HADAD: A Lightweight Approach for Optimizing Hybrid Complex Analytics Queries

Rana Alotaibi, Bogdan Cautis, Alin Deutsch, Ioana Manolescu

HADAD: A Lightweight Approach for Optimizing Hybrid Complex Analytics Queries

Rana Alotaibi†, Bogdan Cautis‡, Alin Deutsch†, Ioana Manolescu§
†UC San Diego, USA, ‡Univ. Paris Saclay, France, §Inria & Institut Polytechnique de Paris, France
{ralotaib,deutsch}@eng.ucsd.edu;bogdan.cautis@u-psud.fr;ioana.manolescu@inria.fr

ABSTRACT

Hybrid complex analytics workloads typically include (*i*) data management tasks (joins, filters, 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 a number of areas, including scientific computing, web analytics, business recommendation, natural language processing, speech recognition. Existing solutions for evaluating hybrid complex analytics queries – ranging from LA-oriented systems, to relational systems (extended to handle LA operations), to hybrid systems – fail to provide a *unified optimization framework* for such a hybrid setting. These systems either optimize data management and complex analytics tasks separately, or exploit RA properties only while leaving LA-specific optimization opportunities unexplored. Finally, they are not able to exploit *precomputed (materialized) results* to avoid computing again (part of) a given mixed (LA and RA) computation.

We describe HADAD, an *extensible lightweight approach for optimizing hybrid complex analytics queries*, based on a common abstraction that facilitates unified reasoning: a relational model endowed with integrity constraints, which can be used to express the properties of the two computation formalisms. Our approach enables full exploration of LA properties and rewrites, as well as semantic query optimization. Importantly, our approach does not require modifying the internals of the existing systems. Our experimental evaluation shows significant performance gains on diverse workloads, from LA-centered ones to hybrid ones.

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 [18, 22, 34, 41, 43], tasks that are easily expressed using *relational algebra (RA)*-based languages, as well as (*ii*) complex analytics tasks (e.g., regressions, matrices decompositions), which are mostly expressed using *linear algebra (LA)* operations [33]. Such workloads are common in several application domains including scientific computing, web analytics, business recommendation, natural language processing [38], or speech recognition [29]. To perform such analytical tasks, data scientists can choose from a variety of systems, tools, and languages. Languages/libraries such as R [3] and NumPy [2], as well as LA-oriented systems such as SystemML [19] and TensorFlow [11] treat matrices and linear algebra operations as first-class citizens: they offer a rich set of built-in LA operations and algorithms. However, it can be difficult to express data management tasks that include pre-processing and data transformation 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 haved sought to efficiently *integrate RA and LA processing in a hybrid environment* where both algebraic styles can be used together [1, 27, 32, 35, 37, 39, 45]. Some works propose calling LA packages through user defined functions (UDFs), where libraries such as R and NumPy are embedded in the host language [1]. Others suggest to treat LA objects as first-class citizens in a column-oriented store or RDBMS and using built-in functions to express LA operations [32, 37]. However, the semantics of LA operations remain hidden behind these built-in functions and UDFs, i.e., LA routines, which the optimizers treat as black-boxes. LARA [35] introduces a declarative domain-specific language for collections and matrices, which enables optimization across the two algebraic abstractions. SPORES [46] and SPOOF [21] optimize LA expressions, by converting them into RA, optimizing the latter, and then converting the result back to an (optimized) LA expression. Polystore or hybrid systems provide an environment, where mixed RA and LA programs can be written and executed across different systems [27, 45].

We identify *unexplored optimization opportunities* in existing solutions for evaluating hybrid complex analytics queries. First, they *do not fully exploit LA properties and rewrites*, whereas it has been shown that such rewrites can drastically enhance LA-based pipelines' performance [44]. Second, they *do not support semantic query optimization* [12], which includes taking advantage of *partial materialized computation results*, i.e., materialized views, known to improve the performance of a variety of queries.

We propose HADAD, an extensible, lightweight, holistic optimizer for analytical queries, based on reasoning on a common abstraction: *relational model with integrity constraints*. This brings within reach *powerful cost-based optimizations across RA and LA*, without the need to modify the internals of the existing systems; as we show, it is very easy to add within HADDAD knowledge about a wider range of LA operations than previous work could consider, while also enabling view-based rewriting and semantic query optimization using integrity constraints. The benefits of our optimized rewrites apply both to systems that provide a mixed-programming interface, such as polystores, and to LA-oriented systems designed and built for matrix operations. 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 [14] for rewriting queries across many data models, using materialized views, in a polystore setting. The novelty of HADAD is to extend the benefits of rewriting and views optimizations to LA computations, crucial for model analytics and ML workloads.

Contributions. This 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 the need to modify their internals; it is based on a powerful intermediate abstraction that supports reasoning about such a hybrid setting, namely, *a relational model with integrity constraints*.
❷ We formalize the problem of *rewriting computations using previously materialized views in this mixed setting*. 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 complex analytics queries.
❸ We provide *formal guarantees* for our solution in terms of soundness and completeness.
❹ We conduct an extensive set of *experiments on typical LA-based and hybrid pipelines*, which show the benefits of our approach.

Paper Organization. The rest of this paper is organized as follows. Section 2 formalizes the query optimization problem we solve in the context of a hybrid setting. After some preliminaries (Section 3), Section 4 provides an end-to-end overview of our approach. Section 5 presents our novel reduction of a rewriting problem with LA views into one that can be solved by existing techniques from the relational setting. Section 6 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 Section 7 and present our experimental evaluation in Section 8. We then discuss related work and conclude in Section 9.

2 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. We consider two basic data types: *relations (sets of tuples)* as in classical database modeling, 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 set R_{ops} of (unary or binary) *relational algebra operators*; concretely, R_{ops} comprises the standard relational selection, projection, and join. We also consider a set L_{ops} of *linear algebra operators*, comprising: unary operators which apply to a matrix and return a matrix (e.g., inversion and transposition), a number (e.g., the trace), or two matrices (e.g., the LU decomposition [36]); unary operations applied to a matrix and a number and returning a matrix, such as the scalar-matrix multiplication; binary operations applied to two matrices, or a matrix and a number, and returning a matrix or a number, such as matrix sum and product, scalar product, etc. The full set L_{ops} of LA operations we support is detailed in Section 5.1. A *hybrid (RA and/or LA) expression* 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 (potentially empty) set of *materialized views* $\mathcal{V}$, expressions 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 (lowest-cost) 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 RA/LA setting. Importantly, our solution (based on a *relational encoding with integrity constraints*) capitalizes on the framework previously introduced in [14], 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 standard database 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 modern analytics and ML workloads**. In what follows (Section 5), 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.

3 PRELIMINARIES

We recall conjunctive queries [23], integrity constraints [12], and query rewriting under constraints [31]; these concepts are at the core of our approach.

3.1 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 [12]. 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 [25].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: $\forall p \forall r \forall t\ Review(p, r, t) \rightarrow \exists a\ PC(r, a)$.

3.2 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 [26] and improved in [31], 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 $\mathcal{I} \cup C_{\mathcal{V}}$.

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:

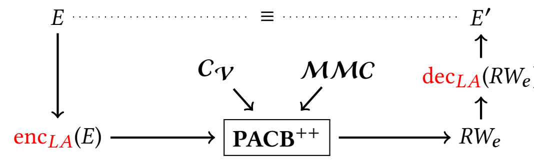


Figure 1: Outline of our reduction

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

(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 $\mathcal{I} \cup C^{OI}_{\mathcal{V}}$, where $C^{OI}_{\mathcal{V}} = \{V_{OI} \mid V \in \mathcal{V}\}$, 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, $C^{IO}_{\mathcal{V}} = \{V_{IO}\}$, 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 $C^{OI}_{\mathcal{V}}$ 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 $\rho(x, y)$:- $V(x, y)$.

4 HADAD OVERVIEW

We outline here our approach as an extension to [14] for solving the rewriting problem for LA-based computations.

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. Other expressions are purely linear-algebra ones (LA); we assume that they are defined in a dedicated LA language such as R [3], DML [19], etc. , using linear algebra operators from our set L_{ops} (see Section 5.1), commonly used in real-world machine learning 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).

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

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 linear algebra 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 LA operations L_{ops}. For instance, we model the *matrix addition* operation using a relation $add_M(M, N, R)$ to denote that O is the result of $M + N$, together with a set of constraints stating that add_M is a *functional* relation that is *commutative*, *associative*, etc. These constraints are EGDs or TGDs (recall Section 3). We detail our relational encoding in Section 5.

Reduction form LA-based to Relational Rewriting. Our reduction translates the declaration of each view $V \in \mathcal{V}$ to additional 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 basic ingredients (matrices).

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 $C_{\mathcal{V}} = enc_{LA}(V_1) \cup \ldots \cup enc_{LA}(V_n)$ of relational integrity constraints encoding the views $\mathcal{V}$. We add as further input a set of relational constraints $enc_{LA}(LA_{prop})$, which encodes relationally the LA_{prop} operators; 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 $C_{\mathcal{V}}$ 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 ($C_{\mathcal{V}} \bigcup \mathcal{MMC}$). 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 the (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'. The reduction is detailed in Section 5.

Relational Rewriting Using Constraints. To solve the relational rewriting problem under constraints, the algorithm of choice is PACB (recall Section 3). Our PACB rewriting engine ($PACB^{++}$ hereafter) has been extended to utilize the $Pruned_{prov}$ algorithm sketched and discussed in [30, 31], which prunes inefficient rewritings during the rewritings search phase, based on a simple cost model using two different matrix sparsity estimators. Section 6 details the choice of an efficient rewriting utilizing the $PACB^{++}$ engine.

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.).

5 REDUCTION TO THE RELATIONAL MODEL

Our internal model is relational, and it makes prominent use of expressive integrity constraints (TGDs and EGDs, recall Section 3). This framework suffices to describe the features and properties of most data models used today, notably including relational, XML, JSON, graph, etc [14, 15].

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 Section 5.1. Then, in Section 5.2, 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. Section 5.3 exemplifies relational rewritings obtained via our reduction.

5.1 Matrix Algebra

We consider a wide range of matrix operations [17, 36], which are common in real-world machine learning algorithms [5]: element-wise multiplication (i.e., Hadamard-product) ($multi_E$), matrix-scalar multiplication ($multi_{MS}$), matrix multiplication ($multi_M$), addition (add_M), division (div_M), transposition (tr), inversion (inv_M), determinant (det), trace ($trace$), exponential (exp), adjoints (adj), direct sum (sum_D), direct product ($product_D$), summation (sum_M), rows and columns summation ($rowSum_M$ and $colSum_M$, respectively), QR decomposition (QR), Cholesky decomposition (cho), LU decomposition (LU), and pivoted LU decomposition (LUP).

5.2 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.

5.2.1 *Base Matrices and Dimensionality Modeling.*

We denote by $M_{k\times z}(\mathcal{D})$ 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 $M_{k\times z}$. We define a virtual relation $name(M_1^i, n) \in \mathcal{VREM}$ attaching a unique ID M_1^i 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*:

$$\mathcal{I}_{name}\colon \forall M_1^i \forall M_2^i\ name(M_1^i, n) \land name(M_2^i, n) \rightarrow M_1^i = M_2^i$$

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.

The dimensions of a matrix are captured by a $size(M_1^i, k, z)$ relation, where k and z are the number of rows, resp. columns and M_1^i 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:

Operation	Encoding	Operation	Encoding	Operation	Encoding
Matrix scan	$name(M_1^i, n)$	Inversion	$inv_M(M_1^i, R^o)$	Cells sum	$sum_M(M_1^i, s)$
Multiplication	$multi_M(M_1^i, M_2^i, R^o)$	Scalar Multiplication	$multi_{MS}(s, M_1^i, R^o)$	Row sum	$rowSum_M(M_1^i, R^o)$
Addition	$add_M(M_1^i, M_2^i, R^o)$	Determinant	$det(M_1^i, R^o)$	Col sums	$colSum_M(M_1^i, R^o)$
Division	$div_M(M_1^i, M_2^i, R^o)$	Trace	$trace(M_1^i, s)$	Direct sum	$sum_D(M_1^i, M_2^i, R^o)$
Hadamard product	$multi_E(M_1^i, M_2^i, R^o)$	Exponential	$exp(M_1^i, R^o)$	Kronecker product	$product_D(M_1^i, M_2^i, R^o)$
Transposition	$tr(M_1^i, R^o)$	Adjoints	$adj(M_1^i, R^o)$	Diagonal	$diag_M(M_1^i, R^o)$

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

$$\mathcal{I}_{size}\colon \forall M_1^i \forall k_1 \forall z_1 \forall k_2 \forall z_2$$
$$size(M_1^i, k_1, z_1) \land size(M_1^i, k_2, z_2) \rightarrow k_1 = k_2 \land z_1 = z_2$$

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

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

5.2.2 ***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 $\mathcal{VREM}$ schema.

Example 5.1. *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(E) takes as argument the LA expression E and returns a conjunctive query whose: (i) body is the relational encoding of E using $\mathcal{VREM}$ (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 $enc(M) = \boldsymbol{Q}_0(M_1^i)$:- $name(M_1^i$, "M.csv");
$enc(N) = \boldsymbol{Q}_1(N_1^i)$:- $name(N_1^i$, "N.csv");
R_1^o = freshId()
in
$\boldsymbol{Q}_2(R_1^o)$:- $multi_M(M_1^i, N_1^i, R_1^o)$,$\boldsymbol{Q}_0(M_1^i)$, $\boldsymbol{Q}_1(N_1^i)$;
R_2^o = freshId()
in
$\boldsymbol{Q}(R_2^o)$:- $tr(R_1^o, R_2^o)$, $\boldsymbol{Q}_2(R_1^o)$;

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^o), using the $multi_M$ relation and encoding the relationship between this product and its inputs in the definition of $\boldsymbol{Q}_2(R_1^o)$. Next, we create a fresh ID R_2^o used to encode the full E (the transposed of $\boldsymbol{Q}_2$) via relation tr, in the query $\boldsymbol{Q}(R_2^o)$. For brevity, we omit the matrices' size relations in this example and hereafter. Unfolding $\boldsymbol{Q}_2(R_1^o)$ in the body of $\boldsymbol{Q}$ yields:

$\boldsymbol{Q}(R_2^o)$:- $tr(R_1^o, R_2^o)$, $multi_M(M_1^i, N_1^i, R_1^o)$,
$\boldsymbol{Q}_0(M_1^i)$,$\boldsymbol{Q}_1(N_1^i)$;

$$\forall M_1^i \forall M_2^i \forall R^o \; add_M(M_1^i, M_2^i, R^o) \rightarrow add_M(M_2^i, M_1^i, R^o) \quad (1)$$
$$\forall M_1^i \forall M_2^i \forall R_1^o \forall R_2^o \; add_M(M_1^i, M_2^i, R_1^o) \land tr(R_1^o, R_2^o) \rightarrow$$
$$\exists R_3^o \exists R_4^o \; tr(M_1^i, R_3^o) \land tr(M_2^i, R_4^o) \land add_M(R_3^o, R_4^o, R_2^o) \quad (2)$$
$$\forall M_1^i \forall R_1^o \forall R_2^o \; inv_M(M_1^i, R_1^o) \land tr(R_1^o, R_2^o) \rightarrow$$
$$\exists R_3^o \; tr(M_1^i, R_3^o) \land inv_M(R_3^o, R_2^o) \quad (3)$$

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

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 $\boldsymbol{Q}$:

$\boldsymbol{Q}(R_2^o)$:- $tr(R_1^o, R_2^o)$, $multi_M(M_1^i, N_1^i, R_1^o)$,
$name(M_1^i$, "M.csv"),$name(N_1^i$, "N.csv");

5.2.3 ***Encoding LA Properties as Integrity Constraints***. Figure 2 shows some of the constraints $\mathcal{MMC}_{LA_{prop}} \subset \mathcal{MMC}$, which capture textbook LA properties [17, 36] of our LA operations (Section 5.1). 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_1^i is equivalent to the inverse of the transposition of M_1^i, respectively. We also express that the *virtual relations are functional* by using EGD key constraints. For example, the following $\mathcal{I}_{multi_M} \in \mathcal{MMC}_{LA_{prop}}$ constraint states that $multi_M$ is functional, that is the products of pairwise equal matrices are equal.

$$\mathcal{I}_{multi_M} : \forall M_1^i \forall M_2^i \forall R_1^o \forall R_2^o$$
$$multi_M(M_1^i, M_2^i, R_1^o) \land multi_M(M_1^i, M_2^i, R_2^o) \rightarrow R_1^o = R_2^o$$

Other properties [17, 36] of the LA operations we consider are similarly encoded; due to space constraints, we relegate them to the technical report [4].

5.2.4 ***Encoding LA Views as Constraints.*** We translate each view definition V (defined in LA language such as R, DML, etc) into relational constraints $enc_{LA}(V) \in \mathcal{C}_{\mathcal{V}}$, where $\mathcal{C}_{\mathcal{V}}$ is the set of relational constraints used to capture the views $\mathcal{V}$. These constraints show how the view's inputs are related to its output over the $\mathcal{VREM}$ 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).

5.2.5 ***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

$$\forall M_1^i \forall N_1^i \forall R_1^o \forall R_2^o \forall R_3^o \forall R_4^o$$
$$name(M_1^i, \text{“}M.csv\text{”}) \wedge\ name(N_1^i, \text{“}N.csv\text{”}) \wedge\ tr(N_1^i, R_1^o) \wedge$$
$$tr(M_1^i, R_2^o) \wedge\ inv_M(R_2^o, R_3^o) \wedge$$
$$add_M(R_1^o, R_3^o, R_4^o) \rightarrow name(R_4^o, \text{“}V.csv\text{”})$$

Figure 3: Relational Encoding of view V

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 $\mathcal{VREM}_{dec}$, which we add to $\mathcal{VREM}$. For instance, to CD we associate a relation $cho(M_1^i, L^o)$, which denotes that L^o is the output of the CD decomposition for a given matrix M whose ID is M_1^i. *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 $\mathcal{I}_{multi_M}$ (Section 5.2.3). The property $M = LL^T$ is captured as a TGD constraint $I_{cho} \in \mathcal{MMC}_{LA_{prop}}$:

$$\mathcal{I}_{cho} : \forall M_1^i\ type(M_1^i, \text{“}S\text{”}) \rightarrow \exists\ L_1^o \exists L_2^o\ cho(M_1^i, L_1^o) \wedge$$
$$type(L_1^o, \text{“}L\text{”}) \wedge tr(L_1^o, L_2^o) \wedge multi_M(L_1^o, L_2^o, M_1^i) \quad (4)$$

The atom $type(M_1^i,\text{“S"})$ indicates the type of matrix M_1^i, where the constant “S" denotes a matrix that is *s*ymmetric positive definite; similarly, $type(L_1^o,\text{“L"})$ denotes that the matrix L_1^o 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:

$$\forall M_1\ name(M_1^i, \text{“}M.csv\text{”}) \rightarrow type(M_1^i, \text{“}S\text{”}) \quad (5)$$

Example 5.2. *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 [36] 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 $\mathcal{MMC}_{LAprop}$:

$$\forall M_1^i \forall n \forall k\ name(M_1^i, n) \wedge size(M_1^i, k, k) \rightarrow\ \exists Q^o \exists R^o$$
$$QR(M_1^i, Q^o, R^o) \wedge type(Q^o, \text{“}O\text{”}) \wedge type(R^o, \text{“}U\text{”})$$
$$\wedge\ multi_M(Q^o, R^o, M_1^i) \quad (6)$$
$$\forall Q_1^i\ type(Q_1^i, \text{“}O\text{”}) \rightarrow \exists I^o\ QR(Q_1^i, Q_1^i, I^o) \wedge identity(I^o)$$
$$\wedge\ multi_M(Q_1^i, I^o, Q_1^i) \quad (7)$$
$$\forall R_1^i\ type(R_1^i, \text{“}U\text{”}) \rightarrow \exists I^o\ QR(R_1^i, I^o, R_1^i) \wedge identity(I^o)$$
$$\wedge\ multi_M(I^o, R_1^i, R_1^i) \quad (8)$$
$$\forall I_1^i\ identity(I_1^i) \rightarrow QR(I_1^i, I_1^i, I_1^i) \quad (9)$$

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

5.2.6 *Encoding LA-Oriented System Rewrite Rules*.

Most LA-oriented systems [2, 3] 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 [19] 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 $rowSums(M)$, $sum(M)$, etc. For example, SystemML uses rule:

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

to rewrite $sum(MN)$ (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 $sum(M^T)$ into $sum(M)$, 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.

Example 5.3. *Consider the LA expression E=$((M^T)^k(M + N)^T)$, where M and N are square matrixes, and expression E'=$sum(E)$, which computes the sum of all cells in E. The expression E' can be rewritten to $RW_0 : sum(E'')$, where E'' is:*

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

Failure to exploit the properties LA_{prop1} : $(MN)^T$=M^TN^T and/or LA_{prop2} : $(M^n)^T = (M^T)^n$ prevents from finding rewriting RW_0. E' admits the alternative rewriting

RW_1: $sum(t(colSums((M^T)^k)) \odot\ t(colSums(M + N))$

which can be obtained by directly applying the rewrite rule (i) above and $rowSums(M^T)$=$colSums(M)^T$, without exploiting the properties LA_{prop1} and LA_{prop2}. However, RW_1 introduces more intermediate results than RW_0.

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 $\mathcal{VREM}$, denoted $\mathcal{MMC}_{StatAgg}$ ($\mathcal{MMC}_{StatAgg} \subset \mathcal{MMC}$).

$RW_1 : (M^{-1})^T + N^T \quad RW_2 : (M^T)^{-1} + N^T$

$RW_3 : N^T + (M^{-1})^T \quad RW_4 : (N^T)^{-1} + N^T$

$RW_5 : (N + M^{-1})^T$

Figure 4: Equivalent rewritings of the pipeline Q_p.

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 $\mathcal{I}_{sum} \in \mathcal{MMC}_{StatAgg}$:

$$\forall M_1^i \forall N_1^i \forall R^o \; multi_M(M_1^i, N_1^i, R^o) \wedge sum(R^o, s) \rightarrow$$
$$\exists R_1^o \exists R_2^o \exists R_3^o \exists R_4^o colSums(M_1^i, R_1^o) \wedge tr(R_1^o, R_2^o)$$
$$\wedge\; rowSums(N_1^i, R_3^o) \wedge multi_E(R_2^o, R_3^o, R_4^o) \wedge sum(R_4^o, s)$$

We refer the reader to the extended version of the paper [4] for a full list of SystemML's encoded rewrite rules.

5.3 Relational Rewriting Using Constraints

With the set of views constraints $C_{\mathcal{V}}$ 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 6.

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 $RW_0(R_4^o)$:- $name(R_4^o, \text{“}V.csv\text{”})$. 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.

6 CHOICE OF AN EFFICIENT REWRITING

We introduce our cost model (Section 6.1), which can take two different sparsity estimators (Section 6.2). Then, we detail our extension to the PACB rewriting engine based on the $Prune_{prov}$ algorithm (Section 6.3) to prune out inefficient rewritings.

6.1 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 Section 6.2) into our framework.

Example 6.1. *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 $\gamma(E_2) = 100 \times 100$.*

6.2 LA-based Sparsity Estimators

We outline below two existing *sparsity estimators* [19, 46] that we have incorporated into our framework to estimate *nnz*.

6.2.1 ***Naïve Metadata Estimator.*** The naïve metadata estimator [20, 46] 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* [20], which we use in our framework.

6.2.2 ***Matrix Non-zero Count (MNC) Estimator.*** The MNC estimator [42] 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 Section 8.

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

We extended the PACB rewriting engine with the $Prune_{prov}$ algorithm sketched and discussed in [30, 31], 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 [14, 31], 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 Section 3 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 6.2. *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:*

$$U(R_2^o) : name(M_1^i, \text{“}M.csv\text{”})^{p_0} \wedge size(M_1^i, 50000, 100)^{p_1} \wedge$$
$$name(N_1^i, \text{“}N.csv\text{”})^{p_2} \wedge size(N_1^i, 100, 50000)^{p_3} \wedge$$
$$multi_M(M_1^i, N_1^i, R_1^o)^{p_4} \wedge multi_M(R_1^o, M_1^i, R_2^o)^{p_5} \wedge$$
$$multi_M(N_1^i, M_1^i, R_3^o)^{p_6} \wedge multi_M(M_1^i, R_3^o, R_2^o)^{p_7}$$

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

$$\forall M_1^i \forall N_1^i \forall R_1^o \forall R_2^o$$
$$multi_M(M_1^i, N_1^i, R_1^o) \wedge multi_M(R_1^o, M_1^i, R_2^o) \rightarrow$$
$$\exists R_4^o multi_M(N_1^i, M_1^i, R_4^o) \wedge multi_M(M_1^i, R_4^o, R_2^o)$$

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*

$multi_M(N_1^i, M_1^i, R_4^o)^{p_4 \wedge p_5} \wedge multi_M(M_1^i, R_4^o, R_2^o)^{p_4 \wedge p_5}$

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 $sq(\pi(h(P)) = multi_M(M_1^i, N_1^i, R_1^o)^{p_4} \wedge multi_M(R_1^o, M_1^i, R_2^o)^{p_5}$*

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. 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 Section 3) whenever the pruned chase step is applied is modeled as a query evaluation of Q against U' (viewed as a symbolic/canonical database [12]). 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 [31]. 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.

7 GUARANTEES ON THE REDUCTION

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

We denote with $\mathcal{L}$ the language of hybrid expressions described in Section 2. Let $\mathcal{V} \subseteq \mathcal{L}$ be a set of materialized view definitions.

Let LA_{prop} be a set of properties of the LA operations in L_{ops} that admits relational encoding over $\mathcal{VREM}$. 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 7.1 (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 7.2 (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 $rw \in HADAD\langle LA_{prop}, \mathcal{V}, \gamma\rangle(E)$.*

8 EXPERIMENTAL EVALUATION

We evaluate our approach, first on **LA pipelines** (Section 8.1), then on **real-world hybrid scenarios** (Section 8.2). Due to space constraints, we delegate other results to our extended version of the paper [4].

Experimental Environment. We used a single node with an Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz, 40 Cores (hyper-threading), 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)**, and **SystemML 1.2.0**; hybrid scenarios were evaluated in the **SparkSQL** [16] polystore.

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 [44], at the optimization level **4**. Additionally, we enable multi-threaded matrix operations in a single node. We run Spark using the standalone cluster manager, and using OpenBLAS (built from the sources as detailed in [10]) to take advantage of its accelerations [44]. 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 [7], 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.

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	$trace(s_1D)$
P1.7	$((A)^T)^T$	P1.8	$s_1A + s_2A$	P1.9	$det(D^T)$
P1.10	$rowSums(A^T)$	P1.11	$rowSums(A^T + B^T)$	P1.12	$colSums(MN)$
P1.13	$sum(MN)$	P1.14	$sum(colSums(N^TM^T))$	P1.15	$(MN)M$
P1.16	$sum(A^T)$	P1.17	$det(CDC)$	P1.18	$sum(colSums(A))$
P1.19	$(C^T)^{-1}$	P1.20	$trace(C^{-1})$	P1.21	$(C+D^{-1})^T$
P1.22	$trace((C+D)^{-1})$	P1.23	$det((CD)^{-1}) + D)$	P1.24	$trace((CD)^{-1})) + trace(D)$
P1.25	$M \odot (N^T/(MNN^T))$	P1.26	$N \odot (M^T/(M^TMN))$	P1.27	$trace(D(CD)^T)$
P1.28	$A \odot (A \odot B + A)$	P1.29	$DCCC$	P1.30	$NM \odot NMR^T$

Table 2: LA Benchmark Pipelines (Part 1)

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 3: 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 4: 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 5: Matrices used for each matrix name in a pipeline

Datasets. We used several **real-world, sparse matrices**, for which Table 3 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 [24]; (*ii*) 2D_54019_highK (**2D_54019** in short) is a 2D semiconductor device simulation [24]; (*iii*) we used several subsets of an Amazon books review dataset [6] (in JSON), and similarly (*iv*) subsets of a Netflix movie rating dataset [8]. The latter two were easily converted into matrices where columns are items and rows are customers [46]; 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 4.

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

Real-world matrix expressions include: a chain of matrix self-products used for reachability queries and other graph analytics [42] (**P1.29** in Table 18); expressions used in Alternating Least Square Factorization (ALS) [46] (**P2.25** in Table 17); Poisson Nonnegative Matrix Factorization (PNMF) [46] (**P1.13** in Table 18); Nonnegative Matrix Factorization (NMF) [44](**P1.25** and **P1.26** in Table 18); complex predicate for image masking [42] (**P1.30** in Table 18); recommendation computation [42] (**P1.30** in Table 18); finally, Ordinary Least Squares Regression (OLS) [44] (**P2.21** in Table 17).

Synthetic expressions 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) [40]. Expressions thus generated include **P2.16**, **P2.16**, **P2.23**, **P2.24** in Table 18.

Methodology. We evaluate our approach using the LA pipelines in Table 18 and Table 17, systems/tools mentioned above, and the matrices in Table 5. For TensorFlow and NumPy, we present the results only for dense matrices, due to limited support for sparse matrices. In Section 8.1, we focus on LA pipeline rewriting, while Section 8.2 describes experiments on *real-world, hybrid* scenarios.

8.1 Experiments on LA-based Pipelines

In Section 8.1.1, we show the performance benefits of our approach to existing LA systems using a set $\mathcal{P}^{\neg Opt} \subset \mathcal{P}$ of **38** pipelines, whose performance can be improved *just by exploiting LA properties* (in the absence of views). In Section 8.1.2, we study our *optimization overhead* for the set $\mathcal{P}^{Opt} = \mathcal{P} \setminus \mathcal{P}^{\neg Opt}$ of **19** pipelines that are already optimized. Finally, in Section 8.1.3, we show how our approach improves the performance of **30** pipelines from $\mathcal{P}$, denoted $\mathcal{P}^{Views}$, using pre-materialized views.

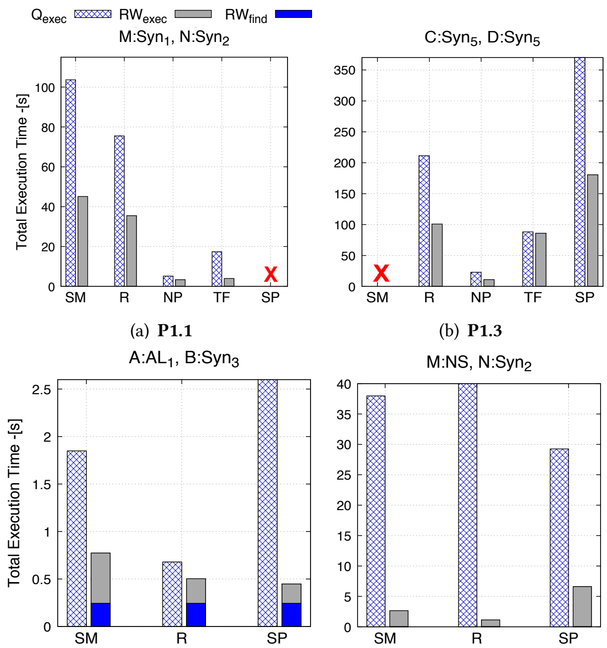
8.1.1 *Effectiveness of LA Rewriting (No Views).*

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

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

Table 6: LA Benchmark Pipelines (Part 2)



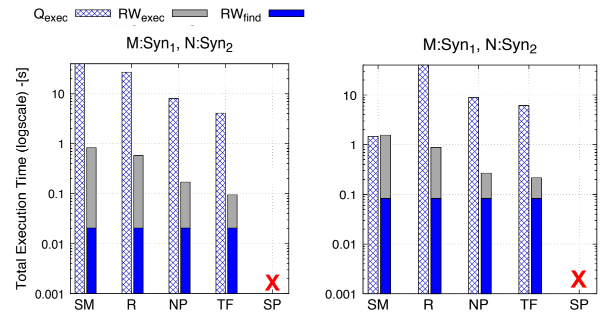
(a) **P1.1** (b) **P1.3** (c) **P1.4** (d) **P1.15**

Figure 5: Evaluation time with and without rewriting

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 $50K \times 100$. 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 runtime exception 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 ≈ 1.8× 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.



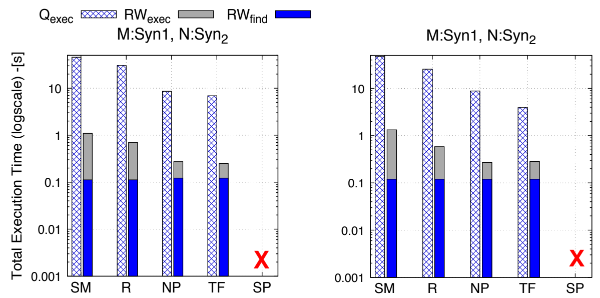
(a) **P1.13** (b) **P1.25**

Figure 6: P1.13 and P1.25 evaluation before and after rewrite

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 [44]).

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) [9]. 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.

Figure 6 studies **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**: $sum(MN)$ into $sum(t(colSums(M)) * rowSums(N))$ 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*



(a) **P1.14** (b) **P2.12**

Figure 7: P1.14 and P2.12 evaluation before and after rewrite

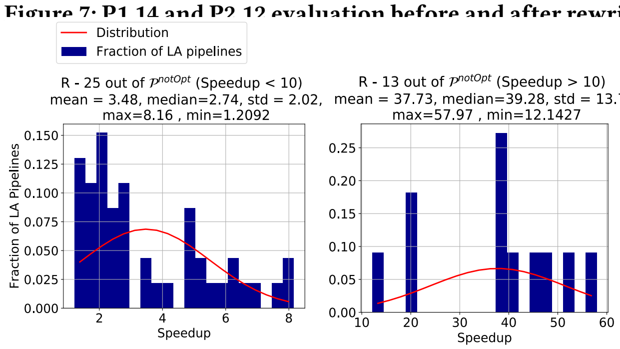


Figure 8: R speed-up on $\mathcal{P}^{\neg Opt}$

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.

Figure 7 shows up to 42× rewriting speed-up achieved by turning **P1.14** and **P2.12** into $sum(t(colSums(M)) * rowSums(N))$. This exploits several properties: (i) $(MN)^T = N^T M^T$, (ii) $sum(M^T) = sum(M)$, (iii) $sum(row/colSums(M)) = sum(M)$, and (iv) $sum(MN) = sum(t(colSums(M)) * rowSums(N))$. 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.

8.1.2 *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 5, 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 HADAD's 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.

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.

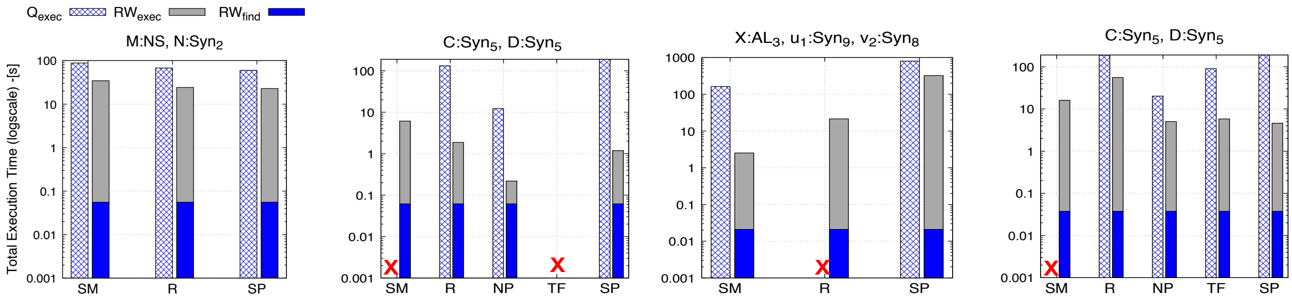
8.1.3 *Effectiveness of view-based LA rewriting*.

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 9(a)), using the view $V_4 = NM$ by and the multiplication associativity leads to up to 2.8× speed-up.

Figure 9(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 9(c)) benefits from a view V_5, which precomputes 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



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

Figure 9: P2.14, P2.21, P2.25 and P2.27 performance before and after rewriting using the views V_{exp}

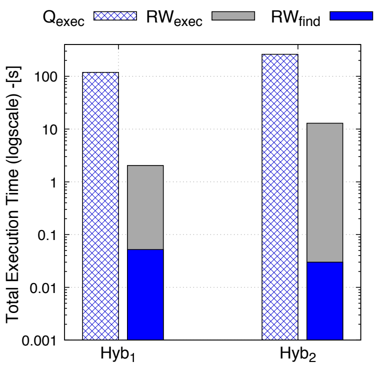


Figure 10: Hyb_1 and Hyb_2 evaluation before and after rewrite

vectors, thus they were treated as dense. In R, the original pipeline triggers a memory allocation failure, which the rewriting avoids.

Figure 9(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 $(CD)E = C(DE)$.

8.2 Hybrid (LA and RA) Scenarios

We now study the benefits of rewriting on hybrid expressions combining relational and linear algebra.

Scenario 1 (Hyb_1): Amazon reviews. This JSON dataset contains product information, product reviews (text) and reviewer information. We defined two materialized views: V_1 stores review id, product id, and the overall rate for "Kindle-Edition" books as a *relational table*; V_2 stores the reviewer id, product id, and the review text as a *text datasource in Solr*. A query constructs a product-review matrix X for "Kindle-Edition" books, where the review text mentions "mystery". This matrix is loaded in SystemML, where we filter rows with rating less than 2. Next, an analysis is run, through the computation: $(u_1 v_2^T - X)v_2$, which appears in the ALS algorithm [46]. Note that u_1 and v_2 are synthetic dense vectors of size $100K \times 1$ and $50K \times 1$, respectively, and X is ultra sparse (0.00028% non-zeros). Rewriting *modifies the pre-processing* by introducing V_1 and V_2; it also *pushes the rating filter* into the pre-processing part. The analysis is rewritten into $u_1 v_2^T v_2 - X v_2$; this is the main reason for the 60× speed-up we bring to SystemML (Figure 10). First, X is ultra sparse which makes the computation of Xv_2 extremely efficient. Second, SystemML evaluates $u_1 v_2^T v_2$ 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 $v_2^T v_2$ first, which results in a scalar, instead of computing $u_1 v_2^T$, which results in a dense matrix of size $100K \times 50K$. Alone, SystemML is unable to exploit its own efficient operations for lack of awareness the LA property $Av + Bv = (A + B)v$.

Scenario 2 (Hyb_2): Netflix reviews. The dataset is in CSV form, and it contains movie id, customer id, overall rate and rating time. A query (in the pre-processing part) constructs a review matrix X, where columns are movies, and rows are customers. In the same fashion as in the previous scenario, X is loaded for analysis in SystemML, where we filter rows with overall rate greater than 3. Next, we run the same analysis computation described in Scenario 1, where the obtained rewriting is also the same. The main difference here is that matrix X is much denser (0.261% non-zeros), and the rewriting does not involve using materialized views. The rewrite achieves ~20× speed-up as shown in Figure 10.

8.3 Experiments Summary

We have shown that HADAD brings significant performance-saving across LA-oriented and potentially polystore engines/systems 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 for $\mathcal{P}^{\neg Opt}$ pipelines hence a worthwhile investment. In addition, the rewriting overhead of $\mathcal{P}^{Opt}$ pipelines 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.

9 RELATED WORK AND CONCLUSION

LA-oriented Systems, Libraries & Languages. SystemML [19] offers high-level R-like linear algebra abstractions, using a declarative language called Declarative Machine Learning (DML). The system applies some logical LA pattern-based rewrites and physical execution optimizations, based on cost estimates for the latter. SparkMLlib [39] provides LA operations and built-in function implementations of popular ML algorithms, such as linear regression, etc. on Spark RDDs. The library supports sparse and dense matrices, but the user has to select this type explicitly. R [3] and NumPy [2] 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 and ML algorithms. However, LA properties in most of these systems remain unexploited, which makes them *miss opportunities to use their own highly efficient operators* (recall Hyb_1 in Section 8.2). Our experiments (Section 8) show that LA pipeline evaluation in these systems can be sped up, often by more 10×, by our rewriting using (*i*) LA properties and (*ii*) materialized views.

Bridging the Gap: Linear and Relational Algebra. There has been a recent increase in research for unifying the execution of relational and linear algebra queries /pipelines [1, 27, 32, 35, 37, 39, 45]. A key limitation of these works is that *the semantics of linear algebra operations remains hidden* behind built-in functions and/or UDFs, preventing performance-enhancing rewrites. Some of these works call LA packages through UDFs, where libraries such as R and NumPy are embedded in the host language [1]. Other works treat LA objects as first-class citizens and use built-in functions to express LA operations [28, 32, 37]. Closer to our work, LARA [35] relies on a declarative domain-specific language for collections and matrices, which can enable optimization across the two algebraic abstractions. SPORES [46] and SPOOF [21] optimize LA expressions, by converting them into RA, optimizing the latter, and then converting the result back to an (optimized) LA expression. SPORES and SPOOF are restricted to a small set of selected LA operations (the ones that can be expressed in relational algebra), while we support significantly more (Section 5.1), and model properties allowing to optimize with them. Further, as they do not reason with constraints, they cannot exploit materialized views in an LA (or hybrid LA/RA) context; as shown in our experiments, such rewritings can bring large performance advantages. Our work can also complementing the optimizations of LARA, SPORES or SPOOF, to extend to these platforms the benefits of views-based rewriting.

Conclusion. HADAD is an extensible lightweight approach for optimizing hybrid complex analytics queries, based on the powerful intermediate abstraction of a *a relational model with integrity constraints*. HADAD extends [14] with a reduction from 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 performance gains of up to several orders of magnitude on LA and hybrid workloads. Future work includes reasoning about cell-wise operations, building upon the FAQ [13] framework.

REFERENCES

[1] Embedded python/numpy in monetdb. https://www.monetdb.org/blog/embedded-pythonnumpy-monetdb.
[2] NumPy. https://numpy.org/.
[3] R. https://www.r-project.org/other-docs.html.
[4] Technical report. https://github.com/hadad-paper/HADAD_SIGMOD2021.
[5] Kaggle Survey. https://www.kaggle.com/kaggle-survey-2019, 2019.
[6] Amazon Review Data (2018).https://nijianmo.github.io/amazon/index.html, 2020.
[7] Breeze Wiki.https://github.com/scalanlp/breeze/wiki, 2020.
[8] Netflix movie rating dataset.https://www.kaggle.com/netflix-inc/netflix-prize-data, 2020.
[9] Sparkmllib.https://spark.apache.org/mllib, 2020.
[10] Using Native Blas in SystemDS. https://apache.github.io/systemds/native-backend, 2020.
[11] 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 *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.
[12] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[13] M. Abo Khamis, H. Q. Ngo, and A. Rudra. Faq: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 13–28, 2016.
[14] R. Alotaibi, D. Bursztyn, A. Deutsch, I. Manolescu, and S. Zampetakis. Towards scalable hybrid stores: Constraint-based rewriting to the rescue. In *Proceedings of the 2019 International Conference on Management of Data*, 2019.
[15] R. Alotaibi, B. Cautis, A. Deutsch, M. Latrache, I. Manolescu, and Y. Yang. ESTOCADA: towards scalable polystore systems. *Proc. VLDB Endow.*, 13(12):2949–2952, 2020.
[16] 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. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1383–1394, 2015.
[17] S. Axler. *Linear Algebra Done Right*. Springer, 2015.
[18] 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 tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395, 2017.
[19] 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. *Proceedings of the VLDB Endowment*, 9(13):1425–1436, 2016.
[20] M. Boehm, A. V. Evfimievski, N. Pansare, and B. Reinwald. Declarative machine learning-a classification of basic properties and types. *arXiv preprint arXiv:1605.05826*, 2016.
[21] 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. *Proceedings of the VLDB Endowment*, 11(12), 2018.
[22] 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. *Proceedings of the VLDB Endowment*, 10(12):1694–1705, 2017.
[23] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90, 1977.
[24] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *TOMS*, 2011.
[25] A. Deutsch. Fol modeling of integrity constraints (dependencies).
[26] A. Deutsch, L. Popa, and V. Tannen. Query reformulation with constraints. *ACM SIGMOD Record*, 35(1):65–73, 2006.
[27] J. Duggan et al. The BigDAWG polystore system. In *SIGMOD*, 2015.
[28] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, et al. The madlib analytics library: or mad skills, the sql. *Proceedings of the VLDB Endowment*, 5(12):1700–1711, 2012.
[29] P.-S. Huang, H. Avron, T. N. Sainath, V. Sindhwani, and B. Ramabhadran. Kernel methods match deep neural networks on timit. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 205–209. IEEE, 2014.
[30] I. Ileana. *Query rewriting using views : a theoretical and practical perspective*. Theses, Télécom ParisTech, Oct. 2014.
[31] I. Ileana, B. Cautis, A. Deutsch, and Y. Katsis. Complete yet practical search for minimal query reformulations under constraints. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1015–1026, 2014.
[32] D. Kernert, F. Köhler, and W. Lehner. Bringing linear algebra objects to life in a column-oriented in-memory database. In *In Memory Data Management and Analysis*, pages 44–55. Springer, 2013.
[33] A. Kumar, M. Boehm, and J. Yang. Data management in machine learning: Challenges, techniques, and systems. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1717–1722. ACM, 2017.
[34] A. Kumar, R. McCann, J. Naughton, and J. M. Patel. Model selection management systems: The next frontier of advanced analytics. *ACM SIGMOD Record*, 44(4):17–22, 2016.
[35] A. Kunft, A. Katsifodimos, S. Schelter, S. Breß, T. Rabl, and V. Markl. An intermediate representation for optimizing machine learning pipelines. *Proceedings of the VLDB Endowment*, 12(11):1553–1567, 2019.
[36] K. Kuttler. *Linear algebra: theory and applications*. The Saylor Foundation, 2012.
[37] S. Luo, Z. J. Gao, M. Gubanov, L. L. Perez, and C. Jermaine. Scalable linear algebra on a relational database system. *IEEE Transactions on Knowledge and Data Engineering*, 31(7):1224–1238, 2018.
[38] C. Manning and D. Klein. Optimization, maxent models, and conditional estimation without magic. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Tutorials-Volume 5*, pages 8–8, 2003.
[39] 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*, 17(1):1235–1241, 2016.
[40] M. Milani, S. Hosseinpour, and H. Pehlivan. Rule-based production of mathematical expressions. *Mathematics*, 6:254, 11 2018.
[41] 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 *Advances in neural information processing systems*, pages 2503–2511, 2015.
[42] J. Sommer, M. Boehm, A. V. Evfimievski, B. Reinwald, and P. J. Haas. Mnc: Structure-exploiting sparsity estimation for matrix expressions. In *Proceedings of*

the 2019 International Conference on Management of Data, pages 1607–1623, 2019.
[43] 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 *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 368–380, 2015.
[44] A. Thomas and A. Kumar. A comparative evaluation of systems for scalable linear algebra-based analytics. *Proceedings of the VLDB Endowment*, 11(13):2168–2182, 2018.
[45] J. Wang, T. Baker, M. Balazinska, D. Halperin, B. Haynes, B. Howe, D. Hutchison, S. Jain, R. Maas, P. Mehta, et al. The myria big data management and analytics system and cloud services.
[46] 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. *CoRR*, 2020.

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

Table 7: L_{ops} Operations Properties (LA_{prop}) Captured as Integrity Constraints

LA Property	Relational Encoding as Integrity Constraints
Addition of Matrices	
$M+N=N+M$	$\forall M_1^i, M_2^i, R^o add_M(M_1^i, M_2^i, R^o) \rightarrow add_M(M_2^i, M_1^i, R^o)$
$(M+N)+D=M+(N+D)$	$\forall M_1^i, N_1^i, D_1^i, R_1^o, R_2^o\ add_M(M_1^i, N_1^i, R_1^o) \wedge add_M(R_1^o, D_1^i, R_2^o) \rightarrow$ $\exists R_3^o add_M(N_1^i, D_1^i, R_3^o) \wedge add_M(M_1^i, R_3^o, R_2^o)$
$c(M+N)=cM+cN$	$\forall c, M_1^i, N_1^i, R_1^o, R_2^o\ add_M(M_1^i, N_1^i, R_1^o) \wedge multi_{MS}(c, R_1^o, R_2^o) \rightarrow$ $\exists R_3^o, R_4^o multi_{MS}(c, M_1^i, R_3^o) \wedge multi_{MS}(c, N_1^i, R_4^o) \wedge add_M(R_3^o, R_4^o, R_2^o)$
$(c+d)M=cM+dM$	$\forall c, d, s, M_1^i, R_1^o add_S(c, d, s) \wedge multi_{MS}(s, M_1^i, R_1^o) \rightarrow$ $\exists R_2^o, R_3^o multi_{MS}(c, M_1^i, R_2^o) \wedge multi_{MS}(d, M_1^i, R_3^o) \wedge add_M(R_2^o, R_3^o, R_1^o)$
$M+0=M$	$\rightarrow \exists Zero(O_1^i)$ $\forall M_1^i, n, O_1^i name(M_1^i, n), Zero(O_1^i) \rightarrow add_M(M_1^i, O_1^i, M_1^i)$ $\forall O_1^i Zero(O_1^i) \rightarrow add_M(O_1^i, O_1^i, O_1^i)$
Product of Matrices	
$(MN)D=M(ND)$	$\forall M_1^i, N_1^i, D_1^i, R_1^o, R_2^o multi_M(M_1^i, N_1^i, R_1^o) \wedge multi_M(R_1^o, D_1^i, R_2^o) \rightarrow$ $\exists R_3^o multi_M(N_1^i, D_1^i, R_3^o) \wedge multi_M(M_1^i, R_3^o, R_2^o)$
$M(N+D)=MN+MD$	$\forall M_1^i, N_1^i, D_1^i, R_1^o, R_2^o add_M(N_1^i, D_1^i, R_1^o) \wedge multi_M(M_1^i, R_1^o, R_2^o) \rightarrow$ $\exists R_3^o, R_4^o multi_M(M_1^i, N_1^i, R_3^o) \wedge multi_M(M_1^i, D_1^i, R_4^o) \wedge add_M(R_3^o, R_4^o, R_2^o)$
$(M+N)D=MD+MD$	$\forall M_1^i, N_1^i, D_1^i, R_1^o, R_2^o add_M(M_1^i, N_1^i, R_1^o) \wedge multi_M(R_1^o, D_1^i, R_2^o) \rightarrow$ $\exists R_3^o, R_4^o multi_M(M_1^i, D_1^i, R_3^o) \wedge multi_M(N_1^i, D_1^i, R_4^o) \wedge add_M(R_3^o, R_4^o, R_2^o)$
$d(MN)=(dM)N$	$\forall d, M_1^i, N_1^i, R_1^o, R_2^o multi_M(M_1^i, N_1^i, R_1^o) \wedge multi_{MS}(d, R_1^o, R_2^o) \rightarrow$ $\exists R_3^o multi_{MS}(d, M_1^i, R_3^o) \wedge multi_M(R_3^o, N_1^i, R_2^o)$
$c(dM)=(cd)M$	$\forall c, dM_1^i, R_1^o, R_2^o multi_{MS}(d, M_1^i, R_1^o) \wedge multi_{MS}(c, R_1^o, R_2^o) \rightarrow$ $\exists s\ multi_S(c, d, s) \wedge multi_{MS}(s, M_1^i, R_2^o)$
$I_k M = M = M I_z$	$\forall M_1^i, n, k, z\ name(M_1^i, n), size(M_1^i, k, z) \rightarrow \exists I_1^i\ Identity(I_1^i), size(I_1^i, k, k)$ $\forall M_1^i, n, k, z\ name(M_1^i, n), size(M_1^i, k, z) \rightarrow \exists I_1^i\ Identity(I_1^i), size(I_1^i, z, z)$ $\forall M_1^i, I_1^i, n, k, z\ name(M_1^i, n), size(M_1^i, k, z), Identity(I_1^i), size(I_1^i, k, k) \rightarrow multi_M(I_1^i, M_1^i, M_1^i)$ $\forall M_1^i, I_1^i, n, k, z\ name(M_1^i, n), size(M_1^i, k, z), Identity(I_1^i), size(I_1^i, z, z) \rightarrow multi_M(M_1^i, I_1^i, M_1^i)$
Transposition of Matrices	
$(MN)^T = (N)^T(M)^T$	$\forall M_1^i, N_1^i, R_1^o, R_2^o multi_M(M_1^i, N_1^i, R_1^o) \wedge tr(R_1^o, R_2^o) \rightarrow$ $\exists R_3^o, R_4^o tr(M_1^i, R_3^o) \wedge tr(N_1^i, R_4^o) \wedge multi_M(R_4^o, R_3^o, R_2^o)$
$(M+N)^T = (M)^T + (N)^T$	$\forall M_1^i, N_1^i, R_1^o, R_2^o add_M(M_1^i, N_1^i, R_1^o) \wedge tr(R_1^o, R_2^o) \rightarrow$ $\exists R_3^o, R_4^o tr(M_1^i, R_3^o) \wedge tr(N_1^i, R_4^o) \wedge add_M(R_3^o, R_4^o, R_2^o)$
$(cM)^T = c(M)^T$	$\forall c, M_1^i, R_1^o, R_2^o multi_{MS}(c, M_1^i, R_1^o) \wedge tr(R_1^o, R_2^o) \rightarrow$ $\exists R_3^o tr(M_1^i, R_3^o) \wedge multi_{MS}(c, R_3^o, R_2^o)$
$((M)^T)^T = M$	$\forall n, M_1^i name(M_1^i, n) \rightarrow \exists R_1^o tr(M_1^i, R_1^o) \wedge tr(R_1^o, M_1^i)$
$(I)^T = I$, where I is identity matrix $(O)^T = O$, where O is zero matrix	$\forall I_1^i Identity(I_1^i) \rightarrow tr(I_1^i, I_1^i)$ $\forall O_1^i Zero(O_1^i) \rightarrow tr(O_1^i, O_1^i)$
Inverses of Matrices	
$((M)^{-1})^{-1} = M$	$\forall n, M_1^i name(M_1^i, n) \rightarrow \exists R_1^o inv_M(M_1^i, R_1^o) \wedge inv_M(R_1^o, M_1^i)$
$(MN)^{-1} = (N)^{-1}(M)^{-1}$	$\forall M_1^i, N_1^i, R_1^o, R_2^o multi_M(M_1^i, N_1^i, R_1^o) \wedge inv_M(R_1^o, R_2^o) \rightarrow$ $\exists R_3^o, R_4^o inv_M(M_1^i, R_3^o) \wedge inv_M(N_1^i, R_4^o) \wedge multi_M(R_4^o, R_3^o, R_2^o)$
$((M)^T)^{-1} = ((M)^{-1})^T$	$\forall M_1^i, R_1^o, R_2^o tr(M_1^i, R_1^o) \wedge inv_M(R_1^o, R_2^o) \rightarrow$ $\exists R_3^o inv_m(M_1^i, R_3^o) \wedge tr(R_3^o, R_2^o)$
$((kM))^{-1} = k^{-1}(M)^{-1}$	$\forall k, M_1^i, R_1^o, R_2^o multi_{MS}(k, M_1^i, R_1^o) \wedge inv_M(R_1^o, R_2^o) \rightarrow$ $\exists R_3^o, s\ inv_S(k, s) \wedge inv_m(M_1^i, R_3^o) \wedge multi_{MS}(s, R_3^o, R_2^o)$
$M^{-1}M = I = MM^{-1}$	$\forall M_1^i, R_1^o, R_2^o\ inv_M(M_1^i, R_1^o) \wedge multi_M(R_1^o, M_1^i, R_2^o) \rightarrow Identity(R_2^o)$ $\forall M_1^i, R_1^o, R_2^o\ inv_M(M_1^i, R_1^o) \wedge multi_M(M_1^i, R_1^o, R_2^o) \rightarrow Identity(R_2^o)$

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

LA Property	Relational Encoding as Integrity Constraints
Determinant of Matrices	
$det(MN) = det(M) * det(N)$	$\forall M_1^i, N_1^i, R_1^o, d\ multi_M(M_1^i, N_1^i, R_1^o) \wedge det(R_1^o, d) \rightarrow$ $\exists d_1, d_2 det(M_1^i, d_1) \wedge det(N_1^i, d_2) \wedge multi_S(d_1, d_2, d)$
$det((M)^T) = det(M)$	$\forall M_1^i, R_1^o, d\ tr(M_1^i, R_1^o) \wedge det(R_1^o, d) \rightarrow det(M_1^i, d)$
$det((M)^{-1}) = (det(M))^{-1}$	$\forall M_1^i, R_1^o, d\ inv_M(M_1^i, R_1^o) \wedge det(R_1^o, d) \rightarrow \exists d_1 det(M_1^i, d_1) \wedge inv_S(d_1, d)$
$det((cM)) = c^k det(M)$	$\forall M_1^i, c, k, d^O\ size(M_1^i, k, k) \wedge multi_{MS}(c, M_1^i, d^O) \rightarrow \exists s_1, s_2$ $pow(c, k, s_1) \wedge det(M_1^i, s_2) \wedge multi_S(s_1, s_2, d^O)$
$det((I)) = 1$	$\forall I_1^i, d^O Identity(I_1^i) \wedge det(I_1^i, d^O) \rightarrow d^O = 1$
Adjoint of Matrices	
$adj(M) = R^T$	$\forall M_1^i, R_1^o\ adj(M_1^i, R_1^o) \rightarrow \exists R_2^o\ cof(M_1^i, R_2^o) \wedge tr(R_2^o, R_1^o)$
$adj(M)^T = adj(M^T)$	$\forall M_1^i, R_1^o, R_2^o\ adj(M_1^i, R_1^o) \wedge tr(R_1^o, R_2^o) \rightarrow \exists R_3^o\ tr(M_1^i, R_3^o) \wedge adj(R_3^o, R_2^o)$
$adj(M)^{-1} = adj(M^{-1})$	$\forall M_1^i, R_1^o, R_2^o\ adj(M_1^i, R_1^o) \wedge inv_M(R_1^o, R_2^o) \rightarrow \exists R_3^o\ inv_M(M_1^i, R_3^o) \wedge adj(R_3^o, R_2^o)$
$adj(MN) = adj(NM)$	$\forall M_1^i, R_1^o, R_2^o\ mult_M(M_1^i, N_1^i, R_1^o) \wedge adj(R_1^o, R_2^o) \rightarrow \exists R_3^o\ mult_M(N_1^i, M_1^i, R_3^o) \wedge adj(R_3^o, R_2^o)$
Trace of Matrices	
$trace(M + N) = trace(M) + trace(N)$	$\forall M_1^i, N_1^i, R_1^o, s_1^O\ add_M(M_1^i, N_1^i, R_1^o) \wedge$ $trace(R_1^o, s_1^O) \rightarrow \exists s_2^O, s_3^O\ trace(M_1^i, s_2^O) \wedge trace(N_1^i, s_3^O) \wedge add_s(s_2^O, s_3^O, s_1^O)$
$trace(MN) = trace(NM)$	$\forall M_1^i, N_1^i, R_1^o, s_1^O\ multi_M(M_1^i, N_1^i, R_1^o) \wedge$ $trace(R_1^o, s_1^O) \rightarrow \exists R_2^o\ multi_M(N_1^i, M_1^i, R_2^o) \wedge trace(R_2^o, s_1^O)$
$trace(M^T) = trace(M)$	$\forall M_1^i, R_1^o, s_1^O\ tr(M_1^i, R_1^o) \wedge trace(R_1^o, s_1^O) \rightarrow trace(M_1^i, s_1^O)$
$trace(cM) = ctrace(M)$	$\forall M_1^i, R_1^o, c, s_1^O\ multi_{MS}(c, M_1^i, R_1^o) \wedge trace(R_1^o, s_1^O) \rightarrow \exists s_2^O trace(M_1^i, s_2^O) \wedge multi_s(c, s_1^O, s_2^O)$
$trace(I_k) = k$	$\forall I_O^i, k, s_1^O\ Identity(I_O^i) \wedge size(I_O^i, k, k) \wedge trace(I_O^i), s_1^O) \rightarrow s_1^O = k$
Direct Sum	
$(M \oplus N) + (C \oplus D) = (M + C) \oplus (N + D)$	$\forall M_1^i, N_1^i, R_1^o, C_1^i, D_1^i, R_2^o, R_3^o\ sum_D(M_1^i, N_1^i, R_1^o)$ $\wedge sum_D(C_1^i, D_1^i, R_2^o) \wedge add_m(R_1^o, R_2^o, R_3^o) \rightarrow \exists R_4^o, R_5^o add_m(M_1^i, C_1^i, R_4^o) \wedge add_m(N_1^i, D_1^i, R_5^o)$
$(M \oplus N) + (C \oplus D) = (MC) \oplus (ND)$	$\forall M_1^i, N_1^i, R_1^o, C_1^i, D_1^i, R_2^o, R_3^o\ sum_D(M_1^i, N_1^i, R_1^o)$ $\wedge sum_D(C_1^i, D_1^i, R_2^o) \wedge multi_m(R_1^o, R_2^o, R_3^o) \rightarrow \exists R_4^o, R_5^o multi_m(M_1^i, C_1^i, R_4^o) \wedge multi_m(N_1^i, D_1^i, R_5^o)$
$c(M \oplus N) = (cM \oplus cN)$	$\forall M_1^i, N_1^i, R_1^o, c, R_2^o\ sum_D(M_1^i, N_1^i, R_1^o)$ $\wedge multi_{MS}(c, R_1^o, R_2^o) \rightarrow \exists R_3^o, R_4^o multi_{MS}(M_1^i, c, R_3^o) \wedge multi_{MS}(N_1^i, c, R_4^o) \wedge sum_D(R_3^o, R_4^o, R_2^o)$
Exponential of Matrices	
$exp(0) = I$	$\forall O_1^i, R_1^o exp(O_1^i), R_1^o) \rightarrow Identity(R_1^o)$
$exp(M^T) = exp(M)^T$	$\forall M_1^i, R_1^o, R_2^o tr(M_1^i, R_1^o) \wedge exp(R_1^o, R_2^o) \rightarrow \exists R_3^o exp(M_1^i, R_3^o) \wedge exp(R_3^o, R_2^o)$

Table 9: Matrix Decompositions Properties Captured as Integrity Constraints

Decomposition Property	**Relational Encoding as Integrity Constraints**
Cholesky Decomposition (CD)	
$cho(M) = L$ such that $M = LL^T$, where M is symmetric positive definite	$\forall M_1^i\ type(M_1^i, \text{“S”}) \rightarrow \exists\ L_1^o \exists L_2^o\ cho(M_1^i, L_1^o) \wedge type(L_1^o, \text{“L”}) \wedge$ $tr(L_1^o, L_2^o) \wedge multi_M(L_1^o, L_2^o, M_1^i)$
QR Decomposition	
$QR(M) = [Q, R]$ such that $M = QR$	$\forall M_1^i \forall n \forall k\ name(M_1^i, n) \wedge size(M_1^i, k, z) \rightarrow\ \exists Q^o, R^o$ $QR(M_1^i, Q_1^o, R_1^o) \wedge type(Q_1^o, \text{“O”}) \wedge size(Q_1^o, k, k) \wedge type(R_1^o, \text{“U”}) \wedge$ $size(R_1^o, k, z) \wedge multi_M(Q^o, R^o, M_1^i)$ $\forall Q_1^i\ type(Q_1^i, \text{“O”}) \wedge size(Q_1^i, k, k) \rightarrow \exists I_1^o\ QR(Q_1^i, Q_1^i, I_1^o) \wedge identity(I_1^o)$ $\wedge size(I_1^o, k, k) \wedge multi_M(Q_1^i, I_1^o, Q_1^i)$ $\forall R_1^i\ type(R_1^i, \text{“U”}) \wedge size(Q_1^i, k, z) \rightarrow \exists I_1^o\ QR(R_1^i, I_1^o, R_1^i) \wedge identity(I_1^o)$ $\wedge size(I_1^o, k, k) \wedge multi_M(I_1^o, R_1^i, Q_1^i)$ $\forall I_1^i\ identity(I_1^i) \rightarrow QR(I_1^i, I_1^i, I_1^i)$
LU Decomposition	
$LU(M) = [L, U]$ such that $M = LU$	$\forall M_1^i \forall n \forall k\ name(M_1^i, n) \wedge size(M_1^i, k, z) \rightarrow\ \exists L_1^o, U_1^o$ $LU(M_1^i, L_1^o, U_1^o) \wedge type(L_1^o, \text{“L”}) \wedge size(L_1^o, k, z) \wedge type(U_1^o, \text{“U”}) \wedge$ $size(U_1^o, z, z) \wedge multi_M(L_1^o, U_1^o, M_1^i)$ $\forall L_1^i\ type(L_1^i, \text{“L”}) \wedge size(L_1^i, k, z) \rightarrow \exists I_1^o\ LU(L_1^i, L_1^i, I_1^o) \wedge identity(I_1^o)$ $\wedge multi_M(L_1^i, I_1^o, L_1^i) \wedge size(I_1^o, z, z)$ $\forall U_1^i\ type(U_1^i, \text{“U”}) \wedge size(U_1^i z, z) \rightarrow \exists I_1^o\ LU(U_1^i, I_1^o, U_1^i) \wedge identity(I_1^o)$ $\wedge size(I_1^o z, z) \wedge multi_M(I_1^o, U_1^o, U_1^o)$ $\forall I_1^i\ identity(I_1^i) \rightarrow LU(I_1^i, I_1^i, I_1^i)$
Pivoted LU Decomposition	
$LU(M) = [L, U, P]$ such that $PM = LU$, where M is a square matrix	$\forall M_1^i \forall n \forall k\ name(M_1^i, n) \wedge size(M_1^i, k, z) \rightarrow\ \exists L_1^o, U_1^o, P_1^o, R_1^o$ $LUP(M_1^i, L_1^o, U_1^o, P_1^o) \wedge type(L_1^o, \text{“L”}) \wedge type(U_1^o, \text{“U”}) \wedge type(P_1^o, \text{“P”})$ $\wedge multi_M(L_1^o, U_1^o, R_1^o) \wedge multi_M(P_1^o, M_1^i, R_1^o)$ $\forall L_1^i\ type(L_1^i, \text{“L”}) \wedge size(L_1^i, k, z) \rightarrow \exists I_1^o, I_2^o\ LUP(L_1^i, L_1^i, I_1^o, I_2^o) \wedge$ $identity(I_1^o) \wedge identity(I_2^o) \wedge size(I_1^o, z, z) \wedge size(I_2^o, k, k)$ $\wedge multi_M(L_1^i, I_1^o, L_1^i) \wedge multi_M(I_2^o, L_1^i, L_1^i)$ $\forall U_1^i\ type(U_1^i, \text{“U”}) \rightarrow \exists I_1^o\ LUP(U_1^i, I_1^o, U_1^i, I_1^o) \wedge$ $identity(I_1^o) \wedge multi_M(I_1^o, U_1^i, U_1^i)$ $\forall I_1^i\ identity(I_1^i) \rightarrow LU(I_1^i, I_1^i, I_1^i)$

B SYSTEMML REWRITE RULES ENCODED AS INTEGRITY CONSTRAINTS

Table 10: SystemML Algebraic Aggregate Rewrite Rules Captured as Integrity Constraints

SystemML Algebraic Simplification Rule	Integrity Constraints $\mathcal{MMC}_{StatAgg}$
UnnecessaryAggregates	
sum(t(M))-> sum(M)	$\forall M^i_1, R^o_1, s\ tr(M^i_1, R^o_1), sum(R^o_1, s) \rightarrow sum(M^i_1, s)$
sum(rev(M))-> sum(M)	$\forall M^i_1, R^o_1, s\ rev(M^i_1, R^o_1), sum(R^o_1, s) \rightarrow sum(M^i_1, s)$
sum(rowSums(M))-> sum(M)	$\forall M^i_1, R^o_1, s\ rowSums(M^i_1, R^o_1), sum(R^o_1, s) \rightarrow sum(M^i_1, s)$
sum(colSums(M))-> sum(M)	$\forall M^i_1, R^o_1, s\ colSums(M^i_1, R^o_1), sum(R^o_1, s) \rightarrow sum(M^i_1, s)$
min(rowMins(M))-> min(M)	$\forall M^i_1, R^o_1, s\ rowMins(M^i_1, R^o_1), min(R^o_1, s) \rightarrow min(M^i_1, s)$
min(colMins(M))-> min(M)	$\forall M^i_1, R^o_1, s\ colMins(M^i_1, R^o_1), min(R^o_1, s) \rightarrow min(M^i_1, s)$
max(colMax(M))-> max(M)	$\forall M^i_1, R^o_1, s\ colMax(M^i_1, R^o_1), max(R^o_1, s) \rightarrow max(M^i_1, s)$
max(rowMax(M))-> max(M)	$\forall M^i_1, R^o_1, s\ rowMax(M^i_1, R^o_1), max(R^o_1, s) \rightarrow max(M^i_1, s)$
pushdownUnaryAggTransposeOp	
rowSums(t(M))->t(colSums(M))	$\forall M^i_1, R^o_1, R^o_2\ tr(M^i_1, R^o_1) \wedge rowSums(R^o_1, R^o_2) \rightarrow \exists R^o_3\ colSums(M^i_1, R^o_3) \wedge tr(R^o_3, R^o_2)$
colSums(t(M))->t(rowSums(M))	$\forall M^i_1, R^o_1, R^o_2\ tr(M^i_1, R^o_1) \wedge colSums(R^o_1, R^o_2) \rightarrow \exists R^o_3\ rowSums(M^i_1, R^o_3) \wedge tr(R^o_3, R^o_2)$
rowMeans(t(M))->t(colMeans(M))	$\forall M^i_1, R^o_1, R^o_2\ tr(M^i_1, R^o_1) \wedge colMeans(R^o_1, R^o_2) \rightarrow \exists R^o_3\ rowMeans(M^i_1, R^o_3) \wedge tr(R^o_3, R^o_2)$
colMeans(t(M))->t(rowMeans(M))	$\forall M^i_1, R^o_1, R^o_2\ tr(M^i_1, R^o_1) \wedge rowMeans(R^o_1, R^o_2) \rightarrow \exists R^o_3\ colMeans(M^i_1, R^o_3 \wedge tr(R^o_3, R^o_2)$
rowVars(t(M))->t(colVars(M))	$\forall M^i_1, R^o_1, R^o_2\ tr(M^i_1, R^o_1) \wedge rowVars(R^o_1, R^o_2) \rightarrow \exists R^o_3\ colVars(M^i_1, R^o_3) \wedge tr(R^o_3, R^o_2)$
colVars(t(X))->t(rowVars(X))	$\forall M^i_1, R^o_1, R^o_2\ tr(M^i_1, R^o_1) \wedge colVars(R^o_1, R^o_2) \rightarrow \exists R^o_3\ rowVars(M^i_1, R^o_3) \wedge tr(R^o_3, R^o_2)$
rowMaxs(t(M))->t(colMaxs(M))	$\forall M^i_1, R^o_1, R^o_2\ tr(M^i_1, R^o_1) \wedge rowMaxs(R^o_1, R^o_2) \rightarrow \exists R^o_3\ colMaxs(M^i_1, R^o_3) \wedge tr(R^o_3, R^o_2)$
colMaxs(t(M))->t(rowMaxs(M))	$\forall M^i_1, R^o_1, R^o_2\ tr(M^i_1, R^o_1) \wedge colMaxs(R^o_1, R^o_2) \rightarrow \exists R^o_3\ rowMaxs(M^i_1, R^o_3) \wedge tr(R^o_3, R^o_2)$
rowMins(t(M))->t(colMins(M))	$\forall M^i_1, R^o_1, R^o_2\ tr(M^i_1, R^o_1) \wedge rowMins(R^o_1, R^o_2) \rightarrow \exists R^o_3\ colMins(M^i_1, R^o_3) \wedge tr(R^o_3, R^o_2)$
colMins(t(M))->t(rowMins(M))	$\forall M^i_1, R^o_1, R^o_2\ tr(M^i_1, R^o_1) \wedge colMins(R^o_1, R^o_2) \rightarrow \exists R^o_3\ rowMins(M^i_1, R^o_3) \wedge tr(R^o_3, R^o_2)$
simplifyTraceMatrixMult	
trace(MN)->sum(M⊙t(N))	$\forall M^i_1, N^i_1, R^o_1, r\ multi(M^i_1, N^i_1, R^o_1) \wedge trace(R^o_1, r) \rightarrow$ $\exists R^o_3, R^o_4 tr(N^i_1, R^o_3) \wedge multi_E(M^i_1, R^o_3, R^o_4) \wedge sum(R^o_4, r)$
simplifySumMatrixMult	
sum(MN) -> sum(t(colSums(M))⊙rowSums(N))	$\forall M^i_1, N^i_1, R^o_1, r\ mult(M^i_1, N^i_1, R^o_1) \wedge sum(R^o_1, r) \rightarrow$ $\exists R^o_2, R^o_3, R^o_4, R^o_5\ colSums(M^i_1, R^o_2) \wedge tr(R^o_2, R^o_3) \wedge rowSums(N^i_1, R^o_4) \wedge$ $multi_E(R^o_3, R^o_4, R^o_5), \wedge sum(R^o_5, r)$
colSums(MN) -> colSums(M)N	$\forall M^i_1, N^i_1, R^o_1, R^o_2\ multi(M^i_1, N^i_1, R^o_1) \wedge colSums(R^o_1, R^o_2) \rightarrow$ $\exists R^o_3\ colSums(M^i_1, R^o_3) \wedge multi(R^o_3, N^i_1, R^o_2)$
rowSums(MN) -> MrowSums(N)	$\forall M^i_1, N^i_1, R^o_1, R^o_2\ multi(M^i_1, N^i_1, R^o_1) \wedge rowSums(R^o_1, R^o_2) \rightarrow$ $\exists R^o_3\ rowSums(N^i_1, R^o_3) \wedge multi(M^i_1, R^o_3, R^o_2)$
simplifyColWiseAgg	
colSums(M)->M if x is row vector	$\forall M^i_1, n, i\ name(M^i_1, n) \wedge size(M^i_1, “1”, j) \rightarrow colSums(M^i_1, M^i_1)$
colMeans(M)->M if x is row vector	$\forall M^i_1, n, j\ name(M^i_1, n) \wedge size(M^i_1, “1”, j) \rightarrow colSums(M^i_1, M^i_1)$
colVars(M)->M if x is row vector	$\forall M^i_1, n, j\ name(M^i_1, n) \wedge size(v“1”, j) \rightarrow colVars(M^i_1, M^i_1)$
colMaxs(M)->M if x is row vector	$\forall M^i_1, n, j\ name(M^i_1, n) \wedge size(M^i_1, “1”, j) \rightarrow colMaxs(M^i_1, M^i_1)$
colMins(M)->M if x is row vector	$\forall M^i_1, n, j\ name(M^i_1, n) \wedge size(M^i_1, “1”, j) \rightarrow colMins(M^i_1, M^i_1)$
colSums(M)->sum(M) if x is col vector	$\forall M^i_1, i, R^o_1\ colSums(M^i_1, R^o_1) \wedge size(M^i_1, i, “1”) \rightarrow sum(M^i_1, R^o_1)$
colMeans(M)->Mean(M) if x is col vector	$\forall M^i_1, i, R^o_1\ colMeans(M^i_1, R^o_1) \wedge size(M^i_1, i, “1”) \rightarrow Mean(M^i_1, R^o_1)$
colMaxs(X)->Max(M) if x is col vector	$\forall M^i_1, i, R^o_1\ colMaxs(M^i_1, R^o_1) \wedge size(M^i_1, i, “1”) \rightarrow Max(M^i_1, R^o_1)$
colMins(M)->Min(X) if x is col vector	$\forall M^i_1, i, R^o_1\ colMins(M^i_1, R^o_1) \wedge size(M^i_1, i, “1”) \rightarrow Min(M^i_1, R^o_1)$
colVars(M)->Var(M) if x is col vector	$\forall M^i_1, i, R^o_1\ colVars(M^i_1, R^o_1) \wedge size(M^i_1, i, “1”) \rightarrow Var(M^i_1, R^o_1)$

SystemML Algebraic Simplification Rule	Integrity Constraints $MMC_{StatAgg}$
simplifyRowWiseAgg	
rowSums(M)->M if x is col vector	$\forall M_1^i, n, i \; name(M_1^i, n) \wedge size(M_1^i, i, \text{“1”}) \rightarrow rowSums(M_1^i, M_1^i)$
rowMeans(M)->M if x is col vector	$\forall M_1^i, n, i \; name(M_1^i, n) \wedge size(M_1^i, i, \text{“1”}) \rightarrow rowMeans(M_1^i, M_1^i)$
rowVars(M)->M if x is col vector	$\forall M_1^i, n, i \; name(M_1^i, n) \wedge size(M_1^i, i, \text{“1”}) \rightarrow rowVars(M_1^i, M_1^i)$
rowMaxs(M)->M if x is col vector	$\forall M_1^i, n, i \; name(M_1^i, n) \wedge size(M_1^i, i, \text{“1”}) \rightarrow rowMaxs(M_1^i, M_1^i)$
rowMins(M)->M if x is col vector	$\forall M_1^i, n, i \; name(M_1^i, n) \wedge size(M_1^i, i, \text{“1”}) \rightarrow rowMaxs(M_1^i, M_1^i)$
rowSums(M)->sum(M) if x is row vector	$\forall M_1^i, j, R_1^o \; rowSums(M_1^i, R_1^o) \wedge size(M_1^i, \text{“1”}, j) \rightarrow sum(M_1^i, R_1^o)$
rowMeans(M)->Mean(M) if x is row vector	$\forall M_1^i, j, R_1^o \; rowMeans(M_1^i, R_1^o) \wedge size(M_1^i, \text{“1”}, j) \rightarrow Mean(M_1^i, R_1^o)$
rowMaxs(M)->Max(M) if x is row vector	$\forall M_1^i, j, R_1^o \; rowMaxs(M_1^i, R_1^o) \wedge size(M_1^i, \text{“1”}, j) \rightarrow Max(M_1^i, R_1^o)$
rowMins(X)->Min(M) if x is row vector	$\forall M_1^i, j, R_1^o \; rowMins(M_1^i, R_1^o) \wedge size(M_1^i, \text{“1”}, j) \rightarrow Min(M_1^i, R_1^o)$
rowVars(X)->Var(M) if x is row vector	$\forall M_1^i, j, R_1^o \; rowVars(M_1^i, R_1^o) \wedge size(M_1^i, \text{“1”}, j) \rightarrow Var(M_1^i, R_1^o)$
pushdownSumOnAdd	
sum(M+N) -> sum(A)+sum(B)	$\forall M_1^i, N_1^i, s \; add_M(M_1^i, N^i, s_1) \wedge sum(M_1^i, s_1) \rightarrow$ $\exists s_2, s_3 \; sum(M_1^i, s_2) \wedge sum(N_1^i, s_3) \wedge \wedge add_s(s_2, s_3, s_1)$
ColSumsMVMult	
colSums(M*N) -> t(M)N	$\forall M_1^i, N_1^i, R_1^o, R_2^o, i \; size(N_1^i, i, \text{“1”}) \wedge multi_E(M_1^i, N_1^i, R_1^o) \wedge colSums(R_1^o, R_2^o)$ $\exists R_3^o tr(M_1^i, R_3^o) \wedge multi(R_3^o, N_1^i, R_2^o)$
rowSums(M*M) -> Mt(N)	$\forall M_1^i, N_1^i, R_1^o, R_2^o, j \; size(N_1^i, \text{“1”}, j) \wedge multi_E(M_1^i, N_1^i, R_1^o) \wedge rowSums(R_1^o, R_2^o)$ $\exists R_3^o tr(N_1^i, R_3^o) \wedge multi(M_1^i, R_3^o, R_2^o)$

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	$s_1 trace(D)$
P1.7	A	P1.8	$(s_1 + s_2)A$	P1.9	$det(D)$
P1.10	$colSums(A)^T$	P1.11	$colSums(A+B)^T$	P1.12	$colSums(M)N$
P1.13	$sum(colSums(M)^T * rowSums(N))$	P1.14	$sum(colSums(M)^T * rowSums(N))$	P1.15	$M(NM)$
P1.16	$sum(A)$	P1.17	$det(C) * det(D) * det(C)$	P1.18	$sum(A)$
P1.25	$M \odot (N^T/(M(NN^T)))$				

Table 11: $\mathcal{P}^{\neg Opt}$ Pipelines (Part 1) Rewrites

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

Table 12: $\mathcal{P}^{\neg Opt}$ Pipelines (Part 2) Rewrites

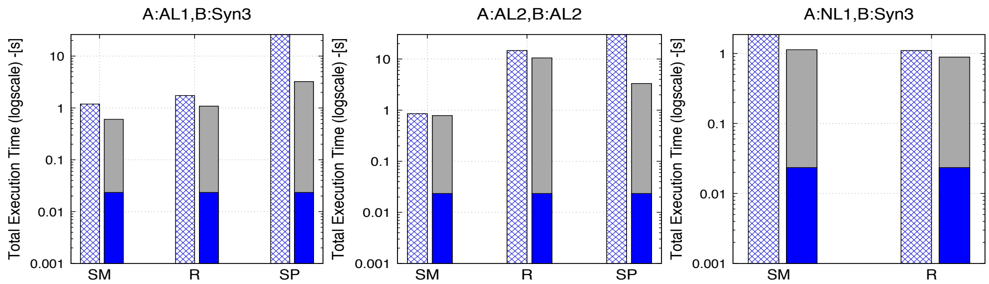
No.	Expression	No.	Expression	No.	Expression
V_1	$(D)^{-1}$	V_2	$(C^T)^{-1}$	V_3	NM
V_4	$u_1 v_2^T$	V_5	DC	V_6	$A+B$
V_7	C^{-1}	V_8	$C^T D$	V_9	$(D+C)^{-1}$
V_{10}	$det(CD)$	V_{11}	$det(DC)$	V_{12}	$(DC)^T$

Table 13: 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	$colSums(V_6)^T$	P1.15	$M(V_3)$	P1.17	$V_{10} * det(C)$
P1.19	V_2	P1.20	$trace(V_7)$	P1.21	$(C+V_1)^T$
P1.22	$trace(V_9)$	P1.24	$trace(V_1 V_7) + trace(D)$	P1.29	$V_5 CCC$
P1.30	$V_3 \odot V_3 R^T$	P2.2	$det(V_1)$	P2.4	$s_1(V6)$
P2.5	$det(V_9)$	P2.6	$(V_1 C)^T$	P2.9	$trace(V_{12}) + trace(D)$
P2.11	$sum(V_6)$	P2.13	$(MV_3)^T$	P2.14	$MV_3 N$
P2.16	$trace(V_7 V_1) + traceD)$	P2.17	$(V_9^T)D$	P2.18	$rowSums(V_6)^T$
P2.20	$(MV_3)^T$	P2.21	$V_1(V_1^T(D^T v_1))$	P2.25	$V_4 v_1 - X v_1$
P1.23	$det((V_7 V_1) + D)$	P2.26	$exp(V_9)$	P2.27	$V_9^T V_5$

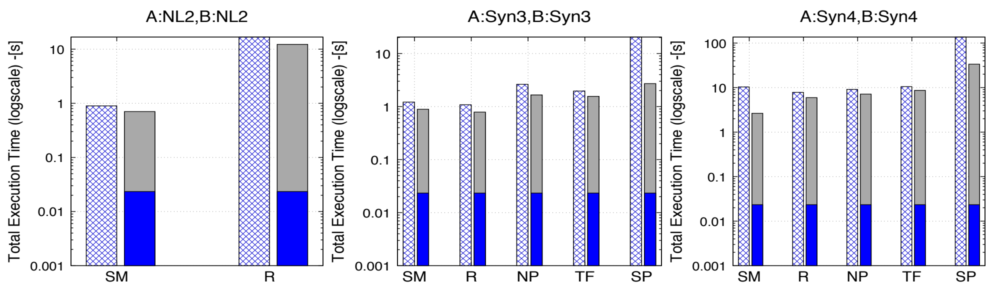
Table 14: $\mathcal{P}^{Views}$ Pipelines Rewrites

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



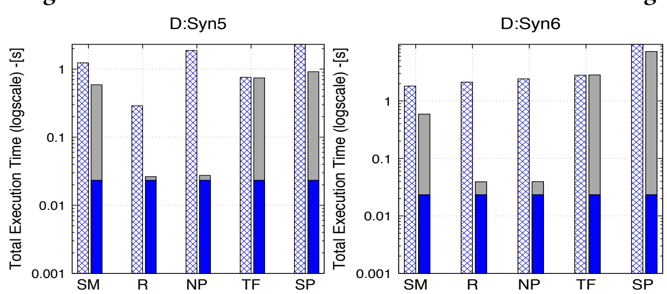
(a) **P1.2** (b) **P1.2** (c) **P1.2**

Figure 11: P1.2 evaluation time with and without rewriting



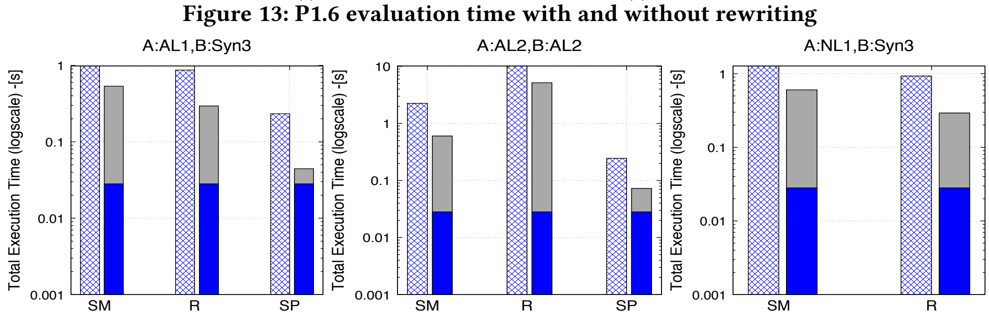
(a) **P1.2** (b) **P1.2** (c) **P1.2**

Figure 12: P1.2 evaluation time with and without rewriting



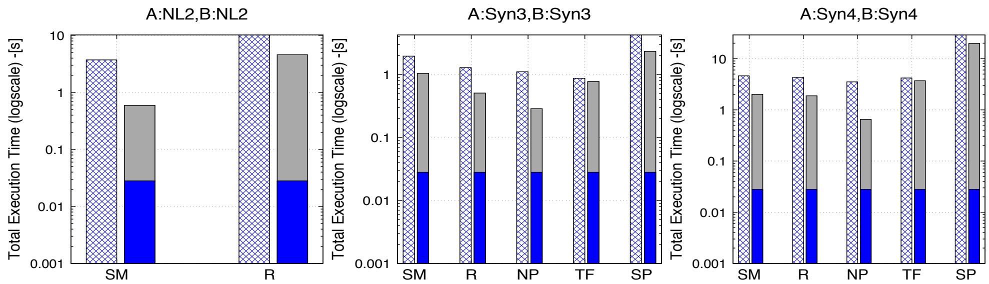
(a) **P1.6** (b) **P1.6**

Figure 13: P1.6 evaluation time with and without rewriting



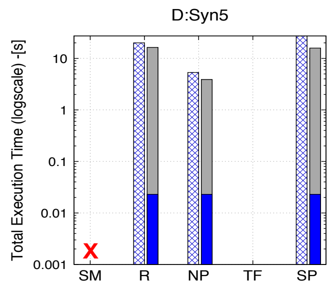
(a) **P1.8** (b) **P1.8** (c) **P1.8**

Figure 14: P1.8 evaluation time with and without rewriting



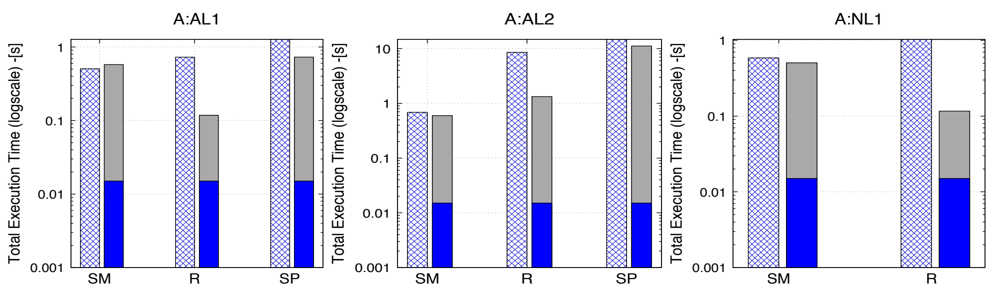
(a) **P1.8** (b) **P1.8** (c) **P1.8**

Figure 15: P1.8 evaluation time with and without rewriting



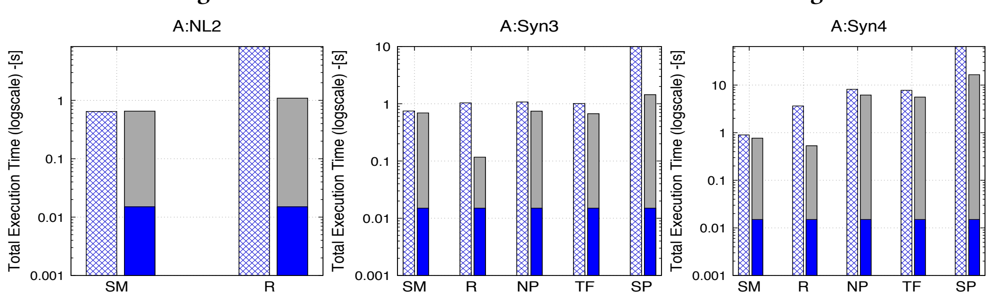
(a) **P1.9**

Figure 16: P1.9 evaluation time with and without rewriting



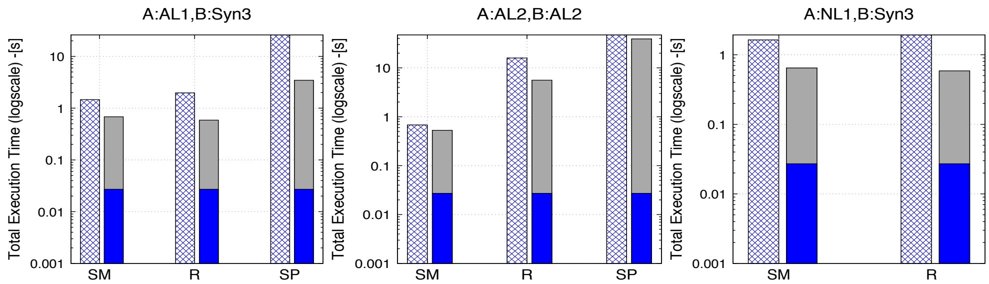
(a) **P1.10** (b) **P1.10** (c) **P1.10**

Figure 17: P1.10 evaluation time with and without rewriting



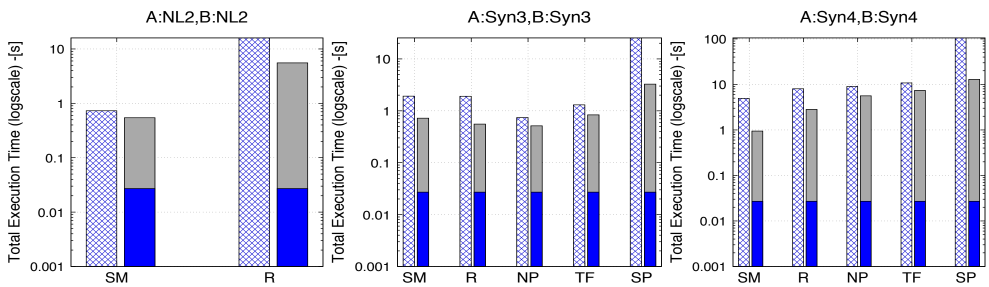
(a) **P1.10** (b) **P1.10** (c) **P1.10**

Figure 18: P1.10 evaluation time with and without rewriting



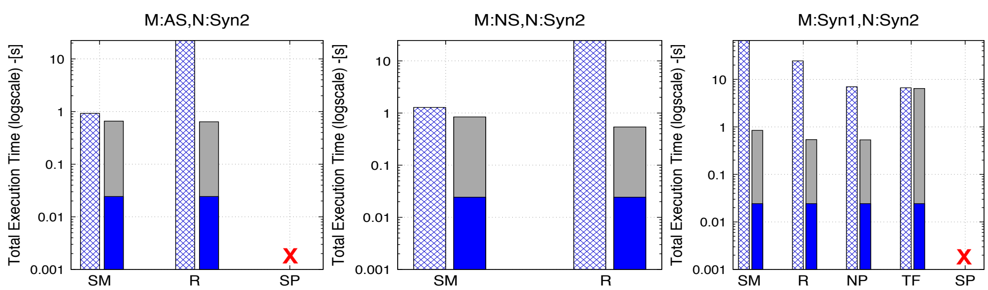
(a) **P1.11** (b) **P1.11** (c) **P1.11**

Figure 19: P1.11 evaluation time with and without rewriting



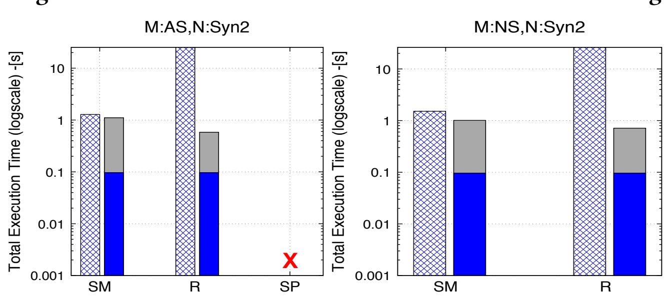
(a) **P1.11** (b) **P1.11** (c) **P1.11**

Figure 20: P1.11 evaluation time with and without rewriting



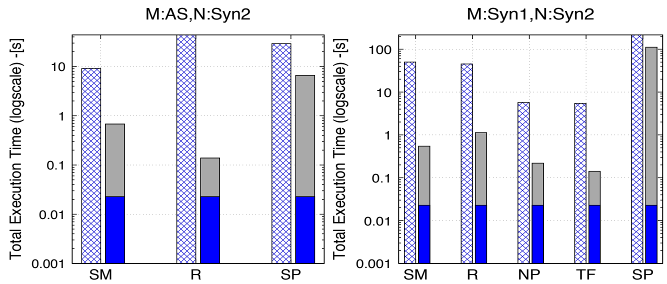
(a) **P1.12** (b) **P1.12** (c) **P1.12**

Figure 21: P1.12 evaluation time with and without rewriting



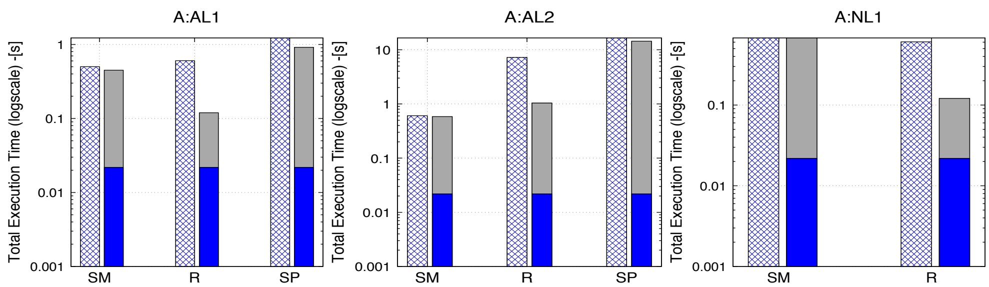
(a) **P1.14** (b) **P1.14**

Figure 22: P1.14 evaluation time with and without rewriting



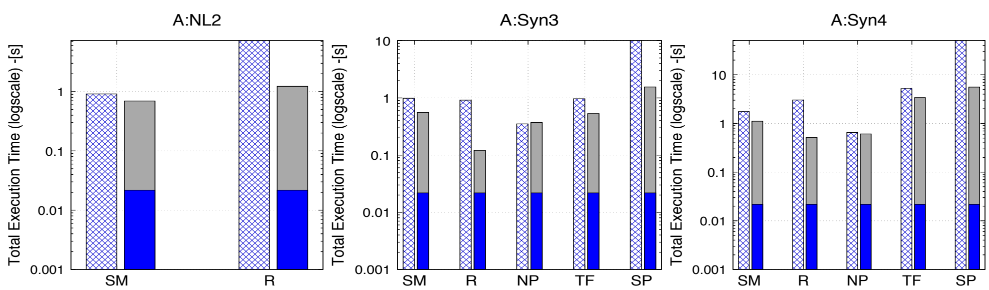
(a) **P1.15** (b) **P1.15**

Figure 23: P1.15 evaluation time with and without rewriting



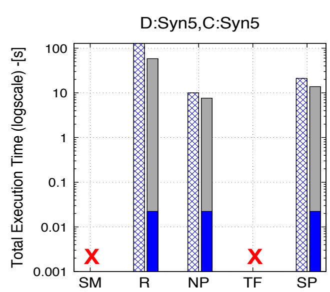
(a) **P1.16** (b) **P1.16** (c) **P1.16**

Figure 24: P1.16 evaluation time with and without rewriting



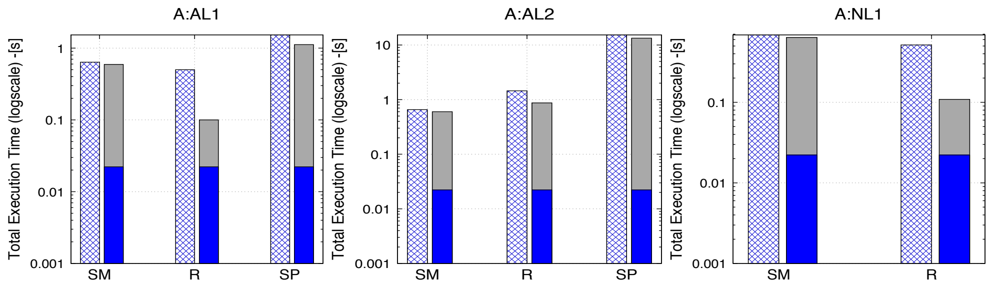
(a) **P1.16** (b) **P1.16** (c) **P1.16**

Figure 25: P1.16 evaluation time with and without rewriting



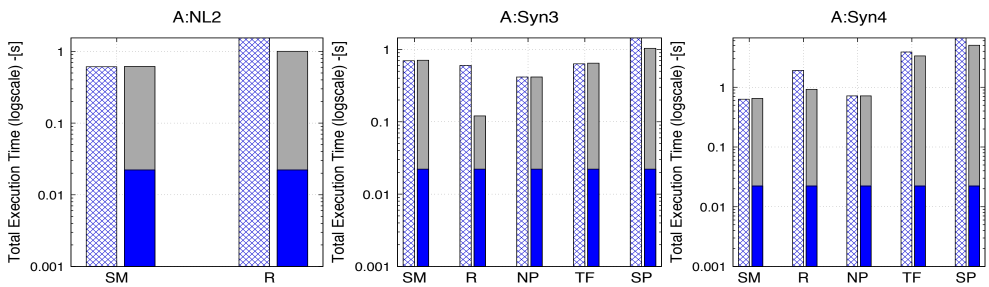
(a) **P1.17**

Figure 26: P1.17 evaluation time with and without rewriting



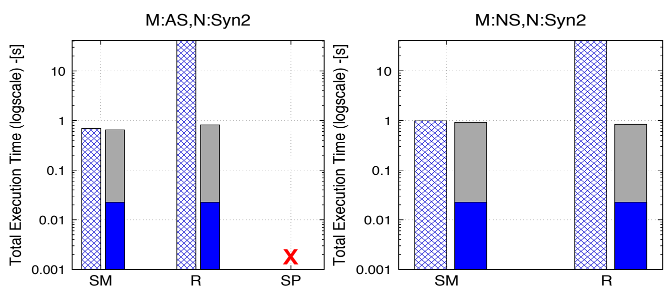
(a) **P1.18** (b) **P1.18** (c) **P1.18**

Figure 27: P1.18 evaluation time with and without rewriting



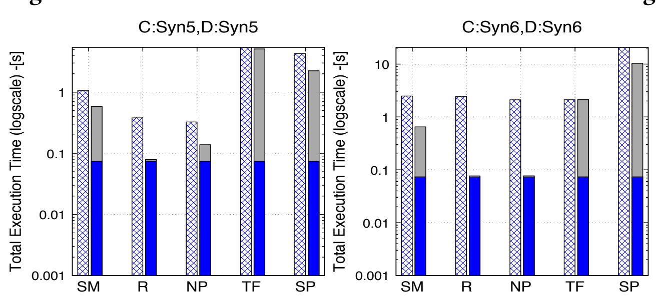
(a) **P1.18** (b) **P1.18** (c) **P1.18**

Figure 28: P1.18 evaluation time with and without rewriting



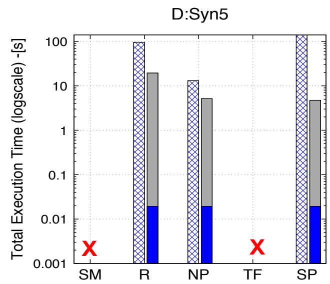
(a) **P1.25** (b) **P1.25**

Figure 29: P1.25 evaluation time with and without rewriting



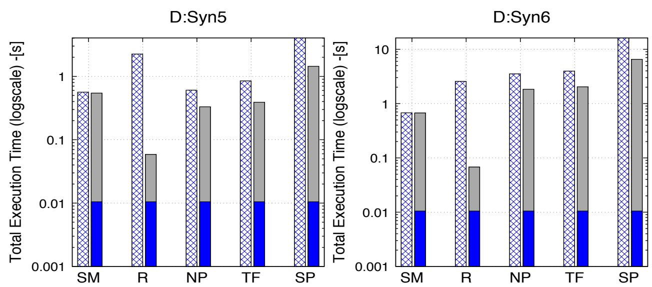
(a) **P2.1** (b) **P2.1**

Figure 30: P2.1 evaluation time with and without rewriting



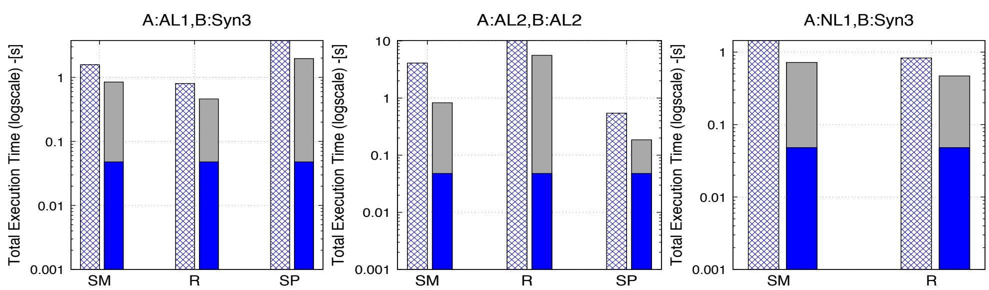
(a) **P2.2**

Figure 31: P2.2 evaluation time with and without rewriting



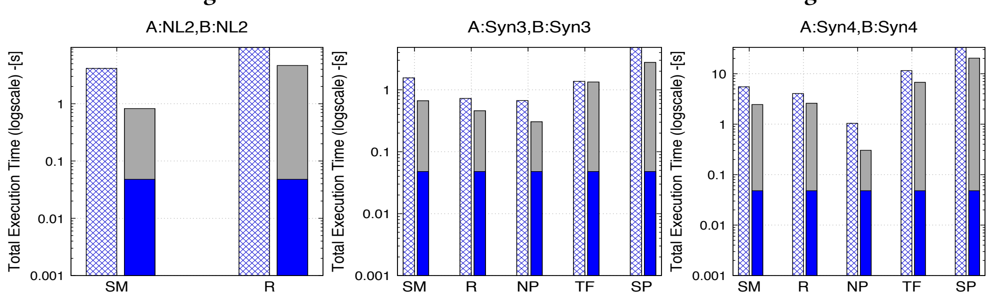
(a) **P2.3** (b) **P2.3**

Figure 32: P2.3 evaluation time with and without rewriting



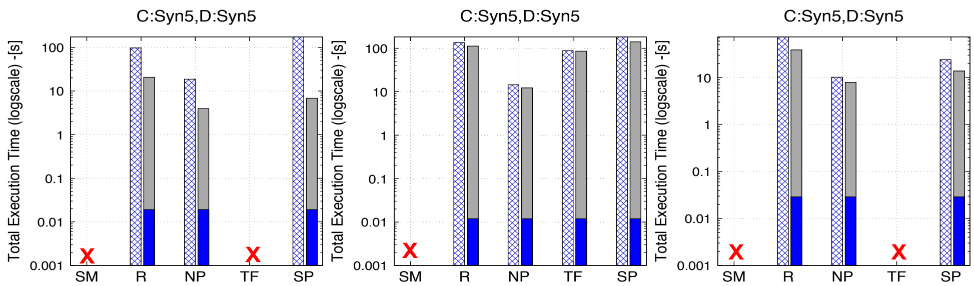
(a) **P2.4** (b) **P2.4** (c) **P2.4**

Figure 33: P2.4 evaluation time with and without rewriting



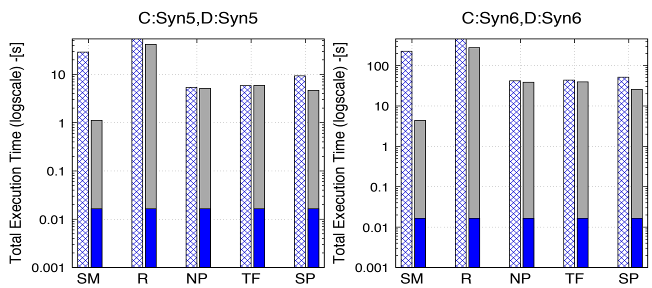
(a) **P2.4** (b) **P2.4** (c) **P2.4**

Figure 34: P2.4 evaluation time with and without rewriting



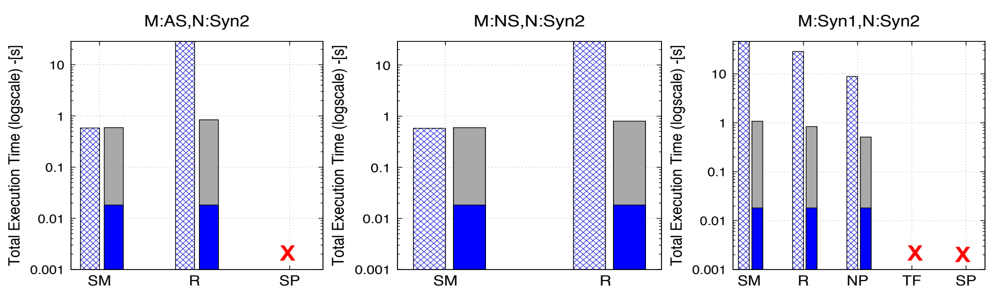
(a) **P2.5** (b) **P2.6** (c) **P2.8**

Figure 35: P2.5, P2.6 and P2.8 evaluation time with and without rewriting



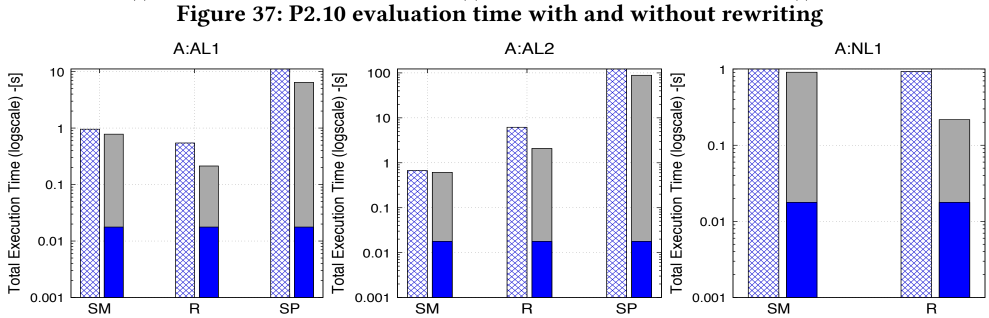
(a) **P2.9** (b) **P2.9S**

Figure 36: P2.9 evaluation time with and without rewriting



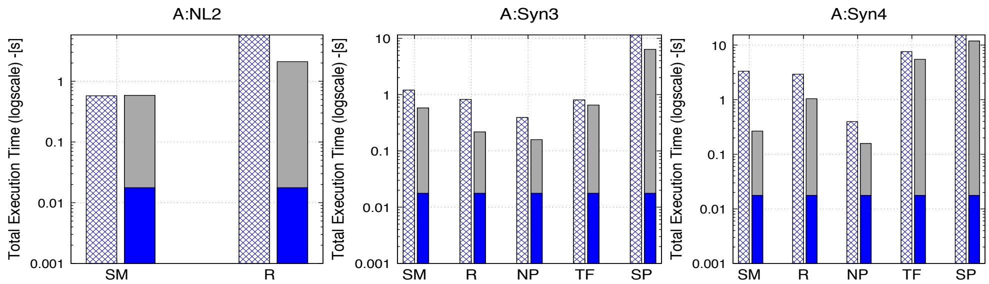
(a) **P2.10** (b) **P2.10** (c) **P2.10**

Figure 37: P2.10 evaluation time with and without rewriting



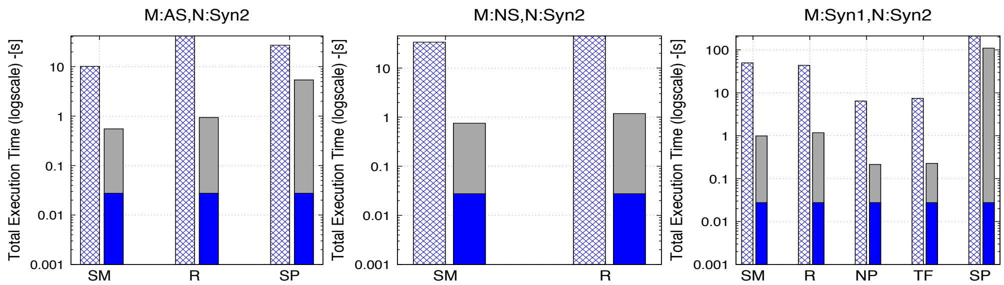
(a) **P2.11** (b) **P2.11** (c) **P2.11**

Figure 38: P2.11 evaluation time with and without rewriting



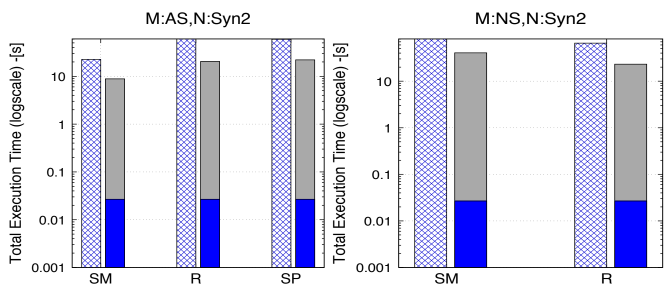
(a) **P2.11** (b) **P2.11** (c) **P2.11**

Figure 39: P2.11 evaluation time with and without rewriting



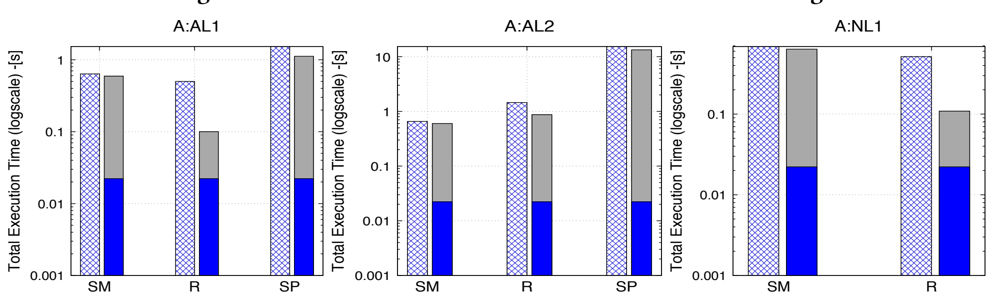
(a) **P2.13** (b) **P2.13** (c) **P2.13**

Figure 40: P2.13 evaluation time with and without rewriting



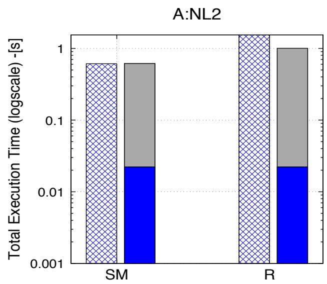
(a) **P2.14** (b) **P2.14**

Figure 41: P2.14 evaluation time with and without rewriting

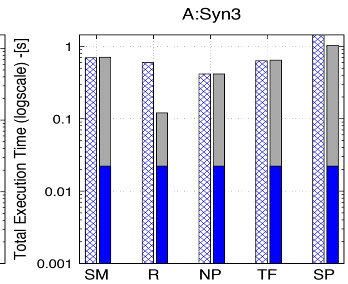


(a) **P2.15** (b) **P2.15** (c) **P2.15**

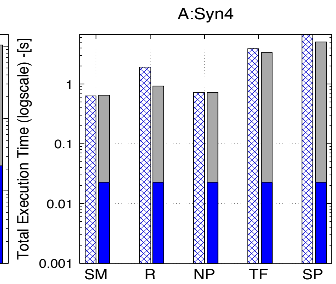
Figure 42: P2.15 evaluation time with and without rewriting



(a) **P2.15**

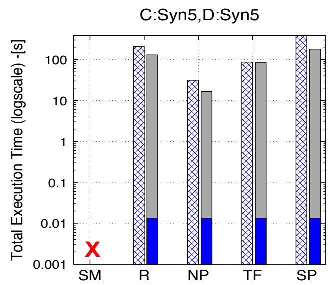


(b) **P2.15**



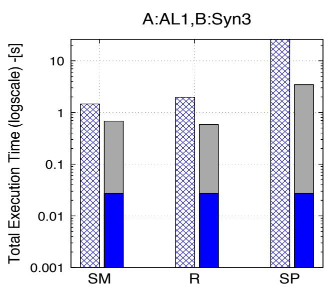
(c) **P2.15**

Figure 43: P2.15 evaluation time with and without rewriting

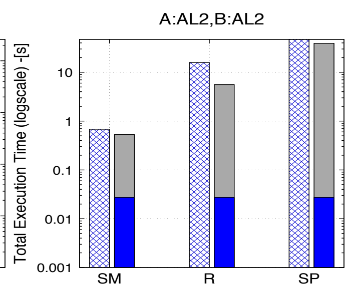


(a) **P2.16**

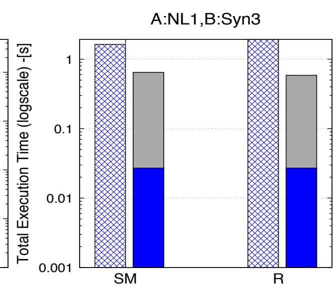
Figure 44: P2.16 evaluation time with and without rewriting



(a) **P2.18**

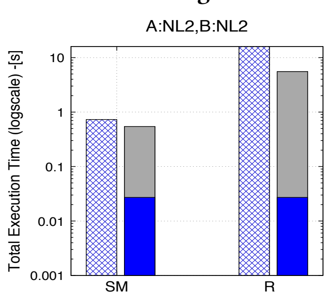


(b) **P2.18**

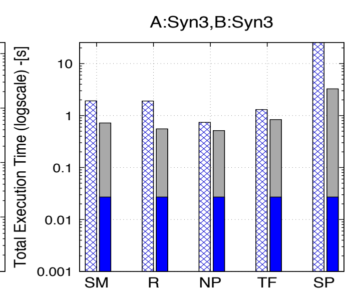


(c) **P2.18**

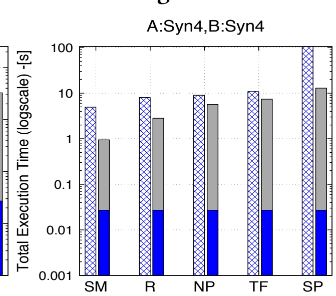
Figure 45: P2.18 evaluation time with and without rewriting



(a) **P2.18**



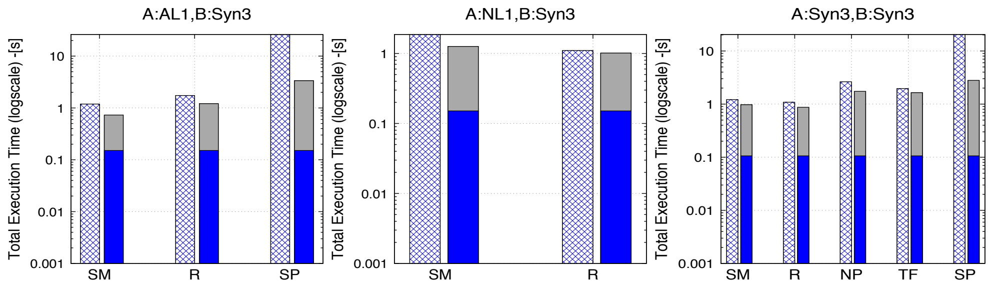
(b) **P2.18**



(c) **P2.18**

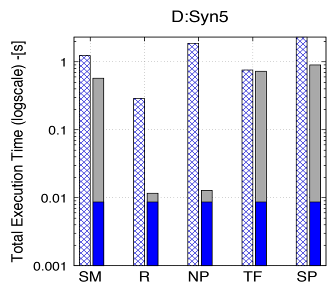
Figure 46: P2.18 evaluation time with and without rewriting

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



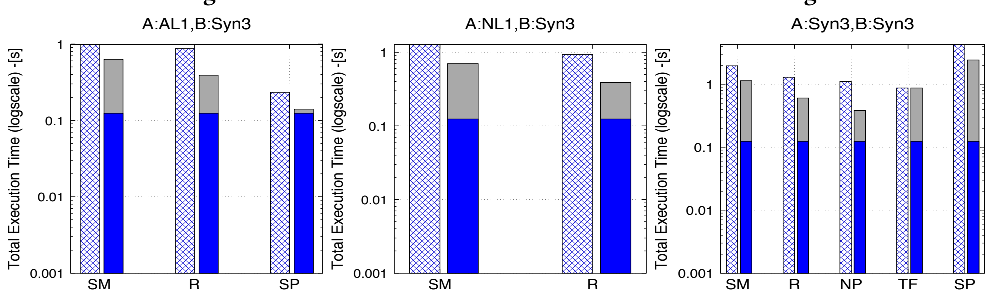
(a) **P1.2** (b) **P1.2** (c) **P1.2**

Figure 47: P1.2 evaluation time with and without rewriting



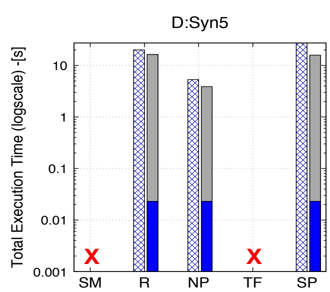
(a) **P1.6**

Figure 48: P1.6 evaluation time with and without rewriting



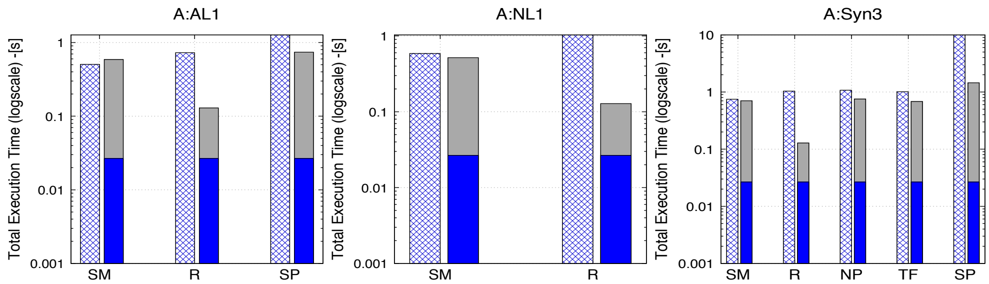
(a) **P1.8** (b) **P1.8** (c) **P1.8**

Figure 49: P1.8 evaluation time with and without rewriting



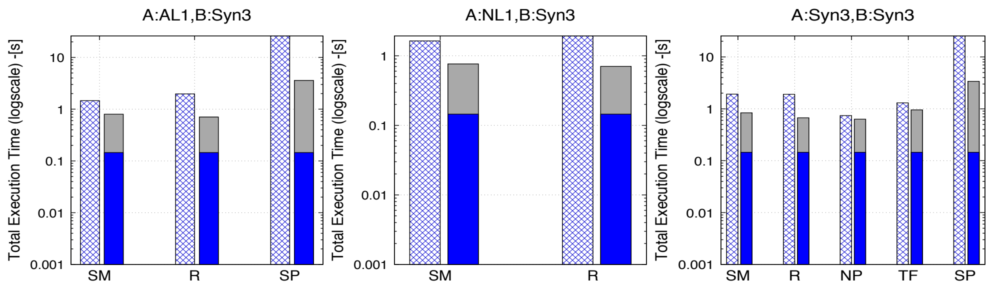
(a) **P1.9**

Figure 50: P1.9 evaluation time with and without rewriting



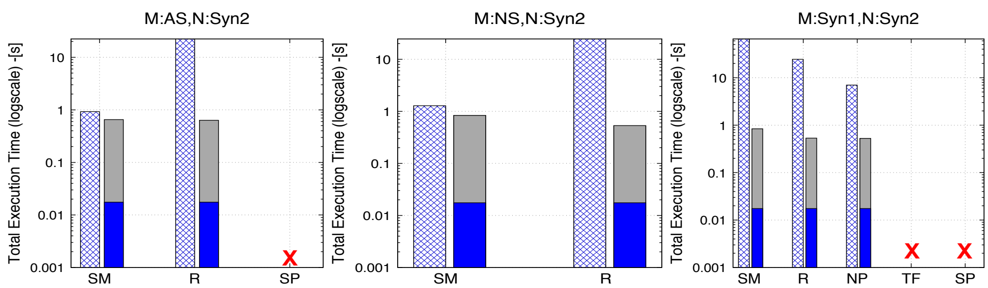
(a) **P1.10** (b) **P1.10** (c) **P1.10**

Figure 51: P1.10 evaluation time with and without rewriting



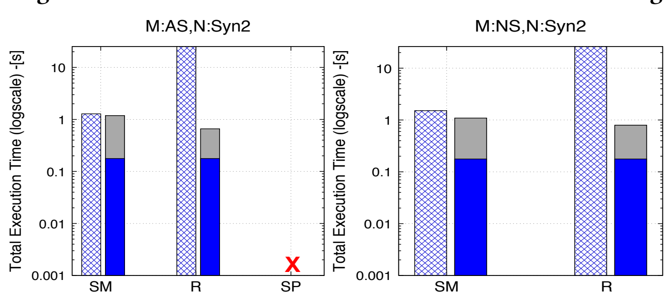
(a) **P1.11** (b) **P1.11** (c) **P1.11**

Figure 52: P1.11 evaluation time with and without rewriting



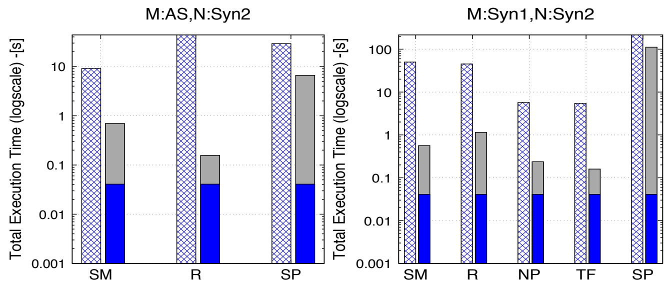
(a) **P1.12** (b) **P1.12** (c) **P1.12**

Figure 53: P1.12 evaluation time with and without rewriting



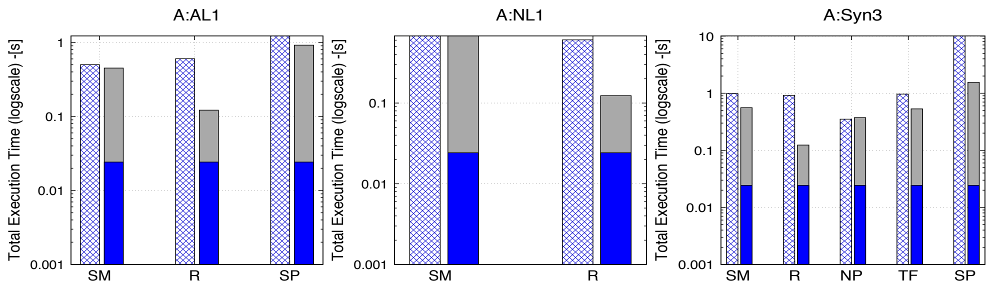
(a) **P1.14** (b) **P1.14**

Figure 54: P1.14 evaluation time with and without rewriting



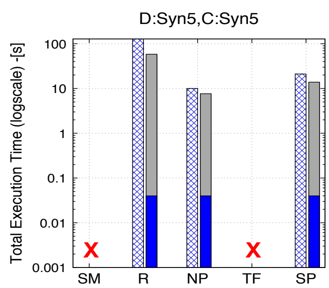
(a) **P1.15** (b) **P1.15**

Figure 55: P1.15 evaluation time with and without rewriting



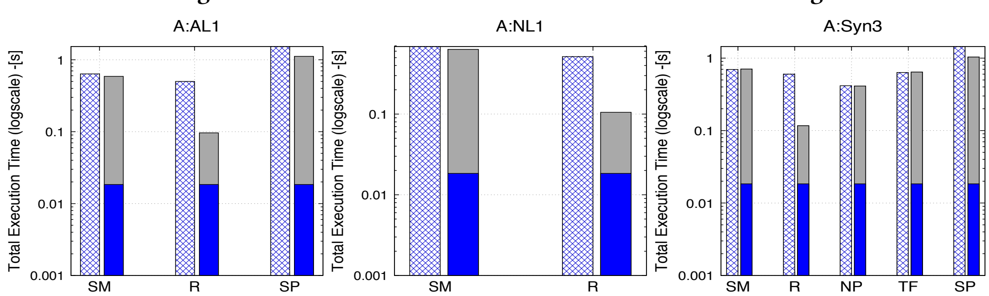
(a) **P1.16** (b) **P1.16** (c) **P1.16**

Figure 56: P1.16 evaluation time with and without rewriting



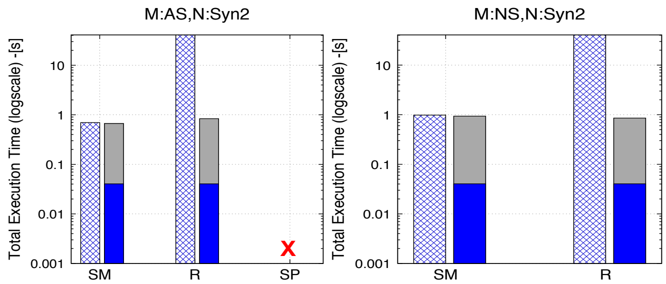
(a) **P1.17**

Figure 57: P1.17 evaluation time with and without rewriting



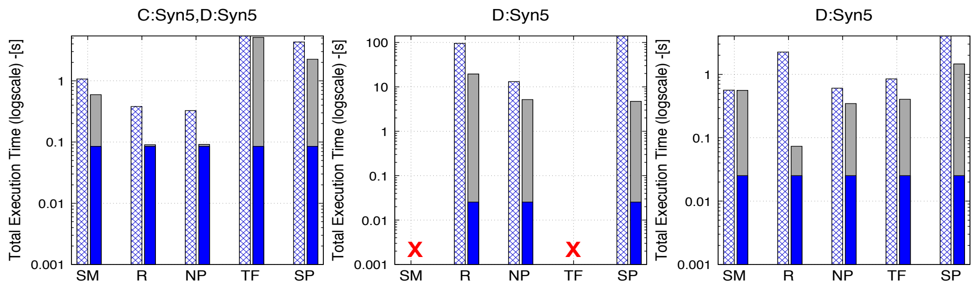
(a) **P1.18** (b) **P1.18** (c) **P1.18**

Figure 58: P1.18 evaluation time with and without rewriting



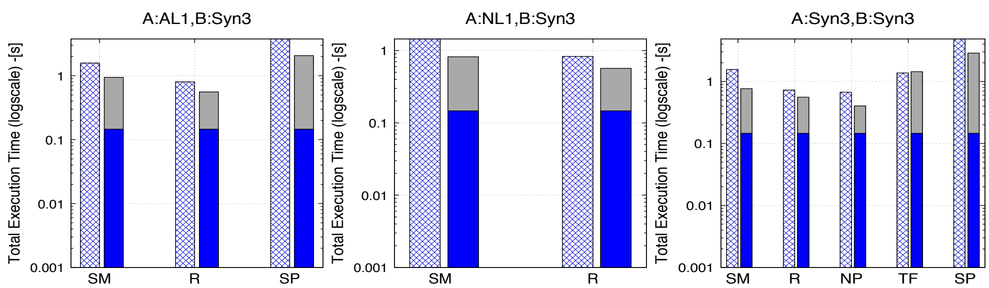
(a) **P1.25** (b) **P1.25**

Figure 59: P1.25 evaluation time with and without rewriting



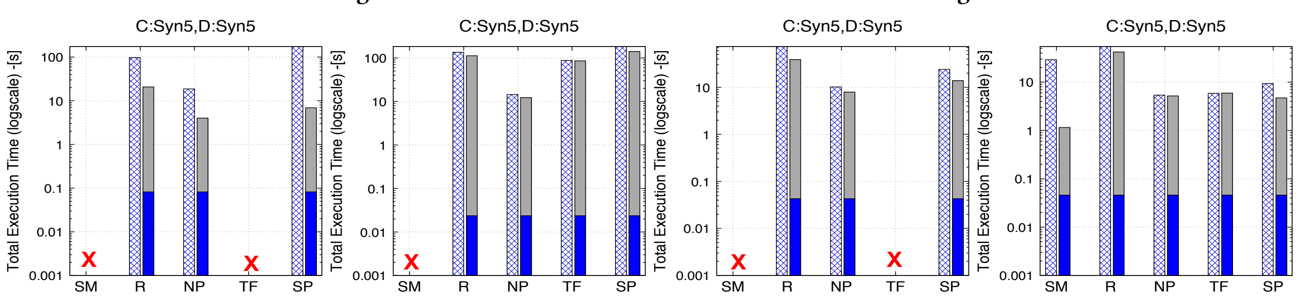
(a) **P2.1** (b) **P2.2** (c) **P2.3**

Figure 60: P2.1 evaluation time with and without rewriting



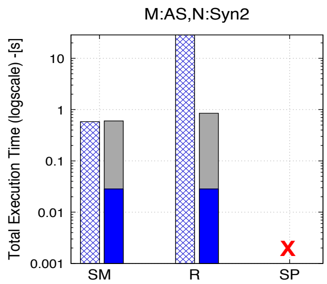
(a) **P2.4** (b) **P2.4** (c) **P2.4**

Figure 61: P2.4 evaluation time with and without rewriting

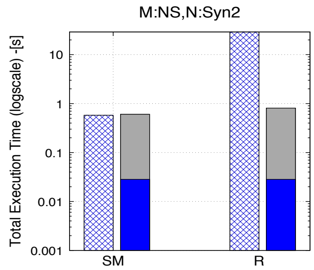


(a) **P2.5** (b) **P2.6** (c) **P2.8** (d) **P2.9**

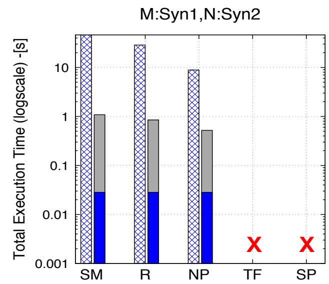
Figure 62: P2.5, P2.6, P2.8 and P2.9evaluation time with and without rewriting



(a) **P2.10**

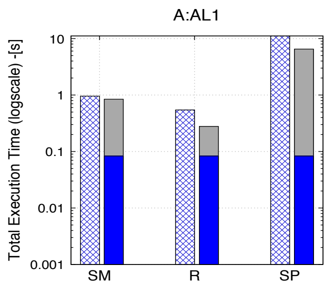


(b) **P2.10**

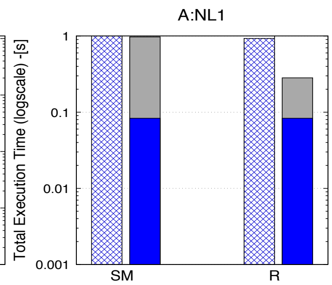


(c) **P2.10**

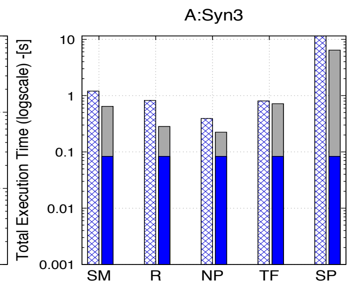
Figure 63: P2.10 evaluation time with and without rewriting



(a) **P2.11**

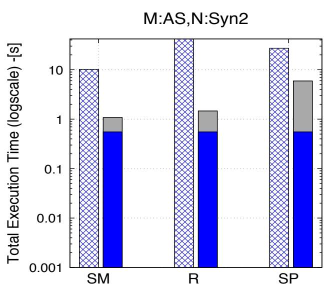


(b) **P2.11**

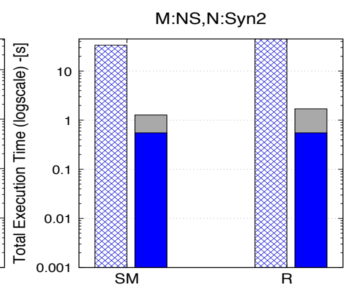


(c) **P2.11**

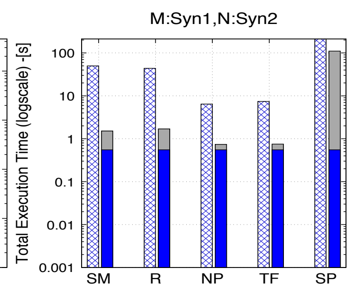
Figure 64: P2.11 evaluation time with and without rewriting



(a) **P2.13**

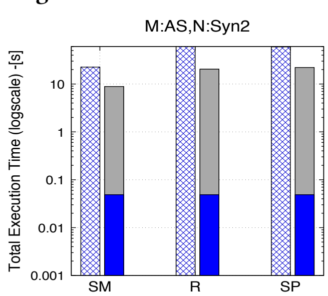


(b) **P2.13**

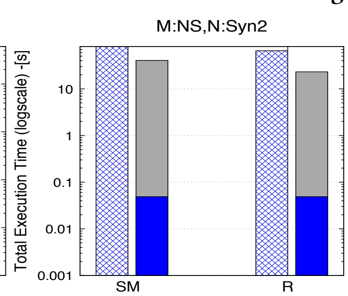


(c) **P2.13**

Figure 65: P2.13 evaluation time with and without rewriting

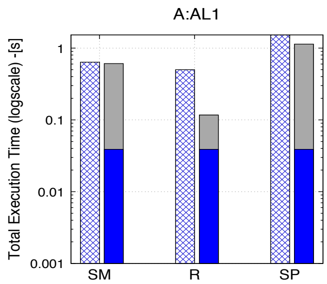


(a) **P2.14**

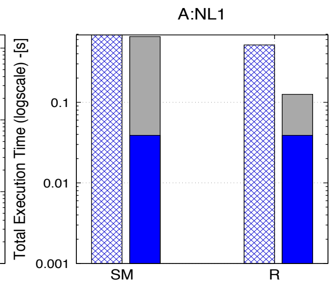


(b) **P2.14**

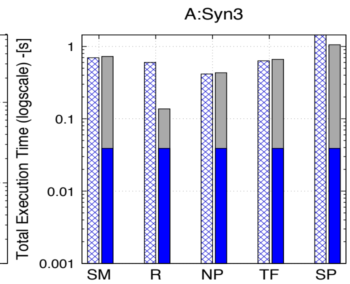
Figure 66: P2.14 evaluation time with and without rewriting



(a) **P2.15**

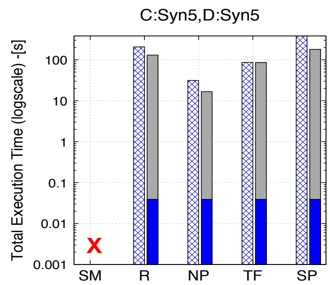


(b) **P2.15**



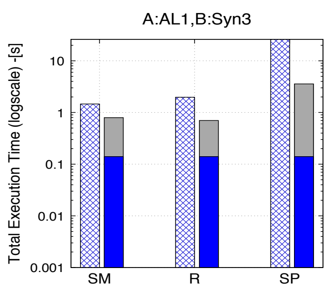
(c) **P2.15**

Figure 67: P2.15 evaluation time with and without rewriting

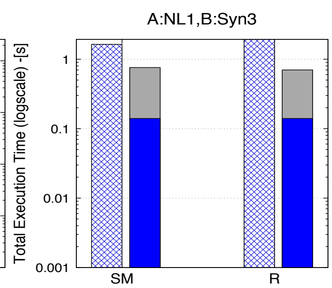


(a) **P2.16**

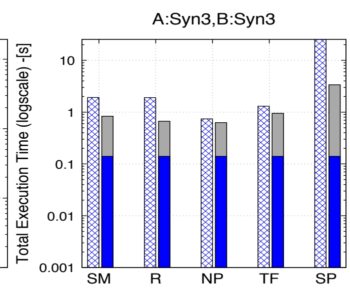
Figure 68: P2.16 evaluation time with and without rewriting



(a) **P2.18**



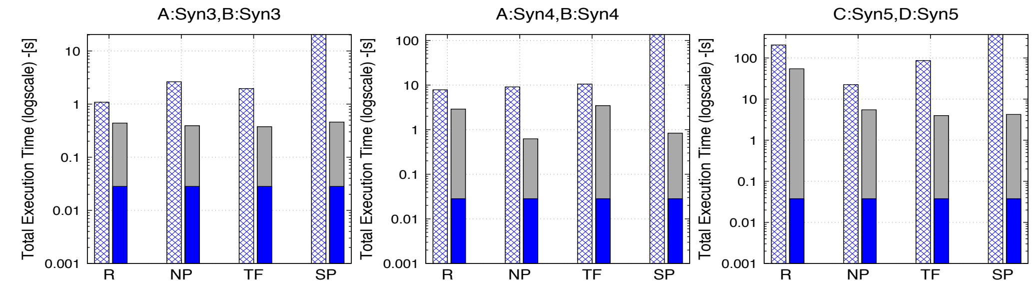
(b) **P2.18**



(c) **P2.18**

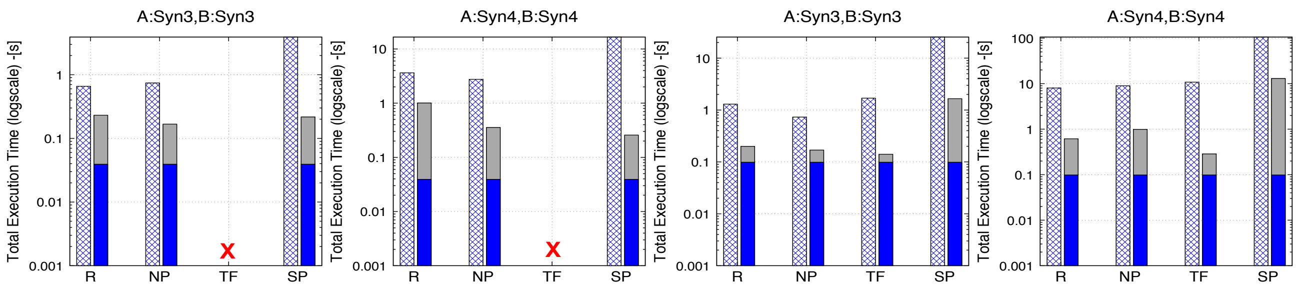
Figure 69: P2.18 evaluation time with and without rewriting

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



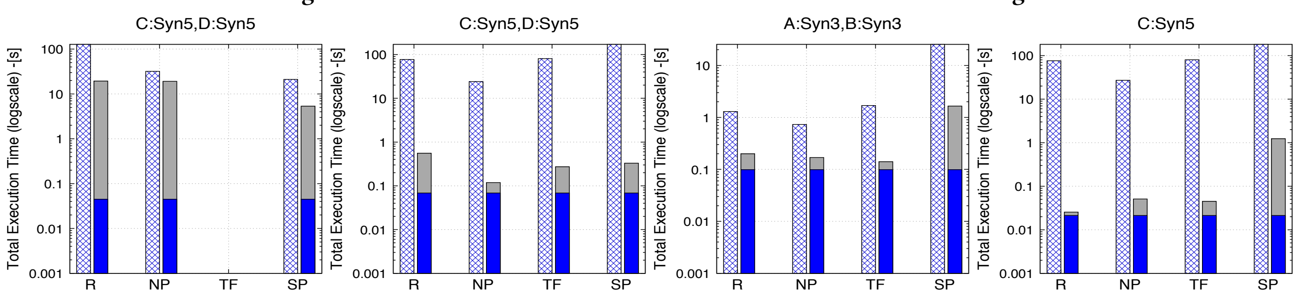
(a) **P1.2** (b) **P1.2** (c) **P1.3**

Figure 70: P1.2 and P1.3 evaluation time with and without rewriting



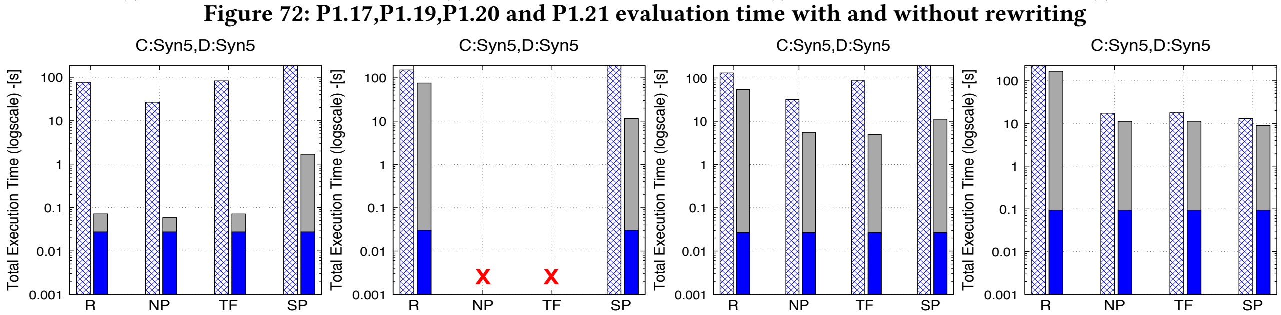
(a) **P1.4** (b) **P1.4** (c) **P1.11** (d) **P1.11**

Figure 71: P1.4 and P1.11 evaluation time with and without rewriting



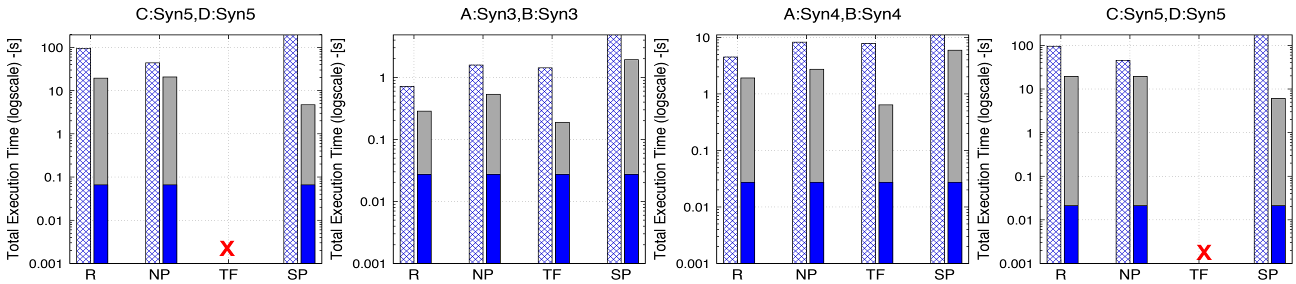
(a) **P1.17** (b) **P1.19** (c) **P1.20** (d) **P1.21**

Figure 72: P1.17,P1.19,P1.20 and P1.21 evaluation time with and without rewriting



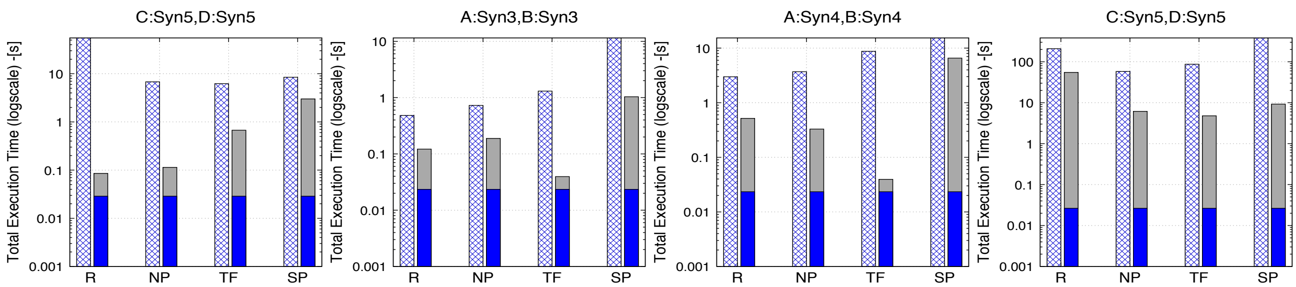
(a) **P1.22** (b) **P1.23** (c) **P1.24** (d) **P1.29**

Figure 73: 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 74: 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 75: P2.9,P2.11 and P2.16 evaluation time with and without rewriting

G $\mathcal{P}^{\neg Opt}$ 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	$s_1 trace(D)$
P1.7	A	P1.8	$(s_1 + s_2)A$	P1.9	$det(D)$
P1.10	$colSums(A)^T$	P1.11	$colSums(A+B)^T$	P1.12	$colSums(M)N$
P1.13	$sum(colSums(M)^T * rowSums(N))$	P1.14	$sum(colSums(M)^T * rowSums(N))$	P1.15	$M(NM)$
P1.16	$sum(A)$	P1.17	$det(C) * det(D) * det(C)$	P1.18	$sum(A)$
P1.25	$M \odot (N^T/(M(NN^T)))$				

Table 15: $\mathcal{P}^{\neg Opt}$ Pipelines (Part 1) Rewrites

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

Table 16: $\mathcal{P}^{\neg Opt}$ 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}	$det(CD)$	V_{11}	$det(DC)$	V_{12}	$(DC)^T$

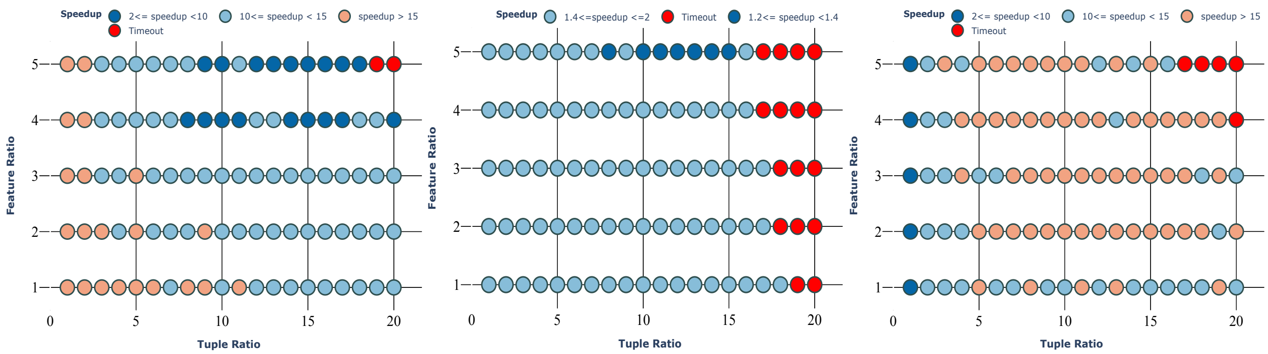
Table 17: The set of views V_{exp}

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	$colSums(V_6)^T$	P1.15	$M(V_3)$	P1.17	$V_{10} * det(C)$
P1.19	V_2	P1.20	$trace(V_7)$	P1.21	$(C+V_1)^T$
P1.22	$trace(V_9)$	P1.24	$trace(V_1 V_7) + trace(D)$	P1.29	$V_5 CCC$
P1.30	$V_3 \odot V_3 R^T$	P2.2	$det(V_1)$	P2.4	$s_1(V6)$
P2.5	$det(V_9)$	P2.6	$(V_1 C)^T$	P2.9	$trace(V_{12}) + trace(D)$
P2.11	$sum(V_6)$	P2.13	$(MV_3)^T$	P2.14	$MV_3 N$
P2.16	$trace(V_7 V_1) + traceD)$	P2.17	$(V_9^T)D$	P2.18	$rowSums(V_6)^T$
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	$det((V_7 V_1) + D)$	P2.26	$exp(V_9)$	P2.27	$V_9^T V_5$

Table 18: $\mathcal{P}^{Views}$ Pipelines Rewrites

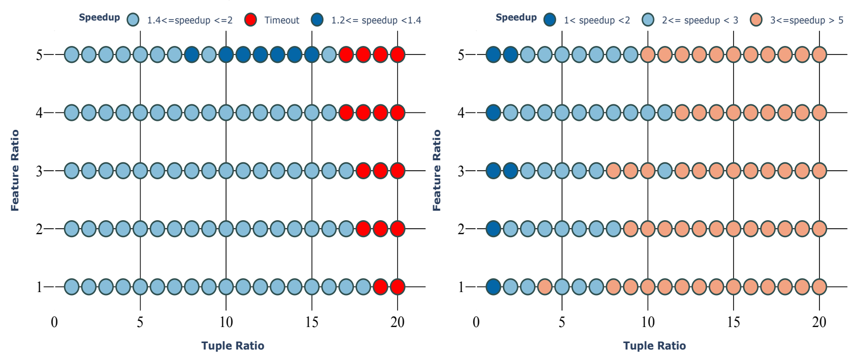
H ADDITIONAL RESULTS: FACTORIZED LA EXPERIMENTS

⟹[RA: Still more results to plot .. waiting for the experiments to finish]



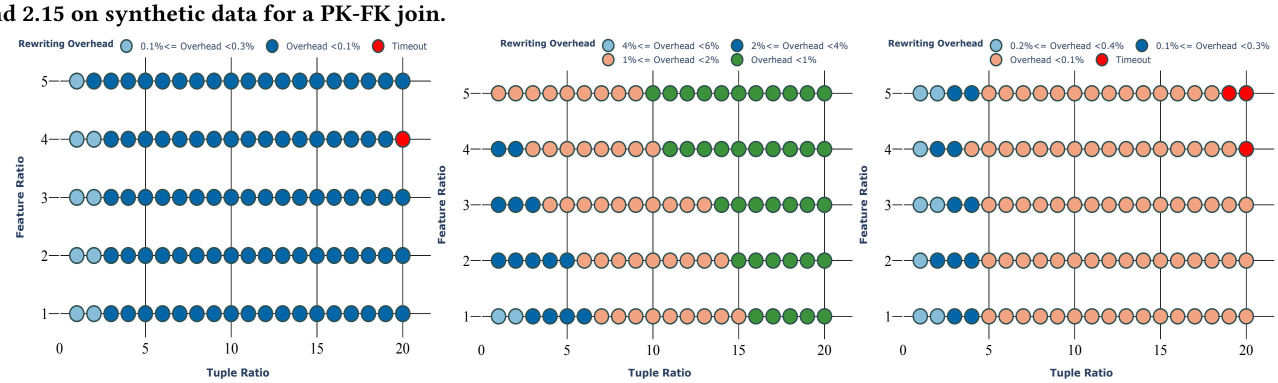
(a) **P1.12** (b) **P1.14** (c) **P2.11**

Figure 76: Speed-ups of MorpheusR (with HADAD rewrites) over MorpheusR (without HADAD rewrites) for pipelines 1.12, 1.14 and 2.11 on synthetic data for a PK-FK join.



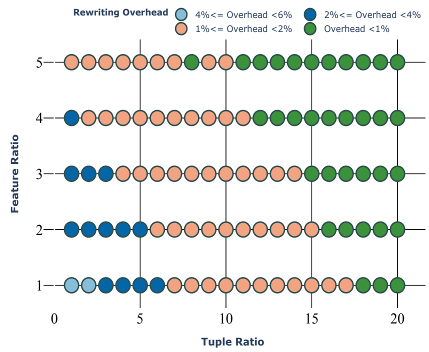
(a) **P2.12** (b) **P2.15**

Figure 77: Speed-ups of MorpheusR (with HADAD rewrites) over MorpheusR (without HADAD rewrites) for pipelines 2.12 and 2.15 on synthetic data for a PK-FK join.



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

Figure 78: RW_{find} **overhead as a percentage (%) of the total time (**$Q_{exec} + RW_{fond}$**) for pipelines 1.1,1.10 and 1.13**



(a) **P1.16**

Figure 79: RW_{find} **overhead as a percentage (%) of the total time (**$Q_{exec} + RW_{fond}$**) for pipelines 1.16 and 1.18**