11(rID, mID, v21, v22, v23, v24, v25, v26, v27, v28, v29, v210, v211)->
v11 = v21, v12 = v22, v13 = vv23, v14 = v24, v15 = v25, v16 = v26, v17 = v27, v18 = v28, v19 = v29, v110 = v210, v111 = v211;
constructMtxMatrix(i, j, v, id)(i1, j1, v1, id), constructMtxMatrix(i2, j2, v2, id)-> i2 = i1, j2 = j1, v2 = v1;
indicator(id1, id2, idr31), indicator(id1, id2, idr32)-> idr32 = idr31;
deriveFrom(id1, id2, idr31), deriveFrom(id1, id2, idr32)-> idr32 = idr31;
concat(id1, id2, idr1), concat(id1, id2, idr2)-> idr2 = idr1;
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr1),
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr2)-> idr2 = idr1;
subset_user(id1, id2, id3, id4, id5, id6, idr1),
subset_user(id1, id2, id3, id4, id5, id6, idr2)-> idr2 = idr1;
tweet(id1), tweet(id2)-> id2 = id1;
child_sp(x, y1, z, d), child_sp(x, y2, z, d)-> y2 = y1;
val_sp(x, v1), val_sp(x, v2)-> v2 = v1;
child_sj_V2(x, y1, z, d), child_sj_V2(x, y2, z, d)-> y2 = y1;

#TGDs
colSums(id1, idr1), tr(idr1, idr2), rowSums(id2, idr3), multi_e(idr2, idr3, idr4), sum(idr4, s)->
multi(id1, id2, idr5), sum(idr5, s);
multi(id1, id2, idr1), colSums(idr1, idr2)-> colSums(id1, idr3), multi(idr3, id2, idr2);
multi(idA, idB, idr1), multi(idr1, idC, idr2)-> multi(idB, idC, idr3), multi(idA, idr3, idr2);
construcMtxMatrix(i, j, v, id), name(id, n), matrixFilterLT(id, x, idr1)->
name(idr1, n), construcMtxMatrix(i, j, v, idr1), LT(v, x);
add(idA, idB, idr1), multi(idr1, idC, idr2)-> multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);

subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idK), deriveFrom(idS1, idR1, id),
multi(idX, id, idr1)->
multi(idX, idK, idr2),
multi(idr2, idR1, idr3),
multi(idX, idS1, idr4),
concat(idr4, idr3, idr1);

user(EID, FOLC, FRIC, LC, P, V),
tweetR(EID, FC, QC, RC, F, PS, R),
constructMatrix_11(mID1, rwID, FOLC, FRIC, LC, P, V, FC, QC, RC, F, PS, R)->
nameI(mID1, n),
name(mID2, "tweet"),
name(mID3, "user"),
subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idr3),
deriveFrom(idS1, idR1, mID1);

name(idS, "tweet"),
name(idR, "user"),
indicator(idS, idR, idK),
subset_tweet(idS, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(idR, "FOLC", "FRIC", "LC", "P", "V", idR1),
name(idC, "C"),
multi(idC, idK, idr1),
multi(idr1, idR1, idr3),
multi(idC, idS1, idr4),
concat(idr4, idr3, idr5)-> name(idr5, "VV1");

tweet(id),
child_sp(id, "tweetID", "o", t1D),
child_sp(id, "flevel", "o", f1D),
child_sp(id, "entities", "o", e1D),
child_sp(eID, "hashtag", "o", h1D),
child_sp(h1D, "hID", "o", hcID),
child_sp(id, "place", "o", p1D),
child_sp(p1D, "code", "o", c1D)-> V1(t1D, hcID, f1D, c1D);

tweet(id),
child_sp(id, "tweetID", "o", t1D),
child_sp(id, "text", "o", text)->
V2(idV2),
child_sj_V2(idV2, "tweetID", "o", t1D),
child_sj_V2(idV2, "text", "o", text);
```

Listing G9: Forward Constraints Used for Q9

```
#EGDs
name(id1, n1), name(id2, n1)− > id2 = id1;
tr(id1, idt1), tr(id1, idt2)− > idt2 = idt1;
in(id1, idt1), in(id1, idt2)− > idt2 = idt1;
ins(id1, idt1), ins(id1, idt2)− > idt1 = idt2;
in(id1, idr1), in(idr1, idr3)− > idr3 = id1;
multi(idA, idB, idr1), multi(idA, idB, idr2)− > idr2 = idr1;
multi_s(idA, idB, idr1), multi_s(idA, idB, idr2)− > idr2 = idr1;
det(idA, d1), det(idA, d2)− > d2 = d1;
trace(idA, id1), trace(idA, id2)− > id2 = id1;
add(idA, idB, idr1), add(idA, idB, idr2)− > idr1 = idr2;
size(idA, i1, j1), size(idA, i2, j2)− > j2 = j1, i2 = i1;
name(id1, n1), name(id2, n1)− > id2 = id1;
nameI(mID1, n1), nameI(mID1, n2)− > n1 = n2;
add_s(idA, idB, idr1), add_s(idA, idB, idr2)− > idr1 = idr2;
sum(id1, idt1), sum(id1, idt2)− > idt2 = idt1;
rowSums(idA, id1), rowSums(idA, id2)− > id2 = id1;
colSums(idA, id1), colSums(idA, id2)− > id2 = id1;
user(EID, FOLC1, FRIC1, LC1, P1, V1), user(EID, FOLC2, FRIC2, LC2, P2, V2)− > FOLC2 = FOLC1, FRIC2 = FRIC1, LC2 = LC1, P2 = P1, V2 = V1;
constructMatrix_2(rID, mID, v11, v12), constructMatrix_2(rID, mID, v21, v22)− > v21 = v11, v22 = v12;
constructMatrix_4(rID, mID, v11, v12, v13, v14), constructMatrix_4(rID, mID, v21, v22, v23, v24)− > v21 = v11, v22 = v12, v23 = v13, v24 = v14;
constructMatrix_11(rID, mID, v11, v12, v13, v14, v15, v16, v17, v18, v19, v110, v111), constructMatrix_11(rID, mID, v21, v22, v23, v24, v25, v26, v27, v28, v29, v210, v211)− >
v11 = v21, v12 = v22, v13 = vv23, v14 = v24, v15 = v25, v16 = v26, v17 = v27, v18 = v28, v19 = v29, v110 = v210, v111 = v211;
constructMtxMatrix(i, j, v, id)(i1, j1, v1, id), constructMtxMatrix(i2, j2, v2, id)− > i2 = i1, j2 = j1, v2 = v1;
indicator(id1, id2, idr31), indicator(id1, id2, idr32)− > idr32 = idr31;
deriveFrom(id1, id2, idr31), deriveFrom(id1, id2, idr32)− > idr32 = idr31;
concat(id1, id2, idr1), concat(id1, id2, idr2)− > idr2 = idr1;
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr1),
subset_tweet(id1, id2, id3, id4, id5, id6, id7, idr2)− > idr2 = idr1;
subset_user(id1, id2, id3, id4, id5, id6, idr1),
subset_user(id1, id2, id3, id4, id5, id6, idr2)− > idr2 = idr1;
tweet(id1), tweet(id2)− > id2 = id1;
child_sp(x, y1, z, d), child_sp(x, y2, z, d)− > y2 = y1;
val_sp(x, v1), val_sp(x, v2)− > v2 = v1;
child_sj_V2(x, y1, z, d), child_sj_V2(x, y2, z, d)− > y2 = y1;

#TGDs
colSums(id1, idr1), tr(idr1, idr2), rowSums(id2, idr3), multi_e(idr2, idr3, idr4), sum(idr4, s)− >
multi(id1, id2, idr5), sum(idr5, s);
multi(id1, id2, idr1), colSums(idr1, idr2)− > colSums(id1, idr3), multi(idr3, id2, idr2);
multi(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idB, idC, idr3), multi(idA, idr3, idr2);
construcMtxMatrix(i, j, v, id), name(id, n), matrixFilterLT(id, x, idr1)− >
name(idr1, n), construcMtxMatrix(i, j, v, idr1), LT(v, x);
add(idA, idB, idr1), multi(idr1, idC, idr2)− > multi(idA, idC, c1), multi(idB, idC, c2), add(c1, c2, idr2);

subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idK), deriveFrom(idS1, idR1, id),
multi(idX, id, idr1)− >
multi(idX, idK, idr2),
multi(idr2, idR1, idr3),
multi(idX, idS1, idr4),
concat(idr4, idr3, idr1);

user(EID, FOLC, FRIC, LC, P, V),
tweetR(EID, FC, QC, RC, F, PS, R),
constructMatrix_11(mID1, rwID, FOLC, FRIC, LC, P, V, FC, QC, RC, F, PS, R)− >
nameI(mID1, n),
name(mID2, "tweet"),
name(mID3, "user"),
subset_tweet(mID2, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(mID3, "FOLC", "FRIC", "LC", "P", "V", idR1),
indicator(mID2, mID3, idr3),
deriveFrom(idS1, idR1, mID1);

name(idS, "tweet"),
name(idR, "user"),
indicator(idS, idR, idK),
subset_tweet(idS, "FC", "QC", "RC", "F", "PS", "R", idS1),
subset_user(idR, "FOLC", "FRIC", "LC", "P", "V", idR1),
name(idC, "C"),
multi(idC, idK, idr1),
multi(idr1, idR1, idr3),
multi(idC, idS1, idr4),
concat(idr4, idr3, idr5)− > name(idr5, "VV1");

tweet(id),
child_sp(id, "tweetID", "o", t1D),
child_sp(id, "flevel", "o", f1D),
child_sp(id, "entities", "o", e1D),
child_sp(eID, "hashtag", "o", h1D),
child_sp(h1D, "hID", "o", hcID),
child_sp(id, "place", "o", p1D),
child_sp(p1D, "code", "o", c1D)− > V1(t1D, hcID, f1D, c1D);

tweet(id),
child_sp(id, "tweetID", "o", t1D),
child_sp(id, "text", "o", text)− >
V2(idV2),
child_sj_V2(idV2, "tweetID", "o", t1D),
child_sj_V2(idV2, "text", "o", text);
```

Listing G10: Forward Constraints Used for Q10

H HYBRID TWITTER BENCHMARK QUERIES AND VIEWS

```
import scala.math._
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.MatrixFormat._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql._
import scala.io.Source
import scala.tools.nsc.io._
import scala.collection.immutable._
import org.apache.spark.storage.StorageLevel._

//Preprocessing M
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val User = sqlContext.read.format("csv").option("header", "true").load("User.csv");
User.createOrReplaceTempView("User")
val Tweet = sqlContext.read.format("csv").option("header", "true").load("Tweet.csv");
Tweet.createOrReplaceTempView("Tweet")
val M = spark.sql("""SELECT followers_count,friends_count, listed_count,protected,verified
                           favorite_count,quote_count, reply_count,retweet_count,favorited,
                           possibly_sensitive,retweeted
                  FROM User as U, Tweet as T
                  WHERE  T.id = U.id""")
constructMatirx (M2,"M.csv");

//Preprocessing (N)
val TweetJSON = spark.read.json("tweets/")
TweetJSON.createOrReplaceTempView("TweetJSON")
val N = spark.sql("""SELECT TJ.id, hID, TJ.filter_level
                      FROM TweetJSON AS TJ
                           LATERAL VIEW EXPLOD (entities.hashtags.id) AS hID
                      WHERE text LIKE '%covid%' AND
                            TJ.place.country_code ='US'""")
constructMTXMatirx (2380000,1000,N,"N.mtx");

//Analysis
val ml = new MLContext(spark)
ml.setConfigProperty("sysml.cp.parallel.ops","true")
ml.setConfigProperty("sysml.optlevel","4")
ml.setConfigProperty("sysml.localtmpdir","systemml")
//DML Query
val dmlTextQuery =
  s"""
    |  M  = read("M.csv", format="csv", header=FALSE, sep=",");
    |  N  = readMM("N.mtx");
    |  V1 = read("V1.csv", format="csv", header=FALSE, sep=",");
    |  U1 = read("U1.csv", format="csv", header=FALSE, sep=",");
    |  X = read("X.csv", format="csv", header=FALSE, sep=",");
    |  NF = ifelse(N<=4,N,0)
    |  print(as.scalar(NF[20000,400]))
    |  res =rowSums(X%*%M) + (((U1%*%t(V1) ) + t(NF))%*%V1)
    |  while(FALSE){}
    |   print(as.scalar(res[500,1]))
    | }
""".stripMargin
val dmlScript = dml(dmlTextQuery)
val result = ml.execute(dmlScript)
```

Listing H1: Q1

```
import scala.math._
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.MatrixFormat._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql._
import scala.io.Source
import scala.tools.nsc.io._
import scala.collection.immutable._
import org.apache.spark.storage.StorageLevel._

//Preprocessing M
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val User = sqlContext.read.format("csv").option("header", "true").load("User.csv");
User.createOrReplaceTempView("User")
val Tweet = sqlContext.read.format("csv").option("header", "true").load("Tweet.csv");
Tweet.createOrReplaceTempView("Tweet")
val M = spark.sql("""SELECT followers_count,friends_count, listed_count,protected,verified
                           favorite_count,quote_count, reply_count,retweet_count,favorited,
                           possibly_sensitive,retweeted
                    FROM User as U, Tweet as T
                    WHERE  T.id = U.id""")
constructMatirx (M2,"M.csv");

//Preprocessing (N)
val TweetJSON = spark.read.json("tweets/")
TweetJSON.createOrReplaceTempView("TweetJSON")
val N = spark.sql("""SELECT TJ.id, hID, TJ.filter_level
                       FROM TweetJSON AS TJ
                            LATERAL VIEW EXPLOD (entities.hashtags.id) AS hID
                       WHERE text LIKE '%covid%' AND
                            TJ.place.country_code ='US'""")
constructMTXMatirx (2380000,1000,N,"N.mtx");

//Analysis
val ml = new MLContext(spark)
ml.setConfigProperty("sysml.cp.parallel.ops","true")
ml.setConfigProperty("sysml.optlevel","4")
ml.setConfigProperty("sysml.localtmpdir","systemml")
//DML Query
val dmlTextQuery =
  s"""
    |  M  = read("M.csv", format="csv", header=FALSE, sep=",")
    |  N  = readMM("N.mtx")
    |  X = read("X.csv", format="csv", header=FALSE, sep=",");
    |  U = read("U.csv", format="csv", header=FALSE, sep=",");
    |  NF = ifelse(N<=4,N,0)
    |  print(as.scalar(NF[20000,400]))
    |  res =U%*%colSums(t(X%*%M))+NF
    |  while(FALSE){}
    |   print(as.scalar(res[1,80]))
    | }
""".stripMargin
val dmlScript = dml(dmlTextQuery)
val result = ml.execute(dmlScript)
```

Listing H2: Q2

```
import scala.math._
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.MatrixFormat._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql._
import scala.io.Source
import scala.tools.nsc.io._
import scala.collection.immutable._
import org.apache.spark.storage.StorageLevel._

//Preprocessing M
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val User = sqlContext.read.format("csv").option("header", "true").load("User.csv");
User.createOrReplaceTempView("User")
val Tweet = sqlContext.read.format("csv").option("header", "true").load("Tweet.csv");
Tweet.createOrReplaceTempView("Tweet")
val M = spark.sql("""SELECT followers_count,friends_count, listed_count,protected,verified
                           favorite_count,quote_count, reply_count,retweet_count,favorited,
                           possibly_sensitive,retweeted
                  FROM User as U, Tweet as T
                  WHERE  T.id = U.id""")
constructMatirx (M2,"M.csv");

//Preprocessing (N)
val TweetJSON = spark.read.json("tweets/")
TweetJSON.createOrReplaceTempView("TweetJSON")
val N = spark.sql("""SELECT TJ.id, hID, TJ.filter_level
                     FROM TweetJSON AS TJ
                         LATERAL VIEW EXPLOD (entities.hashtags.id) AS hID
                     WHERE text LIKE '%covid%' AND
                          TJ.place.country_code ='US'""")
constructMTXMatirx (2380000,1000,N,"N.mtx");

//Analysis
val ml = new MLContext(spark)
ml.setConfigProperty("sysml.cp.parallel.ops","true")
ml.setConfigProperty("sysml.optlevel","4")
ml.setConfigProperty("sysml.localtmpdir","systemml")
//DML Query
val dmlTextQuery =
  s"""
    |  M  = read("M.csv", format="csv", header=FALSE, sep=",");
    |  N  = readMM("N.mtx");
    |  X = read("X.csv", format="csv", header=FALSE, sep=",");
    |  V = read("V.csv", format="csv", header=FALSE, sep=",");
    |  NF = ifelse(N<=4,N,0)
    |  print(as.scalar(NF[20000,400]))
    |  res = ((NF+X)%*%V)%*% colSums(M)
    |  while(FALSE){}
    |   print(as.scalar(res[500,1]))
    | }
""".stripMargin
val dmlScript = dml(dmlTextQuery)
val result = ml.execute(dmlScript)
```

Listing H3: Q3

```
import scala.math._
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.MatrixFormat._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql._
import scala.io.Source
import scala.tools.nsc.io._
import scala.collection.immutable._
import org.apache.spark.storage.StorageLevel._

//Preprocessing M
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val User = sqlContext.read.format("csv").option("header", "true").load("User.csv");
User.createOrReplaceTempView("User")
val Tweet = sqlContext.read.format("csv").option("header", "true").load("Tweet.csv");
Tweet.createOrReplaceTempView("Tweet")
val M = spark.sql("""SELECT followers_count,friends_count, listed_count,protected,verified
                            favorite_count,quote_count, reply_count,retweet_count,favorited,
                            possibly_sensitive,retweeted
                    FROM User as U, Tweet as T
                    WHERE  T.id = U.id""")
constructMatirx (M2,"M.csv");

//Preprocessing (N)
val TweetJSON = spark.read.json("tweets/")
TweetJSON.createOrReplaceTempView("TweetJSON")
val N = spark.sql("""SELECT TJ.id, hID, TJ.filter_level
                      FROM TweetJSON AS TJ
                          LATERAL VIEW EXPLOD (entities.hashtags.id) AS hID
                      WHERE text LIKE '%covid%' AND
                           TJ.place.country_code ='US'""")
constructMTXMatirx (2380000,1000,N,"N.mtx");

//Analysis
val ml = new MLContext(spark)
ml.setConfigProperty("sysml.cp.parallel.ops","true")
ml.setConfigProperty("sysml.optlevel","4")
ml.setConfigProperty("sysml.localtmpdir","systemml")
//DML Query
val dmlTextQuery =
  s"""
    |  M  = read("M.csv", format="csv", header=FALSE, sep=",")
    |  N  = readMM("N.mtx")
    |  X = read("X.csv", format="csv", header=FALSE, sep=",");
    |  U = read("U.csv", format="csv", header=FALSE, sep=",");
    |  v = read("v.csv", format="csv", header=FALSE, sep=",");
    |  NF = ifelse(N<=4,N,0)
    |  print(as.scalar(NF[20000,400]))
    |  res = sum(U+NF%*%colSums(X%*%M)%*%v)
    |  while(FALSE){}
    |   print(res)
    | }
""".stripMargin
val dmlScript = dml(dmlTextQuery)
val result = ml.execute(dmlScript)
```

Listing H4: Q4

```
import scala.math._
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.MatrixFormat._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql._
import scala.io.Source
import scala.tools.nsc.io._
import scala.collection.immutable._
import org.apache.spark.storage.StorageLevel._

//Preprocessing M
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val User = sqlContext.read.format("csv").option("header", "true").load("User.csv");
User.createOrReplaceTempView("User")
val Tweet = sqlContext.read.format("csv").option("header", "true").load("Tweet.csv");
Tweet.createOrReplaceTempView("Tweet")
val M = spark.sql("""SELECT followers_count,friends_count, listed_count,protected,verified
                           favorite_count,quote_count, reply_count,retweet_count,favorited,
                           possibly_sensitive,retweeted
                  FROM User as U, Tweet as T
                  WHERE  T.id = U.id""")
constructMatirx (M2,"M.csv");

//Preprocessing (N)
val TweetJSON = spark.read.json("tweets/")
TweetJSON.createOrReplaceTempView("TweetJSON")
val N = spark.sql("""SELECT TJ.id, hID, TJ.filter_level
                     FROM TweetJSON AS TJ
                         LATERAL VIEW EXPLOD (entities.hashtags.id) AS hID
                     WHERE text LIKE '%covid%' AND
                          TJ.place.country_code ='US'""")
constructMTXMatirx (2380000,1000,N,"N.mtx");

//Analysis
val ml = new MLContext(spark)
ml.setConfigProperty("sysml.cp.parallel.ops","true")
ml.setConfigProperty("sysml.optlevel","4")
ml.setConfigProperty("sysml.localtmpdir","systemml")
//DML Query
val dmlTextQuery =
  s"""
    |  M  = read("M.csv", format="csv", header=FALSE, sep=",")
    |  N  = readMM("N.mtx")
    |  U = read("U.csv", format="csv", header=FALSE, sep=",");
    |  X = read("X.csv", format="csv", header=FALSE, sep=",");
    |  NF = ifelse(N<=4,N,0)
    |  print(as.scalar(NF[20000,400]))
    |  res = U%*%colSums(M%*%X) + NF
    |  while(FALSE){}
    |   print(res)
    | }
""".stripMargin
val dmlScript = dml(dmlTextQuery)
val result = ml.execute(dmlScript)
```

Listing H5: Q5

```
import scala.math._
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.MatrixFormat._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql._
import scala.io.Source
import scala.tools.nsc.io._
import scala.collection.immutable._
import org.apache.spark.storage.StorageLevel._

//Preprocessing M
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val User = sqlContext.read.format("csv").option("header", "true").load("User.csv");
User.createOrReplaceTempView("User")
val Tweet = sqlContext.read.format("csv").option("header", "true").load("Tweet.csv");
Tweet.createOrReplaceTempView("Tweet")
val M = spark.sql("""SELECT followers_count,friends_count, listed_count,protected,verified
                            favorite_count,quote_count, reply_count,retweet_count,favorited,
                            possibly_sensitive,retweeted
                    FROM User as U, Tweet as T
                    WHERE  T.id = U.id""")
constructMatirx (M2,"M.csv");

//Preprocessing (N)
val TweetJSON = spark.read.json("tweets/")
TweetJSON.createOrReplaceTempView("TweetJSON")
val N = spark.sql("""SELECT TJ.id, hID, TJ.filter_level
                       FROM TweetJSON AS TJ
                            LATERAL VIEW EXPLOD (entities.hashtags.id) AS hID
                       WHERE text LIKE '%covid%' AND
                             TJ.place.country_code ='US'""")
constructMTXMatirx (2380000,1000,N,"N.mtx");

//Analysis
val ml = new MLContext(spark)
ml.setConfigProperty("sysml.cp.parallel.ops","true")
ml.setConfigProperty("sysml.optlevel","4")
ml.setConfigProperty("sysml.localtmpdir","systemml")
//DML Query
val dmlTextQuery =
  s"""
    |  M  = read("M.csv", format="csv", header=FALSE, sep=",");
    |  N  = readMM("N.mtx");
    |  V1 = read("V1.csv", format="csv", header=FALSE, sep=",");
    |  U1 = read("U1.csv", format="csv", header=FALSE, sep=",");
    |  X = read("X.csv", format="csv", header=FALSE, sep=",");
    |  NF = ifelse(N<=4,N,0)
    |  print(as.scalar(NF[20000,400]))
    |  res = rowSums(t(M%*%X)) + (((U1%*%t(V1) ) + NF)%*%V1)
    |  while(FALSE){}
    |   print(res[1000,1])
    | }
""".stripMargin
val dmlScript = dml(dmlTextQuery)
val result = ml.execute(dmlScript)
```

Listing H6: Q6

```
import scala.math._
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.MatrixFormat._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql._
import scala.io.Source
import scala.tools.nsc.io._
import scala.collection.immutable._
import org.apache.spark.storage.StorageLevel._

//Preprocessing M
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val User = sqlContext.read.format("csv").option("header", "true").load("User.csv");
User.createOrReplaceTempView("User")
val Tweet = sqlContext.read.format("csv").option("header", "true").load("Tweet.csv");
Tweet.createOrReplaceTempView("Tweet")
val M = spark.sql("""SELECT followers_count,friends_count, listed_count,protected,verified
                           favorite_count,quote_count, reply_count,retweet_count,favorited,
                           possibly_sensitive,retweeted
                  FROM User as U, Tweet as T
                  WHERE  T.id = U.id""")
constructMatirx (M2,"M.csv");

//Preprocessing (N)
val TweetJSON = spark.read.json("tweets/")
TweetJSON.createOrReplaceTempView("TweetJSON")
val N = spark.sql("""SELECT TJ.id, hID, TJ.filter_level
                     FROM TweetJSON AS TJ
                         LATERAL VIEW EXPLOD (entities.hashtags.id) AS hID
                     WHERE text LIKE '%covid%' AND
                          TJ.place.country_code ='US'""")
constructMTXMatirx (2380000,1000,N,"N.mtx");

//Analysis
val ml = new MLContext(spark)
ml.setConfigProperty("sysml.cp.parallel.ops","true")
ml.setConfigProperty("sysml.optlevel","4")
ml.setConfigProperty("sysml.localtmpdir","systemml")
//DML Query
val dmlTextQuery =
  s"""
    |  M  = read("M.csv", format="csv", header=FALSE, sep=",")
    |  N  = readMM("N.mtx")
    |  X = read("X.csv", format="csv", header=FALSE, sep=",");
    |  U = read("U,csv", format="csv", header=FALSE, sep=",");
    |  NF = ifelse(N<=4,N,0)
    |  print(as.scalar(NF[20000,400]))
    |   res= X%*%NF%*%U + rowSums(t(M))
    |  while(FALSE){}
    |   print(res[2,30])
    | }
""".stripMargin
val dmlScript = dml(dmlTextQuery)
val result = ml.execute(dmlScript)
```

Listing H7: Q7

```
import scala.math._
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.MatrixFormat._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql._
import scala.io.Source
import scala.tools.nsc.io._
import scala.collection.immutable._
import org.apache.spark.storage.StorageLevel._

//Preprocessing M
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val User = sqlContext.read.format("csv").option("header", "true").load("User.csv");
User.createOrReplaceTempView("User")
val Tweet = sqlContext.read.format("csv").option("header", "true").load("Tweet.csv");
Tweet.createOrReplaceTempView("Tweet")
val M = spark.sql("""SELECT followers_count,friends_count, listed_count,protected,verified
                           favorite_count,quote_count, reply_count,retweet_count,favorited,
                           possibly_sensitive,retweeted
                    FROM User as U, Tweet as T
                    WHERE  T.id = U.id""")
constructMatirx (M2,"M.csv");

//Preprocessing (N)
val TweetJSON = spark.read.json("tweets/")
TweetJSON.createOrReplaceTempView("TweetJSON")
val N = spark.sql("""SELECT TJ.id, hID, TJ.filter_level
                       FROM TweetJSON AS TJ
                            LATERAL VIEW EXPLOD (entities.hashtags.id) AS hID
                       WHERE text LIKE '%covid%' AND
                             TJ.place.country_code ='US'""")
constructMTXMatirx (2380000,1000,N,"N.mtx");

//Analysis
val ml = new MLContext(spark)
ml.setConfigProperty("sysml.cp.parallel.ops","true")
ml.setConfigProperty("sysml.optlevel","4")
ml.setConfigProperty("sysml.localtmpdir","systemml")
//DML Query
val dmlTextQuery =
  s"""
    |  M  = read("M.csv", format="csv", header=FALSE, sep=",")
    |  N  = readMM("N.mtx")
    |  C = read("C.csv", format="csv", header=FALSE, sep=",");
    |  V = read("V.csv", format="csv", header=FALSE, sep=",");
    |  X = read("X.csv", format="csv", header=FALSE, sep=",");
    |  NF = ifelse(N<=4,N,0)
    |  print(as.scalar(NF[20000,400]))
    |  res = NF*trace(C+ V%*%colSums(M%*%X)%*%C)
    |  while(FALSE){}
    |   print(res[2,30])
    | }
""".stripMargin
val dmlScript = dml(dmlTextQuery)
val result = ml.execute(dmlScript)
```

Listing H8: Q8

```
import scala.math._
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.MatrixFormat._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql._
import scala.io.Source
import scala.tools.nsc.io._
import scala.collection.immutable._
import org.apache.spark.storage.StorageLevel._

//Preprocessing M
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val User = sqlContext.read.format("csv").option("header", "true").load("User.csv");
User.createOrReplaceTempView("User")
val Tweet = sqlContext.read.format("csv").option("header", "true").load("Tweet.csv");
Tweet.createOrReplaceTempView("Tweet")
val M = spark.sql("""SELECT followers_count,friends_count, listed_count,protected,verified
                           favorite_count,quote_count, reply_count,retweet_count,favorited,
                           possibly_sensitive,retweeted
                  FROM User as U, Tweet as T
                  WHERE  T.id = U.id""")
constructMatirx (M2,"M.csv");

//Preprocessing (N)
val TweetJSON = spark.read.json("tweets/")
TweetJSON.createOrReplaceTempView("TweetJSON")
val N = spark.sql("""SELECT TJ.id, hID, TJ.filter_level
                     FROM TweetJSON AS TJ
                         LATERAL VIEW EXPLOD (entities.hashtags.id) AS hID
                     WHERE text LIKE '%covid%' AND
                          TJ.place.country_code ='US'""")
constructMTXMatirx (2380000,1000,N,"N.mtx");

//Analysis
val ml = new MLContext(spark)
ml.setConfigProperty("sysml.cp.parallel.ops","true")
ml.setConfigProperty("sysml.optlevel","4")
ml.setConfigProperty("sysml.localtmpdir","systemml")
//DML Query
val dmlTextQuery =
  s"""
    |  M  = read("M.csv", format="csv", header=FALSE, sep=",")
    |  N  = readMM("N.mtx")
    |  C = read("C.csv", format="csv", header=FALSE, sep=",");
    |  X = read("X.csv", format="csv", header=FALSE, sep=",");
    |  NF = ifelse(N<=4,N,0)
    |  print(as.scalar(NF[20000,400]))
    |  res = X*sum(t(colSums(C)*rowSums(M))+NF
    |  while(FALSE){}
    |   print(res[2,30])
    | }
""".stripMargin
val dmlScript = dml(dmlTextQuery)
val result = ml.execute(dmlScript)
```

Listing H9: Q9

```
import scala.math._
import org.apache.sysml.api.mlcontext._
import org.apache.sysml.api.mlcontext.ScriptFactory._
import org.apache.sysml.api.mlcontext.MatrixFormat._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.sql._
import scala.io.Source
import scala.tools.nsc.io._
import scala.collection.immutable._
import org.apache.spark.storage.StorageLevel._

//Preprocessing M
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
val User = sqlContext.read.format("csv").option("header", "true").load("User.csv");
User.createOrReplaceTempView("User")
val Tweet = sqlContext.read.format("csv").option("header", "true").load("Tweet.csv");
Tweet.createOrReplaceTempView("Tweet")
val M = spark.sql("""SELECT followers_count,friends_count, listed_count,protected,verified
                           favorite_count,quote_count, reply_count,retweet_count,favorited,
                           possibly_sensitive,retweeted
                    FROM User as U, Tweet as T
                    WHERE  T.id = U.id""")
constructMatirx (M2,"M.csv");

//Preprocessing (N)
val TweetJSON = spark.read.json("tweets/")
TweetJSON.createOrReplaceTempView("TweetJSON")
val N = spark.sql("""SELECT TJ.id, hID, TJ.filter_level
                       FROM TweetJSON AS TJ
                            LATERAL VIEW EXPLOD (entities.hashtags.id) AS hID
                       WHERE text LIKE '%covid%' AND
                            TJ.place.country_code ='US'""")
constructMTXMatirx (2380000,1000,N,"N.mtx");

//Analysis
val ml = new MLContext(spark)
ml.setConfigProperty("sysml.cp.parallel.ops","true")
ml.setConfigProperty("sysml.optlevel","4")
ml.setConfigProperty("sysml.localtmpdir","systemml")
//DML Query
val dmlTextQuery =
  s"""
    |  M  = read("M.csv", format="csv", header=FALSE, sep=",")
    |  N  = readMM("N.mtx")
    |  C = read("C.csv", format="csv", header=FALSE, sep=",");
    |  X = read("X.csv", format="csv", header=FALSE, sep=",");
    |  NF = ifelse(N<=4,N,0)
    |  print(as.scalar(NF[20000,400]))
    |  res = NF*sum((X+C)%*%M)
    |  while(FALSE){}
    |   print(res[2,30])
    | }
""".stripMargin
val dmlScript = dml(dmlTextQuery)
val result = ml.execute(dmlScript)
```

Listing H10: Q10

```
T<-read.csv("Tweet.csv")
U<-read.csv("User.csv")
K <-sparseMatrix(i=1:nrow(U), j=S[,"id"], x=1)
T1 <- as.matrix(subset(T, select = c("favorite_count","quote_count","reply_count",
                                     "retweet_count","favorited","possibly_sensitive","retweeted" )))
U1 <-as.matrix(subset(U, select = c("followers_count","friends_count", "listed_count","protected","verified" )))
V3<-rowSums(T1)+K%*%rowSums(U1)
write.csv(as.matrix(V3),"V3.csv", row.names = FALSE)
```

Listing H11: V3

```
T<-read.csv("Tweet.csv")
U<-read.csv("User.csv")
K <-sparseMatrix(i=1:nrow(U), j=S[,"id"], x=1)
T1 <- as.matrix(subset(T, select = c("favorite_count","quote_count","reply_count",
                                     "retweet_count","favorited","possibly_sensitive","retweeted" )))
U1 <-as.matrix(subset(U, select = c("followers_count","friends_count", "listed_count","protected","verified" )))
V4<-cbind(colSums(T1),colSums(K)%*%U1)
write.csv(as.matrix(V4),"V4.csv", row.names = FALSE)
```

Listing H12: V4

```
T<-read.csv("Tweet.csv")
U<-read.csv("User.csv")
C<-read.csv("C.csv")
K <-sparseMatrix(i=1:nrow(U), j=S[,"id"], x=1)
T1 <- as.matrix(subset(T, select = c("favorite_count","quote_count","reply_count",
                                     "retweet_count","favorited","possibly_sensitive","retweeted" )))
U1 <-as.matrix(subset(U, select = c("followers_count","friends_count", "listed_count","protected","verified" )))
V4<-cbind(C%*%T1,(C%*%K)%*%U1)
write.csv(as.matrix(V5),"V5.csv", row.names = FALSE)
```

Listing H13: V5