Project Paper: Optimal Implementation of Conjunctive Queries in Relational Databases

Topic 1: Foundations of Relational Query Languages - Paper No. 1

s1856331: Muthuri Mwenda - April 2019

Abstract

This paper defines the class of conjunctive queries, which is the fragment of relational calculus without negation, universal quantification and disjunction. The class forms a requisite for the introduction of a generalized join operator, which can be implemented using matrix multiplication, that is used to design a “near-optimal” model of computing conjunctive queries. This optimization is achieved through (potentially) changing the structure of a query by reducing the number of variables in it, a process called minimization, and then choosing an order of computation.

1 Introduction

The relational model of databases describes data by its natural structure only. As defined by Codd [3], given sets $S_1, S_2, ..., S_n$, R is a relation on these n sets if it has a set of n-tuples, each of which have their first element form S_1, the second from S_2 and in that order till the n-th element. Every distinct element z in R belongs to the finite non-empty set D, which is called the domain of R. A relational database B, therefore, is based on such a relational model.

Querying relational databases using relational algebra or tuple relational calculus as defined by Codd [3] has been shown to yield the same results, and thus both languages are equally expressive. Notably, the complexity of general relational algebra and tuple relational calculus queries is PSPACE-complete [11]. This motivates the question of how to optimize query evaluation, which is studied in this paper. It begins by introducing conjunctive queries in 2, then defines the generalized join operator in Section 3 and query minimization in Section 4.1. Section 5 describes an optimized model of computation for conjunctive queries.

2 Conjunctive Queries

A conjunctive query is defined as a restricted form of a first-order query. We recall some first-order notation before formalizing the definition of conjunctive query.

A **Relational database** $B = (D, R_1, R_2, ...R_s)$ consists of domain D and a finite number of relations $R_1, ..., R_s$ on D. The rank of a relation is the number of distinct columns appearing in it. Thus, a relation R_i of rank p is denoted as R_i^p.
A **first order formula** ϕ is an atomic formula that asserts a relationship between certain quantifiable elements. The formula ϕ uses quantifiers $\forall$, $\exists$, logical connectives $\vee, \wedge, \neg$, variables x_i that are contained in domain D, constants a_i taken from a possibly infinite set of constants A and relations R_i.[1] Distinct constants a_i in the formula stand for distinct elements.
A **first order query** is of the form $(x_1, x_2, ..., x_k) \cdot \phi(x_1, ..., x_k)$ where ϕ is a first-order formula which has no free variables other than those in $(x_1, ..., x_k)$ of the query $(x_1, x_2, ..., x_k) \cdot \phi$. The query has rank k, and variables in the vector $(x_1, ..., x_k)$ cannot be repeated.
A first order query Q and database B are **compatible** if:
i. each relation R_i in Q is also in the database B and
ii. each constant a_i in Q is an element of the domain D of B.
A **result**, Res(Q,B), of the query $Q = (x_1, ..., x_k) \cdot \phi(x_1, ..., x_k)$ on a compatible database B with domain D is the set $\{(y_1, ..., y_k) \in D^k | \phi(y_1, ...y_k) \text{ is true in } B\}$.
Two queries Q_1, Q_2 are **equivalent** if for every compatible database B, $\text{Res}(Q_1, B) = \text{Res}(Q_2, B)$. Such an equivalence is denoted as $Q_1 \equiv Q_2$.
A **Boolean query** is of the form $().\phi()$, having no free variables. The result is either *true*, the set containing the empty vector or *false*, the empty set.

[1] First order formulas do not use equality.

A **conjunctive query**, therefore, is defined as a first-order query of the form $(x_1, x_2, ..., x_k) \cdot \exists x_{k+1} x_{k+2} .... x_m \cdot A_1 \wedge A_2 \wedge ... \wedge A_r$ where each A_i us an atomic formula of the form $R^p_j(y_1, ..., y_p)$ where each y is either a variable x_q[2] or a constant a_q.

Example 1. Given relation Footballers (Name, Citizenship), list all footballers who hold a dual citizenship of Malta and Finland. The query is:

$$(x) \cdot \text{Footballers}(x, Malta) \wedge \text{Footballers}(x, Finland)$$

Given relation Routes (Bus Number, Origin, Destination), is there any bus x which is scheduled to ferry passengers from city y and make a round trip back? The query is:

$$() \cdot \exists x, y, z \cdot \text{Routes}(x, y, z) \wedge \text{Routes}(x, z, y)$$

3 The Generalized Join

The generalized join is defined as a canonical operation for conjunctive queries such that expressions using the generalized join, selection and restriction operators are equivalent to conjunctive queries. It is an operation that corresponds to taking the join of two relations that overlap on some columns r, followed by deleting some specified columns. Evidently, it follows that the generalized join is a specification of a *universal* relational algebra operation.

3.1 Operations

As defined by Codd [4], relational algebra is a procedural query language that models data stored in databases by taking relations as input instances and returning relations as outputs. For the purposes of this paper, some of Codd's defined relational algebra operations are redefined with some modifications below (redefined operations are marked with $^\iota$). These form the pre-requisites for defining the generalized join operator.

1. **Cartesian product**, $P \times Q$, is a concatenation of every tuple of relation P with all tuples of relation Q such that for $P = \{x_1, x_2, ..., x_p\}$ and $Q = \{y_1, y_2, ..., y_q\}$, $P \cdot Q = (x_1, x_2, ..., x_p, y_1, y_2, ..., y_q)\}$.
2. **Intersection** between relations P and Q, is $P \cap Q = \{v | v \in P \wedge v \in Q\}$. The relations should be of the same rank.
3. **Permutation** f on Z_p is one of the ways in which elements of the vector Z_p can be ordered. The permutation of a p-ary relation P with respect to f is denoted as $\text{Perm}_f(P) = \{(x_1, ..., x_p) \mid x_{f(1)}, ..., x_{f(p)} \in P\}$
4. **Projection**$^\iota$ of a relation P of rank p, with p ≥ 1, is $\pi_p(P) = \{(x_1, ..., x_{p-1}) | \exists x_p \cdot (x_1, ..., x_p) \in P\}$.
5. **Join** is the merging of relations on their common attributes. Given relations P^p, Q^q where $r \subseteq p$ and $r \subseteq q$, the join of P and Q is $P \bowtie Q = \{(x_1, ..., x_{p-r+q}) | (x_1, ..., x_p) \in P \wedge (x_{p-r+1}, ..., x_{p-r+q}) \in Q\}$. If $r = 0$, join is the same as the Cartesian product.
6. **Restriction**$^\iota$ - Given a p-ary relation P and r, s such that $1 \leq r < s \leq p$, $\text{Restrict}_{r,s}(P) = \{(x_1, ..., x_{s-1}, x_{s+1}, ..., x_p) | (x_1, ..., x_{s-1}, x_r, x_{s+1}, ..., x_p) \in P\}$. See Figure 1 for an example.
7. **Selection** returns tuples satisfying some condition. Given a p-ary relation P, $r \leq p$ and constant $a \in A$[a], selection is denoted as $\sigma_{r,a}(P) = \{(x_1, ..., x_{r-1}, a, x_{r+1}, ..., x_p) \in P\}$.
8. **Union** is an operation on relations P and Q of the same rank defined as $P \cup Q = \{v | v \in P \vee v \in Q\}$.
9. **Difference** is an operation on two relations P and Q of the same rank that returns tuples of P that are not in Q. Formally, $R - S = \{p | p \in P \wedge p \notin Q\}$.
10. **Division**$^\iota$ of a p-ary relation P, $p \geq 1$, and an implicit domain D, the division of P is defined as Div $P = \{(x_1, ..., x_{p-1}) | \forall x_p \in D, (x_1, ..., x_{p-1}, x_p) \in P\}$.

[a]Take A to be the set of constants.

[2]A variable that is within the domain; $q \leq m$.

A **relational expression** is built up from the domain D, relations R_i and using the relational operators defined in Section 3.1 above. The compatibility of expressions is similar to that of first-order queries. The result of a relational expression E on a compatible database B, **Res(*E*,*B*)**, is the value of E where relations R_i and domain D have interpretations as assigned by B. Two relational expressions or an expression and a query are **equivalent**, $\equiv$, if their results are equal for every compatible database.
With these operators and their corresponding operations defined, the generalized join operator is formalized.

3.2 The Generalized Join Operator

Take $Z = \{1, 2, ..., n\}$. Given two relations P^p and Q^q, integer r such that $0 \le r \le p + q$ and two injective maps $f : z_p \to z_{p+q}$ and $g : z_q \to z_{p+q}$ such that $z_r \subset f(z_p) \cup g(z_q)$[3], the generalized join of P and Q with respect to r, f, g is defined as:

$$J_{r,f,g}(P, Q) = \{(x_1, ..., x_r) | \exists x_{r+1}, ..., x_{p+q} \cdot (x_{f(1)}, x_{f(2)}, ..., x_{f(p)}) \in P \bowtie (x_{g(1)}, x_{g(2)}, ..., x_{g(q)}) \in Q\}$$

Example 2. Given relations P^4, Q^4, and maps $\{f(1), ..., f(4) \mapsto (1, 2, 3, 5)\}$ and $\{g(1), ..., g(4) \mapsto (2, 3, 4, 6)\}$, the generalized join of P and Q with respect to $r = 3$ is:

$$J(P, Q) = \{(x_1, x_2, y_2) | \exists x_3, x_4, y_4 \cdot P(x_1, x_2, x_3, x_4) \bowtie Q(x_2, y_2, x_3, y_4)\}$$

This join can be computed by following the steps outlined below.

Step 1: Compute the join of P and Q. Recall that if no attributes are shared, a join is the same as a Cartesian product. This yields an intermediate relation, $P \bowtie Q = (x_1, x_2, y_2, x_3, x_4, y_4)$.

Step 2: With $r = 3$, project out (existentially quantifying) columns $(3 + 1)$ to $(p + q)$ as enacted by $\exists x_3, x_4, y_4 \cdot (x_1, x_2, y_2, x_3, x_4, y_4)$.

Result: The result of the previous two steps is (x_1, x_2, y_2) and is also the result of $J(P, Q)$.

This join can also be computed by first projecting out (existentially quantifying) the fourth columns of P and Q followed by joining the resulting relations:

$$P' = \{(x_1, x_2, x_3) | \exists x_4 \cdot P(x_1, x_2, x_3, x_4)\}$$
$$Q' = \{(y_1, y_2, y_3) | \exists y_4 \cdot P(y_1, y_2, y_3, y_4)\}$$
$$J(P, Q) = J(P', Q') = \{(x_1, x_2, y_2) | \exists x_3 \cdot P'(x_1, x_2, x_3) \wedge Q'(x_2, y_2, x_3)\}$$

Taking $\times$ as a Boolean matrix multiplication, the generalized join $J(P, Q)$ can be obtained by fixing x_2 and treating P'_{x_2} and Q'_{x_2} as Boolean matrices as follows:

$$\begin{aligned} \text{Let } J(P, Q)_{x_2} &= \{(x_1, y_2) | (x_1, x_2, y_2) \in J(P, Q)\} \\ P'_{x_2} &= \{(x_1, x_3) | P'(x_1, x_2, x_3)\} \text{ and} \\ Q'_{x_2} &= \{(x_3, y_2) | Q'(x_2, y_2, x_3)\}, \text{then} \\ J(P, Q)_{x_2} &= P'_{x_2} \times Q'_{x_2} \end{aligned}$$

The relational expressions in Section 3.1 are extended to include the generalized join.

To efficiently compute joins and generalized joins, techniques for Boolean matrix multiplication can be utilized. Let $M(a, b, c)$ be the time taken to compute the product of Boolean matrices of sizes $a \times b$ and $b \times c$.

Lemma 3.1 *Given relations $P \subset D^p$, $Q \subset D^q$ and r, f, g as defined for generalized joins and the size of the domain, $|D| = n$,*

$$s = |\{i | \forall j \cdot g(j) \neq f(i) \le r\}|$$
$$t = |\{i | \exists j \cdot f(i) = g(j) > r\}|$$
$$u = |\{j | \forall i \cdot f(i) \neq g(j) \le r\}|$$
$$v = |\{i | \exists j \cdot f(i) = g(j) \le r\}|$$

then $J_{r,f,g}(P, Q)$ can be computed on a random access machine in time $O(n^p + n^q + n^r) + n^v \cdot M(n^s, n^t, n^u)$.

In the lemma, v represents the fixed variable x_2 as was in our Example 2 above, while s, t and u correspond to variables x_1, x_3 and y_2 respectively.
If $s = t = u = v = 0$, the total running time can be reduced to constant time, $O(1)$ provided that the used data structure for relations is either a set of vectors or a Boolean array along with a bit that indicates if the array is empty or not.
Notably, relational operators 1 to 5 in Section 3.1 are specialized cases of the generalized join as elaborated in the Appendix 7.1.

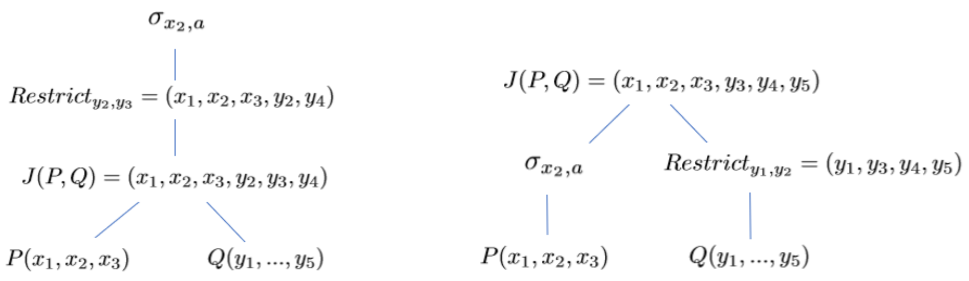
[3]The set Z_r must be smaller than the union of both mappings because if it equals their rank then it implies both mappings are identical.

Lemma 3.2 *Every relational expression containing only the generalized join, selection and restriction is equivalent to some conjunctive query and every conjunctive query is equivalent to some relational expression containing only generalized joins, restriction and selection on the condition that selection and restriction are not applied to a sub-expression containing the generalized join.*

Proof Idea. To convert from relational algebra to relational calculus (conjunctive queries), it suffices that the operations projection, Cartesian product, union and difference correspond to existential quantification, conjunction, disjunction and negation respectively. Selection in relational algebra can be translated to a conjunction of the conditions in the selection clause. Some of the relational calculus operations, however, are not adapted in conjunctive query formulation.

Following the second part of the lemma, restriction and selection can be propagated to the bottom part of the expression tree as illustrated in Figure 1.

Example 3. Take relations $P(x_1, x_2, x_3)$ and $Q(y_1, ..., y_5)$, mappings $\{f(1), f(2), f(3) \mapsto (j, k, l)\}$ and $\{g(1), ..., g(5) \mapsto (j, m, n, o)\}$ and r=6.
The conjunctive query $\sigma_{x_2,a}(Restrict_{y_1,y_2}(J_{r,f,g}(P, Q)))$ illustrated in Figure 1 (a) is prohibited.
Instead, a query like $J_{r,f,g}(\sigma_{x_2,a}(P), Restrict_{y_1,y_2}(Q))$ in Figure 1 (b), a different order of evaluation, is recommended.


$\sigma_{x_2,a}$
$Restrict_{y_2,y_3} = (x_1, x_2, x_3, y_2, y_4)$
$J(P,Q) = (x_1, x_2, x_3, y_2, y_3, y_4)$
$P(x_1, x_2, x_3)$
$Q(y_1, ..., y_5)$
$J(P,Q) = (x_1, x_2, x_3, y_3, y_4, y_5)$
$\sigma_{x_2,a}$
$Restrict_{y_1,y_2} = (y_1, y_3, y_4, y_5)$
$P(x_1, x_2, x_3)$
$Q(y_1, ..., y_5)$

(a) Prohibited order of evaluation (b) Recommended order of evaluation

Figure 1

4 Optimization

Boolean first-order query evaluation on non-trivial databases is PSPACE-complete in combined complexity[4] as illustrated by Vardi [11]. In reality variables of queries tend to be significantly smaller than the entire database. For this reason, it is feasible to optimize such queries with respect to the query size but independent of the database.
One method to optimize a conjunctive queries Q is by *combining variables* in the query in order to find a "new" minimal query Q_0 which:

i contains a smaller number of variables than the original query and

ii evaluates to the same result over all databases; $\text{Res}(Q, B) \equiv \text{Res}(Q_0, B)$ for all databases B.

This optimization has been inspired by minimizing the number of states of a Deterministic Finite Automata (DFA) whose minimization requires finding a DFA with the minimum number of states which is equivalent to the given DFA. It suffices that the minimal DFA is unique *up to renaming of states* [10], a similar observation in conjunctive query minimization, described in Theorem 4.2. This DFA minimization procedure is deterministic and has been shown to be achievable in $O(n\ log\ n)$ by Hopcroft [7]. In contrast, conjunctive query minimization is NP-hard, evidenced by reduction from the 3-colorability problem [8].

Below are necessary definitions that are essential to the description of conjunctive query minimization. The assumption that conjunctive queries within an atomic formula cannot be repeated is held.

Definition 4. Let $Q = (x_1, ..., x_k) \cdot \exists y_1, ..., y_p \cdot A_1 \wedge ... \wedge A_B$ be a conjunctive query. The set of all constants appearing in Q, $X_Q = \{x_1, ..., x_k\}$ and $Y_Q = \{y_1, ..., y_p\}$ is defined as $A_Q = \{a_1, ...a_q\}$. Q is said to be **trivial** is $A_Q = X_Q = Y_Q = \{\}$.
The natural model M_Q of Q is the algebraic structure defined as follows:

i. The domain of M_Q is $D_Q = X_Q \cup Y_Q \cup A_Q$ and

ii. for every relation R^r (with $r \geq 0$) in Q, the corresponding relation in M_Q has value $\{(z_1, ..., z_r) \in D_Q^r |\ R(z_1, ..., z_r)$ is an atomic formula in $Q\}$.

[4]Combined complexity means that both the query and the database are part of the input.

Additionally, M_Q has a finite set of relations taken from disjoint sets $\{S_1, S_2, ...\}$, $\{S_a | a \in A$, the set of constants$\}$, which are also disjoint from names of relations in the databases. S_i and S_a stand for unary relations in model M_Q such that $S_i, 1 \leq i \leq k$, has the value $\{x_i\}$ while S_a, for each $a \in A_Q$, has value $\{a\}$.

Definition 4. Given a conjunctive query Q as above, a homomorphism $h : M_Q \to M_Q$ and a subset V, $X_Q \cup A_Q \subset V \subset D_Q$, such that for all $z \in V$, $h(z) = z$ and for all z, $h(z) \in V$, then if Q' is such a query such that $M_{Q'} = h(M_Q)$, then Q' is a **folding** of Q.
The "folding" can be seen by observing that the subset V may not include all the quantified variables of query Q and noting that the model of Q' equals a potentially smaller model of Q after homomorphism h. An important observation to make on set V is that it constitutes of designated variables X_Q and constants A_Q all of whose mappings by h are to themselves.

Definition 4. If conjunctive queries Q_1 and Q_2 are the same except for variable renaming and reordering of the atomic formulas and quantified variables, then Q_1 and Q_2 are said to be **isomorphic**[5]. Q_1 and Q_2 are isomorphic if and only if (iff) M_{Q_1} and M_{Q_2} are isomorphic.

Theorem 4.1 *Isomorphism of a conjunctive query is logspace equivalent to isomorphism of undirected graphs.*

Proof. Graph isomorphism is trivially reducible to the isomorphism of Boolean conjunctive queries with a single binary relation R such that if $R(x, y)$ is an atomic formula in a query Q, then so is $R(y, x)$.

From Boolean conjunctive query isomorphism to graph isomorphism, we first reduce isomorphism of conjunctive queries to that of undirected node-labelled graph. The construction of graph G from query $Q = (x_1, ..., x_k).\phi$ proceeds as follows: The set of vertices of G include:

1. Free variables $x_1, ..., x_k$ are labelled $1, ..., k$ respectively,
2. each constant $a_i \in A_Q$ is labelled as itself, a_i,
3. each quantified variable $y_i, ...y_k$ is labelled as symbol y and
4. for each atomic formula $A_i = R(z_1, ..., z_r)$ in ϕ there are vertices $A_{i,j}$, $0 \leq j \leq r$ of which $A_{i,0}$ is labelled R, and $A_{i,j}$ is labelled j' for $j \geq 1$. $A_{i,0}$ is connected to the node z_j.

A folding is defined as Church-Rosser if there are two distinct reductions or sequences of reductions that can be applied to the same term, such that there exists a term that is reachable from both results by applying a possibly empty set of additional reductions [2]. Foldings are Church-Rosser up to isomorphism and they preserve the equivalence of queries as illustrated in the theorem below.

Theorem 4.2 *If Q_2 is a folding of Q_1, then $Q_1 \equiv Q_2$.*

Proof. If the sets of constants, free and quantified variables in Q_1 are empty, then the theorem is trivial. Otherwise let h be the homomorphism folding $Q_1 = (x_1, ..., x_2) \cdot \phi_1(x_1, ..., x_k)$ into $Q_2 = (x_1, ..., x_k) \cdot \phi_2(x_1, ..., x_k)$. From their constructions, it can be seen that $\phi_1 \to \phi_2$. This is adequate to show that $\phi_2 \to \phi_1$.
Let $\phi_1 = \exists y_1, ..., y_p \cdot A_1 \wedge ... \wedge A_r$ and $Q_2 = \exists y_{i_1}, ..., y_{i_q} \cdot A'_1 \wedge ... \wedge A'_s$. If for any compatible database and any element $z_1, ..., z_k \in D$, $\phi_2(z_1, ..., z_k)$ is true then there exists a mapping $f : \{y_{i_1}, ..., y_{1_q} \to D\}$ for which $A'_1 \wedge ... \wedge A'_s$ is true. By extending f to $g : \{y_1, ..., y_p\} \to D$ such that $g(y) = f(h(y))$, it is seen that following the definition of folding, $\phi_1(z_1, ..., z_k)$ is true.

4.1 Query Minimization

For every conjunctive query there is a minimal equivalent query, unique up to isomorphism, that can be obtained from the original query by folding. Folding does not increase the number of variables or atomic formulae in the query. The number of variables either remains unchanged or a new query that is isomorphic to the original is obtained. Below is an illustration that query equivalence is guaranteed only when there exists an isomorphism between two queries.

Lemma 4.3 *For conjunctive queries Q_1 and Q_2, $Q_1 \equiv Q_2$ if and only if there exists homomorphisms $R_1 : M_{Q_1} \to M_{Q_2}$ and $h_2 : M_{Q_2} \to M_{Q_1}$.*

Proof. If Part. Suppose h_1 and h_2 exist. Given a compatible database B with domain D, let $(z_1, ..., z_k) \in \text{Res}(Q_1, B)$ be any element in the result of Q_1. Let the matrix of Q_1 (the conjunct of atomic fomulae) be satisfied by mapping $f : D_{Q_1} \to D$ which is the identity function on A_{Q_1} and maps the i-th variable in X_{Q_1} into z_i.

Similarly, matrix of Q_2 is satisfied with the mapping $g : D_{Q_2} \to D$, $g = f \circ h_2$, because h_2 is the identity function of A_{Q_2} and maps the i-th variable in X_{Q_2} to the i-th variable in X_{Q_1} (and hence maps it into z_i)

[5]Isomorphism is a structure preserving mapping that can be reversed by applying an inverse mapping.

and $|X_{Q_1}| = |X_{Q_2}|$. This shows that $(z_1, ..., z_k) \in Res(Q_2, B)$. Therefore, $\text{Res}(Q_1, B) \subset \text{Res}(Q_2, B)$ and conversely, $\text{Res}(Q_2, B) \subset \text{Res}(Q_1, B)$. Hence $Q_1 \equiv Q_2$.

Only if part. If $Q_1 \equiv Q_2$, then $|X_{Q_1}| = |X_{Q_2}|$ because the matrix of a conjunctive query cannot be equivalent to "false". It follows that $A_{Q_1} = A_{Q_2}$ and for all relations R, if R appears in Q_1 then it also appears in Q_2.

Consider a database B with the same M_{Q_1}[6]. B is compatible with Q_1, Q_2 and let $(x_1, ..., x_k) \in \text{Res}(Q_1, B)$, where Q_1 is of the form $(x_1, ..., x_k) \cdot \phi_1$. Let Q_2 be of the form $(x'_1, ..., x'_k) \cdot \phi_2$. Since $Q_1 \equiv Q_2$, $(x_1, ..., x_k) \in \text{Res}(Q_2, B)$, meaning that there is a mapping $h_2 : D_{Q_2} \to D_{Q_1}$[7] that is an identity over constants A_{Q_2} and maps variables x'_i into x_i for all i such that if $R(z_1, ..., z_r)$ is an atomic formula in Q_2, $(h_2(z_1), ..., h_2(z_r)) \in R$ in the model B. Note that h_2 is a homomorphism from M_{Q_1} to M_{Q_2}. Similarly, h_1 exists.

Definition 4. Lemma 4.3 demonstrates that two queries can yield the same result even with a different sets of variables. Since query minimization is geared towards reducing the number of variables in a conjunctive query, the optimal reduction is one which for any finite model M, if h is a homomorphism $h : M \to M$, then every $|h'h(M)| = |h(M)|$. Any such homomorphism is an automorphism[8] and $h(M)$ is said to be a minimal submodel of M. Clearly, such submodels exist for all finite models. $h(M)$ is isomorphic to the submodel induced from M on the domain of $h(M)$. A model like $h(M)$ is referred to as a **standard submodel**.

Theorem 4.4 *For every conjunctive query Q there is a folding Q_0 such that every $Q' \equiv Q$ has a folding Q'_0 isomorphic to Q_0.*

Proof. Given queries Q and Q', let Q_0 andQ'_0 be queries with standard submodels M_{Q_0} and $M_{Q'_0}$ of M_Q and $M_{Q'}$ respectively. These definitions mean that Q_0, Q'_0 are foldings of Q, Q' respectively.

Following Theorem 4.2 on folding and equivalence, then $Q \equiv Q'$ and $Q_0 \equiv Q'_0$. Recall lemma 4.3 proves that queries can only be equivalent iff there exists a homomorphic mapping from a model of a query to the model of another query and vice versa. This means there exists $h_1 : M_{Q_0} \to M_{Q'_0}$ and $h_2 : M_{Q'_0} \to M_{Q_0}$. These homomorphisms describe an automorphism and hence h_1 and h_2 must be isomorphisms. Based on this argument, Q_0 and Q'_0 are isomorphic.

5 The Optimization Model

The optimization model adapts the minimization technique discussed in Section 4. It advances in two general steps; *(i)* (Possibly) altering the structure of the original conjunctive query to a minimal equivalent query and then *(ii)* choosing an order of computation as studied by Neibuhr et al. [9]. This results in an implementation that is within a constant of optimal.

Computing such an optimal implementation takes exponential time in the size of the query but is independent of the database. In some situations, it might be favourable to modify a query for some given period of time rather than optimizing it completely and incurring the full cost of optimal implementation.

5.1 The Model

The model of computation is a straight-line program[9] with variables taking relations as values. Taking X, Y and Z as variables, the statements allowed by the program are:

a. $X \leftarrow R$, with R being a given relation.
b. $X \leftarrow D$, where D is the domain.
c. $X \leftarrow \text{Perm}_f(Y)$ is a permutation.
d. $X \leftarrow \text{Restrict}_{r,s}(Y)$ is a restriction.
e. $X \leftarrow \text{Select}_{r,a}(Y)$, where "a" is a constant and $r \geq 1$.
f. $X \leftarrow J_{r,f,g}(Y, Z)$ is a generalized join.

The output is the value of a designated variable, say X_0, at the end of the computation. There are three requirements that should hold in the program:

1. No "type-checking" error in the program such that a variable cannot be used before it is defined.
2. The designated variable X_0 must be defined in the program.
3. f, r, s and g in items (c) to (f) in the list above must satisfy restrictions in the definitions of the corresponding operations as elaborated in Section 3.1.

[6]This model does not include special relations $S_i and S_a$.
[7]The domain D_{Q_1} is also the domain of database B.
[8]An automorphism is a mapping from a structure to itself.
[9]A straight-line program is one involves a sequence of basic operations with no branches, no loops and no conditional statements.

Definition 5. A program P is said to be **equivalent** to a conjunctive query Q, $P \equiv Q$, if its output equals the result of the query for every database.

The running time for the program is the sum of the running times for the individual statements. For a database with domain D, $|D| = n$, the running times are considered as follows:

1. For statements (a) to (e) above, the running time is zero.
2. For the generalized join operator (f), let p, q, r, s, t, u and v be as defined in Lemma 3.1.

The running time, therefore, is $(n^p + n^q + n^r + n^{s+t+u+v})$.
The total running time of a program P on a database B is denoted as Time(P, B).

Definition 5. Given a conjunctive query Q, a program P is said to be **near-optimal** for Q if $P \equiv Q$ and there exists a constant c such that for every program $P' \equiv Q$ and every compatible database B, Time$(P, B) \leq c \cdot$ Time(P', B).

5.1.1 Discussion of the Model

Statements (a) to (e) are considered to take zero time. They can, however, be implemented in time independent of n (the size of the domain) by using an array representation for arrays and changing the access functions when an operator is applied. The model could be modified to specify that these operations take constant time, or time n for (b) and time n^p for (a), (c), (d) and (e) where relations R and Y have rank p. This modification, however, does not change the results described below.

The running time for the generalized join operation is the sums of lengths of the inputs and output added to the time required for the trivial algorithm for Boolean matrix multiplication.

5.1.2 Optimizing the Model

The running time of any program as described above is a polynomial in n (the size of the domain) with natural numbers as coefficients. Finding a near-optimal program for a given query is a problem of finding one whose polynomial has minimal degree. This requires consideration of expressions using operations (a) to (f) only. Expressions correspond to programs in which every program variable is used exactly once each on the left hand and on the right hand side of an assignment, except the output variable X_0 which is used only once on the left hand side. Expressions suffice because for every program there is an equivalent expression whose running time is a polynomial of no higher degree.

Definition 5. For a program or expression P, of running time $T(n)$, then $T(n)$ is a polynomial whose degree is denoted by Deg(P).
An expression using operators (a) to (f) as in Section 5.1 is said to have **property *** if it is of the form Perm$(E \times D^k)$[10] in which:

i. E is an expression containing no Perm or D, and
ii. no sub-expression of a selection or restriction contains a join.

Lemma 5.1 *For each program P there is an expression E having property * such that $E \equiv P$ and Deg$(E) \leq$ Deg(P).*

Proof Idea. Program P is represented as a directed acyclic graph and expanded into an expression (tree). Elements of domain D that are joined with any other "column" of a relation are deleted. Restrictions and selections of the remaining elements are propagated through joins.
Each expression $E = $ Perm$(E_1 \times D^k)$ having property * is associated with a conjunctive query Q (denoted as $Q \sim E$) such that Q is equivalent to E and is constructed from E. The relation $\sim$ is extended such that if Q_1 and Q_2 are isomorphic and $Q \sim E$, then $Q_2 \sim E$. These definitions readily support the following lemma.

Lemma 5.2 *Given conjunctive queries Q_1 and Q_2, and expression E_1 such that Q_2 is a folding of Q_1 and $Q_1 \sim E_1$, then there is an expression E_2 such that $Q \sim E_2$ and for every compatible database B, Time $(Q_2, B) \leq$ Time (Q_1, B).*

Proof Idea. E_2 can be constructed from E_1 by deleting occurrences of relations in E_1 that correspond to atomic formulae in Q_1 which do not appear in Q_2.
The algorithm that constructs a near-optimal program for a given conjunctive query is described below.

[10]Cartesian product, $\times$, is used instead of join.

5.1.3 The Algorithm

Given a conjunctive query Q, find the minimal query $Q_1 \equiv Q$. Then for all expressions E, $Q_1 \sim E$, choose the one whose running time is a polynomial of minimal degree. The output is a program that implements this expression.
This algorithm runs in exponential time in the length of the input query, but independent of the database.

Example 4. Given query $(x) \cdot \exists u, v, w, y, z \cdot R(x, v, w) \wedge R(x, z, w) \wedge S(u, w) \wedge S(u, v) \wedge S(y, w) \wedge S(y, v) \wedge S(y, z)$.
The minimal query is $(x).\exists w, y, z \cdot R(x, z, w) \wedge S(y, w) \wedge S(y, z)$.
This result is achieved by following the steps enlisted below. It is important to note that the designated variable x is treated as a constant. Recall that the query minimization procedure treats designated variables as constants, and maps them to themselves during the homomorphic mapping.

1. Map variable $v \mapsto z$ from the given query. The result is $(x).\exists u, w, y, z \cdot R(x, z, w) \wedge R(x, z, w) \wedge S(u, w) \wedge S(u, z) \wedge S(y, w) \wedge S(y, z) \wedge S(y, z)$.
2. Since conjunctive queries within an atomic formula cannot be repeated, the query becomes $(x).\exists u, w, y, z \cdot R(x, z, w) \wedge S(u, w) \wedge S(u, z) \wedge S(y, w) \wedge S(y, z)$.
3. Map variable $u \mapsto y$ from the preceding result. The new query is $(x).\exists w, y, z \cdot R(x, z, w) \wedge S(y, w) \wedge S(y, z) \wedge S(y, w) \wedge S(y, z)$. Upon eliminating duplicated tuples, the result is $(x).\exists w, y, z \cdot R(x, z, w) \wedge S(y, w) \wedge S(y, z)$.
   The query cannot be minimized further because the conjuncts of relation S reference different variables of R and thus is the minimal result.

This query can be answered in time $3n^3 + 4n^2 + n$ by computing $X_0 \leftarrow \{\exists z, w \cdot R(x.z.w) \wedge T(y, w)\}$ in time $n + n^2 + 2n^3$.

6 An Alternative Approach to Minimization

Consider a square piece of paper and a fold-and-cut task described as, "Make one complete straight cut on the paper to obtain two pieces.". This is a trivial challenge, as you cut along one straight trajectory. We explore the same task but with the intention to obtain three pieces instead of two. Intuitively, the challenge requires that the piece of paper be folded such that the trajectories of two cuts are perfectly aligned along each other, and not any other part of the paper should fall on this trajectory. Observe that if the original task had a requirement to obtain a rectangle, then the task would require a fold parallel to one edge. If the task is modified requiring that the shape obtained from one straight cut is a triangle, it suffices that the required fold should align all edges along the same trajectory. The fold and cut problem has been studied by Demaine [5] as a mathematical model to origami.

Modelling conjunctive queries graphs as studied in 3-colorability [1] and as trees by Gottlob et al. [6]. We follow this precedence by modelling a Boolean conjunctive as a graph as follows:

i. Graph nodes (vertices) correspond to unary atoms (variables) in a query.
ii. Graph edges correspond to binary atoms.

Lemma 6.1 *Conjunctive query minimization poses a one-to-one correspondence to the fold-and-cut problem in origami.*

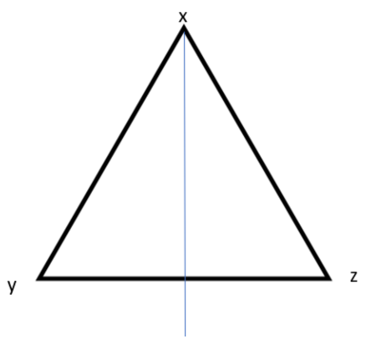
Justification. In the fold-and-cut problem, lining up edges as explained in the preamble, is geared towards lining up all trajectories which ought to be cut to obtain a predefined shape. This can be seen as semantically equivalent to making more than one cut to the same piece of paper to obtain the same result. Similarly, minimizing conjunctive queries is a procedure that attempts to remove variables (or atoms) from a conjunctive query with the purpose of obtaining a semantically equivalent query with fewer variables.

Consider a graph built from a conjunctive query with three binary atoms, $Q(x) = \exists y, z \cdot R(x, y) \wedge R(x, z) \wedge R(z, y)$ as illustrated in Figure 2 (a). By fixing designated variable[11] x of a query to act as a pivot for the other variables y and z, mapping edge (x, y) to (x, z) requires an angular bisection which results in one cut trajectory. The new graph would have two edges. This similarity invites modelling the fold-and-cut algorithm [5] to relational databases that might result in reduction to query minimization times.

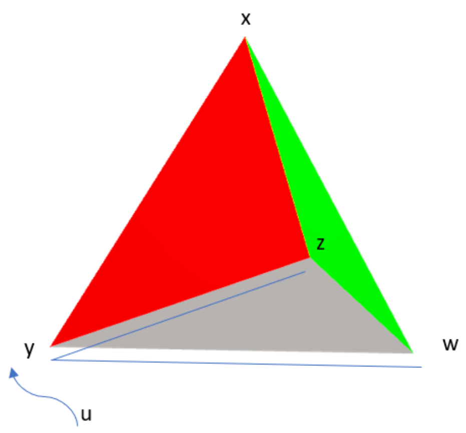
Example 5. We take the result of Example 4 and map it into the graph in Figure 2 (b). The wavy arrow indicates how variable u is mapped to y, an illustration of correspondence of query minimization to

[11] Variables and constants ought to be fixed.

the fold-and-cut problem. If edge (z, w) in the graph were act as a hinge, point u must directly overlap with point v exactly once during one complete rotation of u around (z, w). This is an indication for minimization of u to y or vice versa. Indeed, u is eliminated from the atoms during the query minimization step 3 of the example.
Observe that during the rotation, if u is stopped so as to form a flat plane of nodes (y, z, u, w) that does not overlap with y, then the edge (z, w) would be the cut trajectory to divide the plane to two triangular pieces.



(a) Angular bisector mapping (x, y) to (x, z).



(b) Result of Example 4 indicating $u \mapsto y$ from step 3 of the example.n

Figure 2

References

[1] Miklos Ajtai, Ronald Fagin, and Larry Stockmeyer. The closure of monadic np. *Journal of Computer and System Sciences*, 60(3):660–716, 2000.

[2] Alonzo Church and J Barkley Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, 1936.

[3] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[4] Edgar F Codd. A data base sublanguage founded on the relational calculus. In *Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, pages 35–68. ACM, 1971.

[5] Erik D Demaine, Martin L Demaine, and Anna Lubiw. Folding and one straight cut suffice. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 891–892. Society for Industrial and Applied Mathematics, 1999.

[6] Georg Gottlob, Christoph Koch, and Klaus U Schulz. Conjunctive queries over trees. *Journal of the ACM (JACM)*, 53(2):238–272, 2006.

[7] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.

[8] Phokion G Kolaitis, David L Martin, and Madhukar N Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 197–204. ACM, 1998.

[9] KE Niebuhr and SE Smith. N-ary joins for processing query by example. *IBM Tech. Disclosure Bull*, 19(6):2377–2381, 1976.

[10] Ambuj Tewari, Utkarsh Srivastava, and Phalguni Gupta. A parallel dfa minimization algorithm. In *International Conference on High-Performance Computing*, pages 34–40. Springer, 2002.

[11] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.

7 Appendix

7.1 Relational Operations as Specialized Cases of the Generalized Join

An elaboration of how the generalized join in Section 3 can perform operations 1 to 5 of Section 3.1. The 2 steps in Example 2 of Section 3.2 apply. We use relations P^p and Q^q in the sequel below.

1. Cartesian product, $(P \times Q)$ - Specify the generalized join to ensure that no attributes overlap. This forces the join in Step 1 to act as a Cartesian product of P and Q. Then set $r = (p + q)$ in Step 2.
2. Intersection, $P \cap Q$ - Ensure $\forall x_i \in P \equiv y_i \in Q$. Compute $P \bowtie Q$ in Step 1. In Step 2, set $r = p$.
3. Permutation, $\mathrm{Perm}_f(P)$ - Implement the injective function f to mirror the desired permutation. Compute $P \bowtie P$ Step 1 and set $r = p$ in Step 2.
4. Projection, $\pi_p(P)$ - Compute $P \bowtie P$ in Step 1 and set $r = p - 1$ in Step 2.
5. Join, $P \bowtie Q$ - Compute $P \bowtie Q$ in Step 1. Set $r = p + q$ in Step 2.