A Tractable Class of Disjunctive Deductive Databases *

Zahidul A. Khandaker[1] José A. Fernández[1] Jack Minker[1,2]

[1]Department of Computer Science
and
[2] Institute for Advanced Computer Studies
University of Maryland at College Park
College Park, MD 20742

October 27, 1992

Abstract

In general, computing answers to queries in disjunctive deductive databases is CoNP-complete and therefore computationally infeasible. However, there are some tractable classes of disjunctive databases. In this paper, we present polynomial time algorithms to compute answers to queries in one such tractable class of disjunctive databases where at most two atoms are allowed in any disjunction.

*We greatly appreciate the financial support of the National Science Foundation, provided under the grant Nr. IRI-89-16059, and the Air Force Office of Scientific Research, provided under the grant Nr. AFOSR-91-0350, that made this work possible.

1 Introduction

Disjunctive deductive databases are an outgrowth of work performed in the field of disjunctive logic programming where non-Horn clauses are allowed in the program [GM92]. The declarative, fixpoint and procedural semantics for disjunctive logic programs have been presented in [LMR92] which can be carried over to disjunctive deductive databases to solve many of the semantic issues for such databases.

Answering queries in disjunctive deductive databases has been studied by a number of researchers. Grant and Minker [GM86] were among the first to develope algorithms to answer conjunctive queries in disjunctive databases where there are no rules. Yahya and Henschen [YH85] developed a deductive method to determine whether a conjunction of ground atoms is entailed by the disjunctive database or not. Henschen and Park [HP88] presented a method to answer yes/no queries where the rules may be recursive. Liu and Sunderraman [LS91] generalized the relational model and introduced *M-tables* to represent disjunctive data. They extended the relational algebra operators to apply on M-tables to compute answers to queries. Their extended algebra is sound, but not complete. Yuan and Chiang [YC89] defined an extended algebra and developed a sound and complete query evaluation algorithm for disjunctive databases. Demolombe [Dem89] extended the concepts of magic sets to disjunctive databases. Fernandez and Minker [FM91] introduced a data structure called *model trees* to store the database and developed algorithms for bottom-up evaluation of rules and computation of answers.

All these methods to compute answers in disjunctive databases are in general bound to take exponential time since it has been demonstrated that even in deductive databases with only disjunctive facts and no rules, there are simple conjunctive queries with CoNP-complete complexity [IV89] [Imi91]. Although these results indicate that evaluation of rules and general queries in disjunctive deductive databases is computationally infeasible, a natural question arises is whether there are classes of disjunctive databases where queries can be computed in polynomial time or whether there are classes of queries that can be computed in polynomial time given some syntactic information about the database.

A number of researchers have investigated the complexity issues in disjunctive deductive databases. In [Var86] Vardi considers the complexity of query processing in databases with incomplete information. In [IV89] Imielinsky and Vadaparty give syntactic characterization of CoNP-complete conjunctive queries for a class of disjunctive databases called OR-databases. They give a polynomial time algorithm to compute conjunctive queries that are tractable. However, they consider only definite answers. In [LYW92] Lobo, Yu and Wang show that the computation of sets of pairs in the transitive closure of an acyclic relation with a disjunctive description is polynomial. In [Dal92] Dalal defines a parameter called *cryptness* and uses the measure of cryptness to identify classes of programs that can be queried in polynomial time. However, the algorithms do not provide a practical method for query evaluation; they can be used only as theoretical tools to study complexity of queries.

In this paper, we present algorithms for a tractable class of databases that we call 2-disjunctive databases. We represent the database as a graph and prove a correspondence between answers to queries in the database and paths in the graph with a particular structure. The algorithm computes the paths in polynomial time and derives answers to the query from these paths. The paper is organized as follows. In section 2 we provide some basic notions in disjunctive deductive databases, and discuss previous complexity results. In section 3 we introduce D-graphs and Q-graphs as a way of representing 2-disjunctive deductive databases, define Q-paths in the Q-graphs and show some properties of these graphs. In section 4, we present a polynomial time algorithm to evaluate 2-join queries by computing paths in the Q-graph. In section 6, we classify some classes of rules as tractable in a 2-disjunctive database and present algorithms to evaluate such rules. In the last section, we summarize the results of this paper and discuss directions for fututre research.

2 Background

2.1 Disjunctive deductive databases

A disjunctive deductive database is one in which indefinite information is allowed in the form of disjunctions. Deductive databases are based on function-free first-order logic programs in which only Horn-clauses are allowed. Disjunctive deductive databases are based on function-free disjunctive logic programs in which non-Horn clauses are also allowed. Formally, disjunctive deductive databases are defined as follows:

Definition 1 ([FM91]) Let DB be a function-free disjunctive logic program, and let E_{DB} and I_{DB} be a finite set of extensional and intensional rules, respectively. DB is called a *disjunctive deductive database* iff

1. $E_{DB} = \{C \in DB \mid c = (a_1 \vee \ldots \vee a_n) \wedge C$ is ground $\}$
2. $I_{DB} = \{C \in DB \mid c = (a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_m) \wedge C$ is range-restricted $\}$
3. $DB = E_{DB} \cup I_{DB}$

where the a_i and b_j are atoms.

A clause is said to be *range-restricted* when all variables that occur in the head of the clause also occur in its body.

Example 1 The union of the following two sets of clauses yields a disjunctive deductive database:

$$\begin{aligned} E_{DB} = \quad & father(pat, john) \vee mother(pat, john). \\ & mother(ruth, pat). \end{aligned}$$

$$\begin{aligned} I_{DB} = \quad & parent(X, Y) \leftarrow mother(X, Y). \\ & parent(X, Y) \leftarrow father(X, Y). \end{aligned}$$

The semantics of a disjunctive deductive database DB is defined by a set of minimal Herbrand models of DB. If we denote by $\mathcal{M}_{DB}$ the set of minimal models defining the semantics of a database DB, then a positive clause C is a logical consequence of DB if and only if $\forall M \in \mathcal{M}_{DB}$, $M \models C$.

This characterization of the semantics using $\mathcal{M}_{DB}$ is called model theoretic. A second way to characterize the semantics of a disjunctive database is by defining $\mathcal{M}_{DB}$ as the fixpoint of a monotonic immediate consequence operator. We refer to [LMR92] for such a fixpoint semantics for disjunctive databases.

A query in a deductive database consists of a conjunction of atoms $A_1, \ldots, A_n$ where $n \geq 1$. There are two possible answers for these queries: *yes/no* answers and *generative* answers. An answer to a query $Q(\vec{x})$ in a disjunctive deductive database DB is *yes* iff $DB \models \exists\vec{x}(A_1, \ldots, A_n)$. We refer to such queries for which only *yes/no* answers are sought as *closed queries*. An answer to a query is called *generative* if it is a set of substitutions $\{\theta_1, \ldots, \theta_k\}$ of the variables in the query such that $DB \models (A_1 \wedge \ldots \wedge A_n)\theta_1 \vee \ldots (A_1 \wedge \ldots \wedge A_n)\theta_k$. We refer to answers that can be represented with a single substitution as *definite answers*; otherwise they are called *disjunctive answers*.

In deductive databases, it is necessary to obtain not only a correct answer but the set of all answers. However, we are interested only in minimal answers, i.e. answers that are not subsumed by other answers.

Definition 2 ([FM91]) A set of answers S is a *complete set of answers for a query* $Q(\vec{x})$ iff $(\forall C' \in S, C'$ is an answer to $Q(\vec{x}))$ and $(\forall C'' \notin S, (C''$ is an answer to $Q(\vec{x}) \Rightarrow (\exists C' \in S, C' \Rightarrow C'')))$.

If the set S is empty, then we say that the query $Q(\vec{x})$ is *not entailed* by the database DB.

2.2 Complexity of disjunctive databases

It is known that for a very simple class of queries and disjunctive databases, the complexity of answering queries is CoNP-complete [IV89]. The following example is taken from [Imi91]:

Example 2 Assume that we are given an instance of a graph $G = (V, E)$ and we want to decide whether G can be colored using three colors: *red, green* and *black*. We construct an instance of a disjunctive database DB as follows: *i)* $v \in V$ iff $color(v, red) \vee color(v, green) \vee color(v, black) \in DB$ (the disjunctions represent the information that each node can be colored with red, green or black), *ii)* $\langle m, n\rangle \in E$ iff $edge(m, n) \in DB$(the atoms represent the edges in the graph). Let Q be the following query: $\exists X \exists Y \exists Z edge(X, Y) \wedge color(X, Z) \wedge color(Y, Z)$. The query Q encodes the graph-colorability problem. If the answer to the query Q is *yes*, then the graph G cannot be colored with three colors without having two adjacent nodes with the same color. Thus, for a very simple query, an *yes/no* query of conjunctions of atoms in a database with disjunctions but no rules is intractable.

Even in a restricted case when disjunctions in a database are limited to one column of a single predicate, the intractibility remains. More formally, let $B = P_1(t_1) \vee \ldots \vee P_n(t_n)$ be a ground disjunction. B is called a *restricted* disjunction iff

1. All P_i for $i = 1, \ldots, n$ are the same predicate.
2. All tuples t_i differ exactly on one attribute.

For example, $P(a, b) \vee P(c, d)$ is not a restricted disjunction, but $P(a, b) \vee P(a, c)$ is.

It has been argued that such restricted disjunctions occur much more often in practice then unrestricted disjunctions. Unfortunately, restricted disjunctions are not computationally beneficial (at least in the worst case). The following example shows that, in the worst case, they are equivalent to arbitrary disjunctions [IV89].

Example 3 Assume we have a set of monotonic clauses $C = \{C_1, \ldots, C_n\}$ each with 3 literals such that in each clause, all the literals are either positive or negative but not both.. We construct a disjunctive database as follows. For each clause $C_i = l_1 \vee l_2 \vee l_3$, we have a disjunctive fact in the database $p_1 \vee p_2 \vee p_3$ where p_i = true(x) if l_i is a positive literal x, and p_i = false(x) if l_i is a negative literal of the form $\neg x$. Let $Q(X)$ = true(X),false(X). The query Q encodes the complement of the monotonic 3-satisfiability problem. If the answer to the query is *yes*, then the set C of monotonic clauses is not satisfiable. Thus, for a very simple *yes/no* query of conjunctions of atoms in a database with restricted disjunctions but no rules, the computation is intractable.

Conjunctive queries in deductive databases represent natural joins in relational algebra. Therefore, even a fundamental operation like join is CoNP-complete when general disjunctions are allowed in relational databases. The natural question that arises here is whether there are interesting classes of queries which can always be processed in polynomial time in a database with disjunctive information or whether there are subclasses of disjunctive databases where general queries can be computed in polynomial time.

Cadoli and Lenzerini [CL90] identify one such tractable subclass of disjunctive logic programs, where at most two literals (either positive or negative) are allowed in each clause (they refer to such clauses as *Krom clauses*). In [Dal92] Dalal notices that disjunctive logic programs with at most two atoms in a clause are tractable, which are in a subclass of the Krom class. However, they do not present practical and efficient algorithms to evaluate the queries. In the next section, we show how answers to some queries can be computed efficiently in polynomial time in a 2-disjunctive database where at most two atoms are allowed in a disjunction.

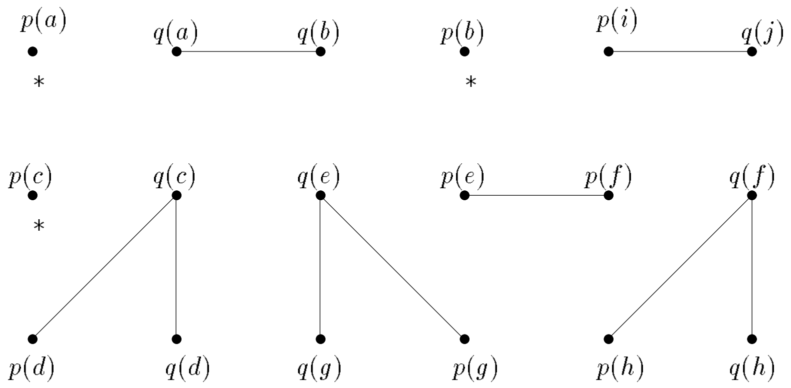


Figure 1: The D-graph $G_D B$ that represents the 2-disjunctive database of Example 3.

3 2-Disjunctive Deductive Databases

We consider a subclass of disjunctive databases we call 2-disjunctive databases in which there are no rules and at most two atoms are allowed in any ground disjunction. 2-Disjunctive databases are a subclass of the Krom class which has been identified as tractable. We represent the database by a graph and show how some queries can be evaluated in polynomial time by computing paths in the graph.

Definition 3 A disjunctive deductive database is called a 2-disjunctive database iff all the ground disjunctions in DB contain at most 2 atoms and there are no rules (i.e., $I_{DB} = \emptyset$).

Definition 4 Let DB be a 2-disjunctive database. The D-graph G_{DB} that represents the database DB is a tuple $\langle V, D, f, mark\rangle$ where V is the set of vertices, D is the set of undirected edges called *d-edges*, $f : V \longrightarrow H_{DB}$ is the labelling function and $mark : V \longrightarrow \{true, false\}$ is the marking function such that

1. H_{DB} is the set of all atoms belonging to DB.
2. f is one-to-one and onto.
3. $D = \{(m, n) \mid$ the 2-disjunctive clause $f(m) \vee f(n)$ belongs to $DB\}$.
4. $mark(n) = true$ iff the definite clause $f(n)$ belongs to DB, $false$ otherwise.

A node n in the D-graph G_{DB} for which $mark(n) = true$ is called a *marked node* and is denoted with an asterisk in the pictorial representation of the D-graph.

Example 4 Let DB be the following 2-disjunctive database:

$p(a)$	$p(b)$
$p(c)$	$q(a) \vee q(b)$
$q(c) \vee q(d)$	$q(c) \vee p(d)$
$p(e) \vee p(f)$	$q(e) \vee q(g)$
$q(e) \vee p(g)$	$q(f) \vee q(h)$
$q(f) \vee p(h)$	$p(i) \vee q(j)$

Figure 1 shows the D-graph G_{DB} that represents the 2-disjunctive database DB.

Let DB be a 2-disjunctive database represented by the D-graph G_{DB}. For simplicity, we assume that DB contains only two different types of predicates p and q (with possibly mixed 2-disjunctions). Queries to DB

can be answered by using the D-graph G_{DB}. For simple selection and projection queries, the computation is straight-forward. However, computing queries that involve natural join is more complex. In the rest of this section, we describe how we can compute simple selection-projection queries, join queries with at most 2 atoms, and negated ground queries in DB in polynomial time by using the D-graph G_{DB}. We assume that the predicates have only one argument, although our methods can be easily extended for more general cases.

Answering selection/projection queries Queries with only one atom in their body are called selection/projection queries. Answers to such queries in a 2-disjunctive database DB can be computed using the D-graph G_{DB} that represents DB.

Lemma 1 Let $Q = p(X)$ be a selection/projection query, and let $\theta, \theta_1, \theta_2$ be ground substitutions.

1. θ is a definite answer to Q iff $\exists n \in V$ such that $mark(n) = true$ and $Q\theta = f(n)$.
2. $\theta_1 \vee \theta_2$ is a disjunctive answer to Q iff $\exists (m, n) \in D$ such that $mark(m) = false$, $mark(n) = false$, $Q\theta_1 = f(m)$ and $Q\theta_2 = f(n)$.
3. There are no other minimal answers.

Answering 2-join queries Queries defined by a conjunction of two atoms are called 2-join queries. Let $Q = p(X) \wedge q(X)$ be the 2-join query to a 2-disjunctive database DB that we want to evaluate. We need some additional tools to evaluate this query.

Definition 5 A Q-graph $G_{DB\cup\{\leftarrow Q\}}$ of a D-graph $G_{DB} = \langle V, D, f, mark\rangle$ that represents a 2-disjunctive database DB with respect to a 2-join query $Q = p(X) \wedge q(X)$ is a tuple $\langle V, D, J, f, mark, S\rangle$ where J is a set of edges called *j-edges* and S is a function that maps j-edges to ground substitutions such that

1. $J = \{(m, n) \mid \exists$ a ground substitution θ and $Q\theta = f(m) \wedge f(n)\}$.
2. $S((m, n)) = \theta$ iff $Q\theta = f(m) \wedge f(n)$ where θ is a ground substitution.

In the pictorial representation of a Q-graph, d-edges are shown with straight lines and j-edges are shown with dotted lines. Notice that both types of edges may be present at the same time between a pair of nodes m and n in a Q-graph.

Example 5 Figure 2 shows the Q-graph that represents $G_{DB\cup\{\leftarrow Q\}}$ with respect to the 2-join query $Q = p(X) \wedge q(X)$.

Answers to the 2-join query Q are related to different types of paths in the Q-graph $G_{DB\cup\{\leftarrow Q\}}$.

Definition 6 A path $P = (n_1, n_2, \ldots, n_{2k})$ is called a *simple Q-path* in the Q-graph $G_{DB\cup\{\leftarrow Q\}}$ iff

1. $\forall i \neq j$, $n_i \neq n_j$, nodes n_1 and n_{2k} are marked and all other nodes are not marked.
2. $\forall i$ such that $1 \leq i < 2k$, if i is odd, then (n_i, n_{i+1}) is a j-edge in the Q-graph.
3. $\forall i$ such that $1 \leq i < 2k$, if i is even, then (n_i, n_{i+1}) is a d-edge in the Q-graph.

Definition 7 A path $P = (n_1, n_2, \ldots, n_{2k}, n_j, n_{j-1}, \ldots, n_1)$ is a *cyclic Q-path* in the Q-graph $G_{DB\cup\{\leftarrow Q\}}$ iff

1. n_{2k} is not marked, but marking n_{2k} gives a simple Q-path $P' = (n_1, n_2, \ldots, n_{2k})$.
2. $1 < j < 2k$, (n_{2k}, n_j) is a d-edge and (n_{j-1}, n_j) is a j-edge in the Q-graph.

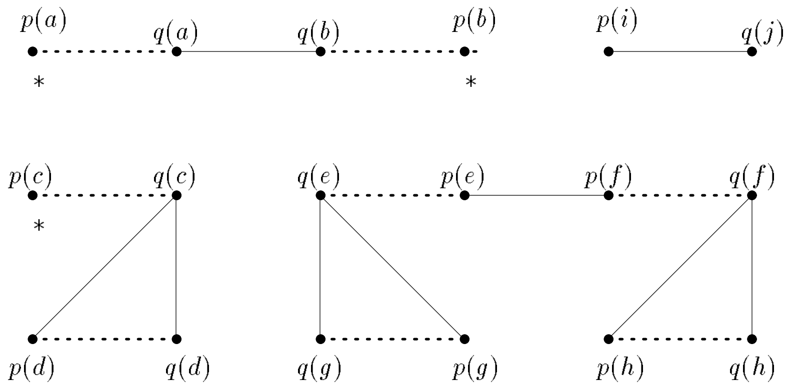


Figure 2: The Q-graph $G_{DB\cup\{\leftarrow Q\}}$ with respect to 2-join query Q.

Definition 8 A path $P = (n_1, n_2, \ldots, n_{2i}, n_j, \ldots, n_1, m_1, \ldots, m_{2k}, m_l, \ldots, m_1)$ is a *complex Q-path* in the Q-graph $G_{DB\cup\{\leftarrow Q\}}$ iff

1. n_1 is not marked, but marking n_1 gives a cyclic Q-path $P' = (n_1, n_2, \ldots, n_{2i}, n_j, \ldots, n_1)$.
2. m_1 is not marked, but marking m_1 gives a cyclic Q-path $P'' = (m_1, m_2, \ldots, m_{2k}, m_l, \ldots, m_1)$.
3. (m_1, n_1) is a d-edge and (m_1, n_1) is not a j-edge in the Q-graph.

Example 6 Figure 3 shows (a)a simple Q-path, (b)a cyclic Q-path and (c)a complex Q-path in the Q-graph of Example 2.

Definition 9 An *s-set of a Q-path* P is the set of all ground substitutions $S((m,n))$ such that (m,n) is a j-edge in the path P.

Example 7 The *s-set*s of the paths in Figure 5 are (a)$\{a, b\}$, (b)$\{c, d\}$ and (c)$\{e, f, g, h\}$.

Edges in Q-paths represent resolution steps in the proof of the 2-join query Q in DB. Therefore, each Q-path represents the computation towards a particular answer. The following theorems relate answers to a 2-join query Q with the s-sets of the Q-paths defined above.

Lemma 2 If $P = (n_1, n_2, \ldots, n_{2k})$ is a simple Q-path with s-set$(P) = \{x_1, \ldots, x_k\}$ then $X = x_1 \vee X = x_2 \vee \ldots X = x_k$ is an answer to the query $Q = p(X) \wedge q(X)$.

Proof: (by induction on k)

Base Case: $k = 1$.

The only possible simple Q-path is (n_1, n_2) where either $f(n_1) = p(x_1)$ and $f(n_2) = q(x_1)$ or $f(n_1) = q(x_1)$ and $f(n_2) = p(x_1)$ and both n_1 and n_2 are marked. Therefore, $DB \models p(x_1)$ and $DB \models q(x_1)$, which implies $DB \models p(x_1) \wedge q(x_1)$ i.e., $X = x_1$ is an answer to the query $Q = p(X) \wedge q(X)$.

Induction Step:

Assume that if $P = (n_1, n_2, \ldots, n_{2k})$ is a simple Q-path with s-set$(P) = \{x_1, \ldots, x_k\}$ then $X = x_1 \vee X = x_2 \vee \ldots X = x_k$ is a minimal answer to the query $Q = p(X) \wedge q(X)$.

Let $P_1 = (m_1, m_2, \ldots, m_{2k}, m_{2k+1}, m_{2k+2})$ be a simple Q-path in the graph with s-set(P_1)=$\{x_1, \ldots, x_k, x_{k+1}\}$. By the definition of a simple path, (m_2, m_3) is a d-edge in the Q-graph. Without loss of generality, let us assume that $f(m_2) = q(x_1)$ and $f(m_3) = p(x_2)$. Since (m_2, m_3) is a d-edge in the Q-graph,

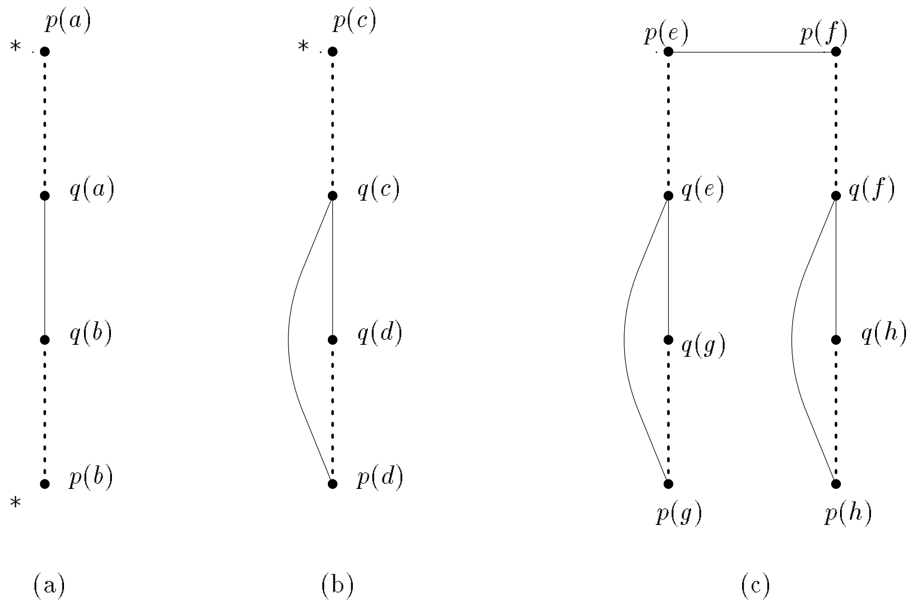


Figure 3: (a) a simple Q-path, (b) a cyclic Q-path, (c) a complex Q-path.

$q(x_1) \vee p(x_2)$ must be a disjunction in the database DB. Clearly, $DB \cup \{q(x_1)\} \models p(x_1) \wedge q(x_1)$ because $(f^{-1}(p(x_1)), f^{-1}(q(x_1)))$ is a simple Q-path in the graph of $DB \cup \{q(x_1)\}$. Similarly, in the graph of $DB \cup \{p(x_2)\}$, $P' = (m_3, m_4, \ldots, m_{2k+2})$ is a simple Q-path with s-set$(P') = \{x_2, \ldots, x_{k+1}\}$; therefore, by the induction hypothesis, $DB \cup \{p(x_2)\} \models (p(x_2) \wedge q(x_2)) \vee \ldots \vee (p(x_{k+1}) \wedge q(x_{k+1})$. Therefore, $DB \cup \{q(x_1) \vee p(x_2)\} \models (p(x_1) \wedge q(x_1)) \vee (p(x_2) \wedge q(x_2)) \vee \ldots \vee (p(x_{k+1}) \wedge q(x_{k+1})$ which implies $X = x_1 \vee \ldots \vee X = x_{k+1}$ is an answer to the query Q in the database DB.

Lemma 3 If $P = (n_1, n_2, \ldots, n_{2k}, n_j, \ldots, n_1)$ is a cyclic Q-path with s-set$(P) = \{x_1, \ldots, x_k\}$ then $X = x_1 \vee X = x_2 \vee \ldots X = x_k$ is an answer to the query $Q = p(X) \wedge q(X)$.

Proof:

By the definition of a cyclic Q-path, (n_{2k}, n_j) is a d-edge in the Q-graph. Without loss of generality, let us assume that $f(n_{2k}) = q(x_k)$ and $f(n_j) = q(x_{\frac{j}{2}})$. Since (n_{2k}, n_j) is a d-edge in the Q-graph, $q(x_k) \vee q(x_{\frac{j}{2}})$ must be a disjunction in the database DB. Clearly, in the Q-graph of the database $DB \cup \{q(x_{\frac{j}{2}})\}$, $P' = (n_1, n_2, \ldots, n_j)$ is a simple Q-path with s-set$(p') = \{x_1, \ldots, x_{\frac{j}{2}}\}$; therefore, by 2, $DB \cup \{p(x_j)\} \models (p(x_1) \wedge q(x_1)) \vee \ldots \vee (p(x_{\frac{j}{2}}) \wedge q(x_{\frac{j}{2}})$. Similarly, in the Q-graph of $DB \cup \{p(x_k)\}$, $P' = (n_1, n_2, \ldots, m_{2k})$ is a simple Q-path with s-set$(P') = \{x_1, \ldots, x_k\}$; therefore, by 2, $DB \cup \{q(x_k)\} \models (p(x_1) \wedge q(x_1)) \vee \ldots \vee (p(x_k) \wedge q(x_k)$. Therefore, $DB \cup \{q(x_k) \vee p(x_{\frac{j}{2}})\} \models (p(x_1) \wedge q(x_1)) \vee (p(x_2) \wedge q(x_2)) \vee \ldots \vee (p(x_k) \wedge q(x_k))$ which implies $X = x_1 \vee \ldots \vee X = x_k$ is an answer to the query Q in the database DB.

Lemma 4 If $P = (n_1, n_2, \ldots, n_{2i}, n_j, \ldots, n_1, m_1, m_2, \ldots, m_{2k}, m_l, \ldots, m_2, m_1)$ is a complex Q-path with s-set$(P) = \{x_1, \ldots, x_h\}$ then $X = x_1 \vee X = x_2 \vee \ldots X = x_h$ is an answer to the query $Q = p(X) \wedge q(X)$.

Proof:

By the definition of a complex Q-path, (n_1, m_1) is a d-edge in the graph. Without loss of generality, let us assume that $f(n_1) = p(x_i)$ and $f(m_1) = q(x_k)$ where $x_i \neq x_k$. Since (n_1, m_1) is a d-edge in the Q-graph, $p(x_i) \vee q(x_k)$ must be a disjunction in the database DB. Clearly, in the D-graph of the database $DB \cup \{p(x_i)\}$, $P' = (n_1, n_2, \ldots, n_{2i}, n_j, \ldots, n_1)$ is a simple Q-path with s-set$(P') = \{x_1, \ldots, x_i\}$; therefore, by 3, $DB \cup \{p(x_i)\} \models (p(x_1) \wedge q(x_1)) \vee \ldots \vee (p(x_i) \wedge q(x_i))$. Similarly, in the Q-graph of $DB \cup \{q(x_k)\}$,

$P'' = (m_1, m_2, \ldots, m_{2k}, m_l, \ldots, m_1)$ is a simple Q-path with s-set$(P'') = \{x'_1, \ldots, x'_k\}$; therefore, by 3, $DB \cup \{q(x_k)\} \models (p(x'_1) \wedge q(x'_1)) \vee \ldots \vee (p(x'_k) \wedge q(x'_k)$. Since s-set$(p)$ = s-set(p') $\cup$ s-set(p''), $DB \cup \{q(x_k) \vee p(x_i)\} \models (p(x_1) \wedge q(x_1)) \vee (p(x_2) \wedge q(x_2)) \vee \ldots \vee (p(x_h) \wedge q(x_h))$ which implies $X = x_1 \vee \ldots \vee X = x_h$ is an answer to the query Q in the database DB.

Theorem 1 If P is a simple, cyclic or complex Q-path with s-set$(P) = \{x_1, \ldots, x_k\}$ in the Q-graph of a 2-disjunctive database DB with respect to a 2-join query $Q = p(X) \wedge q(X)$, then $X = x_1 \vee X = x_2 \vee \ldots X = x_k$ is an answer to the query Q.

Proof: by Lemmas 2, 3 and 4.

Theorem 2 If $X = x_1 \vee X = x_2 \vee \ldots \vee X = x_k$ is a minimal answer to the 2-join query $Q = p(X) \wedge q(X)$ in a 2-disjunctive database DB, then there is a simple, cyclic or complex Q-path P in the Q-graph of DB with respect to the query Q such that s-set$(P) = \{x_1, x_2, \ldots, x_k\}$.

Proof: Since $DB \models (p(x_1) \wedge q(x_1)) \vee (p(x_2) \wedge q(x_2)) \vee \ldots (p(x_k) \wedge q(x_k))$, all of the following must be true in DB:

$DB \models p(x_1) \vee p(x_2) \ldots \vee p(x_k)$
$DB \models q(x_1) \vee p(x_2) \ldots \vee p(x_k)$
$DB \models p(x_1) \vee p(x_2) \ldots \vee p(x_k)$
$DB \models p(x_1) \vee q(x_2) \ldots \vee p(x_k)$
$\vdots$
$DB \models q(x_1) \vee q(x_2) \ldots \vee p(x_k)$
$DB \models q(x_1) \vee q(x_2) \ldots \vee q(x_k)$

Since, only 2-disjunctions are allowed in DB, all of the above must be implied by definite and 2-disjunctive facts only. There could be three cases:

Case 1: There are 2 definite facts (without loss of generality, let us assume $p(x_i)$ and $q(x_j)$ where $x_i \neq x_j$). It can be shown that the Q-graph of DB has a simple Q-path P where $f^{-1}(p(x_i))$ is the start node and $f^{-1}(q(x_j))$ is the end node of P.

Case 2: There is a definite fact (without loss of generality, let us assume $p(x_i)$). It can be shown that the Q-graph of DB has a cyclic Q-path P where $f^{-1}(p(x_i))$ is the start node of P.

Case 3: There is no $i \in \{1, 2, \ldots, k\}$ such that $p(x_i)$ or $q(x_i)$ is true in DB. It can be shown that the Q-graph of DB has a complex Q-path P where s-set$(P) = \{x_1, x_2, \ldots, x_k\}$.

The above theorems are the basis for our algorithm to compute *generative* answers to 2-join queries in 2-disjunctive databases by traversing the Q-graphs and computing Q-paths. The traversal uses a technique we call *alternate-edge* depth-first-search which is a variation of the usual depth-first-search technique for undirected graphs and runs in polynimial time with respect to the size of the database. The algorithm is presented in section 4.

Answering negated ground queries Queries with only one negated ground atom in their body are called negated ground queries. Using the GCWA as the semantics for negation, answers to such queries in a 2-disjunctive database DB can be computed in polynomial time.

Lemma 5 Let $Q = \neg p(a)$ be the negated ground query that we want to evaluate using GCWA.

1. If $\exists n \in V$ such that $f(n) = p(a)$ and $mark(n) = true$ then Q is *false* in DB.
2. If $\exists (n, m) \in D$ such that $f(n) = p(a)$, $mark(m) = false$ and $mark(n) = false$ then Q is *unknown* in DB.
3. Otherwise, Q is *true* in DB.

4 Algorithms

In this section, we present the algorithms to compute *yes/no* answers and *generative* answers to 2-join queries in 2-disjunctive databases by traversing the Q-graphs and computing Q-paths. The traversal uses a technique we call *alternate-edge* depth-first-search which is a variation of the usual depth-first-search technique for undirected graphs. In an *altrnate-edge* depth-first-search, if a node is visited through a d-edge, then its j-neighbor only is visited from that node; similarly if a node is visited through a j-edge, then its d-neighbors only are visited from that node. Search is stopped when a marked node is visited through a j-edge or when an appropriate back-edge is found.

Before we describe the algorithms, we need to define some operators used in Q-graph traversals.

mark(n**)** deletes all d-edges incident on node n and saves them for future reconstruction and sets marked[n] to true.

unmark(n**)** re-inserts all d-edges that were deleted and stored by a previous mark(n) operation and sets marked[n] to false.

d-neighbors(n**)** returns the set of all nodes m such that (m, n) is a d-edge in the Q-graph.

j-neighbor(n**)** returns node m if (m, n) is a j-edge in the Q-graph.

4.1 Computing *yes/no* answers

In this section, we present an algorithm to compute *yes/no* answers to a 2-join query $Q = p(X) \wedge q(X)$ in a 2-disjunctive database. We assume that the Q-graph is computed before the algorithm is invoked. The algorithm searches the Q-graph for a simple, cyclic or complex Q-path using the *alternate-edge* depth-first-search technique and returns *yes* if it can find any such Q-path. Otherwise, it returns *no*.

Algorithm 1: {gives an *yes/no* answer to a join query}

step 1. {look for a simple or cyclic Q-path starting at a marked node}
for each marked node n **do**
if there is a simple or cyclic Q-path starting at node n **then** return an *yes* answer and halt.

step 2. {look for a complex Q-path }
for each unused d-edge (m, n) where (m, n) is not a j-edge **do**
if mark(m) and **mark**(n) result in new simple or cyclic Q-paths starting at nodes m and n **then** a complex Q-path has been found, return an *yes* answer and halt.

step 3. {no simple, cyclic or complex Q-path was found in steps 1 and 2}
return a *no* answer.

Algorithm 1.a:{checks if there is a simple or cyclic Q-path beginning at node n}

1. if node n has no j-neighbor **then** return *no*.

2. begin an *alternate-edge* depth-first-search by visiting the j-neighbor of node n through a j-edge. The search performs successfully if a marked node is ever visited through a j-edge or a node is ever visited through a j-edge such that one of its d-neighbors has already been visited through a j-edge.
if the search succeeds **then** return *yes* **else** return *no*.

The above algorithm can be used to check if a given answer is indeed correct or not. All we have to do is modify the algorithm so that during the search through the graph, it only visits those nodes whose values are contained in the given answer.

Lemma 6 Algorithm 1 runs in polynomial time.

4.2 Computing *generative* answers

We now present an algorithm to compute the set of all answers to a 2-join query in a 2-disjunctive database. We assume that the Q-graph is computed before the algorithm is invoked. The algorithm stores in the set $ANSWER$ the s-sets of all simple, cyclic and complex Q-paths in the Q-graph and returns this set. The actual algorithm that is used to compute the s-sets of all simple and cyclic Q-paths beginning at a specified node n is presented in the Appendix. We refer to the Appendix for a detailed discussion of the *alternate-edge depth-first-search* algorithm and how it is used to compute the s-sets of all simple and cyclic Q-paths beginning at a given node.

Algorithm 2: {returns the set of all answers to a 2-join query}

Initially, $ANSWER \leftarrow \{\emptyset\}$

step 1. {find all simple and cyclic Q-paths and compute s-sets of these Q-paths}
for each marked node n **do**

1.1 $S \leftarrow \{s \mid P$ is a simple or cyclic Q-path beginning at node n and $s =$ s-set$(P)\}$

1.2 $ANSWER \leftarrow ANSWER \cup S$

step 2. {find all complex Q-paths and compute s-sets of these Q-paths}
for each unused d-edge (m,n) where (m, n) is not a j-edge **do**

2.1 **mark**(m); $S_1 \leftarrow \{s \mid P$ is a simple or cyclic Q-path beginning at node m and $s =$ s-set$(P)\}$; **unmark**(m).

2.2 **mark**(n); $S_2 \leftarrow \{s \mid P$ is a simple or cyclic Q-path beginning at node n and $s =$ s-set$(P)\}$; **unmark**(n).

2.3 **if** $S_1 \neq \{\emptyset\}$ and $S_2 \neq \{\emptyset\}$ **then**

2.3.1 $S \leftarrow \{s_1 \cup s_2 | s_1 \in S_1 \wedge s_2 \in S_2\}$

2.3.2 $ANSWER \leftarrow ANSWER \cup S$

2.4 mark d-edge (m,n) as used

step 3. {no more Q-paths in the Q-graph}
return $ANSWER$

Lemma 7 Algorithm 2 runs in polynomial time.

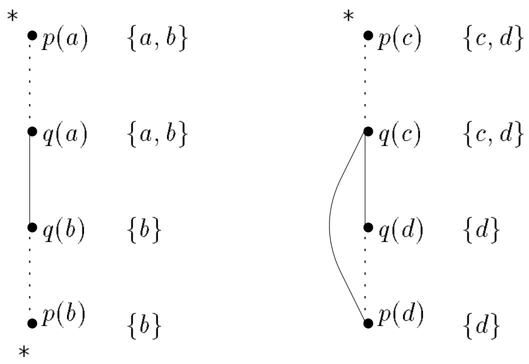
Proof: The worst-case running time of an *alternate-edge* depth-first-search is the same as the worst-case case running time of a depth-first-search in an undirected graph which is $\Theta(n + e)$. Since one such search is performed at every marked node and at nodes that are connected by unused d-edges, the running time of Algorithm 2 is bounded by $\Theta(max(n, e) * (n + e))$.

Example 8 Figure 4 shows how Algorithm 2 travarses the D-graph of example 2 to compute all simple, cyclic and complex paths and their s-sets..

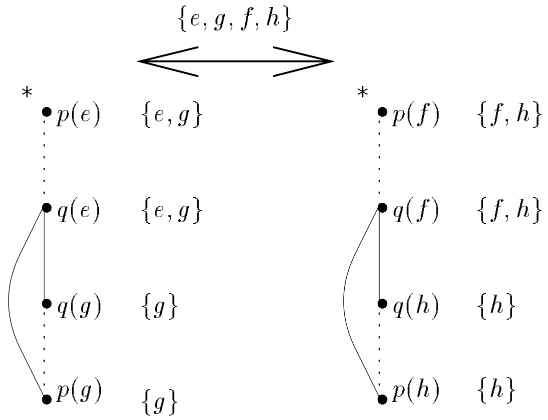
5 2-Disjunctive Databases with Rules

In this section, we identify some classes of rules in a hierarchical disjunctive deductive database with 2-disjunctions that are tractable. In a hierarchical disjunctive deductive database, it is possible to establish an evaluation order for rules in the databases such that each rule needs to be evaluated only once during the computation. The rules that define a predicate p appear in the order prior to a rule that uses p. We classify tractable rules in a hierarchical 2-disjunctive deductive database and present polynomial time algorithms to evaluate rules in each class.

Step 1:



Step 2:



Step 3:

return $ANSWER = \{\{a, b\}, \{c, d\}, \{e, f, g, h\}\}$

Figure 4: Paths and s-sets in the D-graph of example 2 computed by Algorithm 2

5.1 Safe Selection Rules

Rules with only one atom in their head and one atom in their body are called safe selection rules. Evaluation of a safe selection rule $s \leftarrow p$ in a graph G_{DB} that represents a 2-disjunctive database DB yields a larger graph $G_{DB \cup \{s \leftarrow p\}}$ which contains the old graph G_{DB}.

Algorithm: Computes $G_{DB \cup \{s \leftarrow p\}}$ from G_{DB} and a safe selection rule $(s \leftarrow p)$.

For each node n such that $\exists \theta$ ground, $f(n) = p\theta$ do

step 1. create a new node n' with $f(n') = s\theta$ and set new$[n] = n'$

step 2. if n is marked or $\exists m \in$ d-neighbor(n) such that $\exists \theta'$ ground, $f(m) = p\theta'$ and $f(n') = s\theta'$ then mark node n'.

else for each $m \in$ d-neighbor(n) do

create d-edge (n', m)

if $f(m) = p$ then create d-edge $(n'$,new$[m])$

Example 9 Let DB be the following 2-disjunctive database:

$p(a, f)$	$q(f)$
$p(b, g) \vee p(b, h)$	$q(g)$
$p(c, f) \vee p(d, h)$	$q(h)$

The graph G_{DB} that represents DB and $G_{DB \cup \{s(X) \leftarrow p(X,Y)\}}$ after the evaluation of the safe selection rule $\{s(X) \leftarrow p(X, Y)\}$ are shown in Figure 5(a) and 5(b).

5.2 Safe Join Rules

Rules with only one atom in their head and two atoms in their body where one predicate is definite and the other is disjunctive are called safe join rules. Evaluation of a safe selection rule $s \leftarrow p, q$ in a graph G_{DB} that represents a 2-disjunctive database DB yields a larger graph $G_{DB \cup \{s \leftarrow p,q\}}$ which contains the old graph G_{DB}.

Algorithm: Computes $G_{DB \cup \{s \leftarrow p,q\}}$ from G_{DB} and a safe join rule $(s \leftarrow p, q)$ where p is disjunctive and q is definite.

For each node n such that $\exists \theta_1$ ground, $f(n) = p\theta_1$ do if there is a node m such that $\exists \theta_2$ ground, $f(m) = q\theta_1\theta_2$

step 1. create a new node n' with $f(n') = s\theta_1\theta_2$ and set new$[n] = n'$

step 2. for each $m \in$ d-neighbor(n) do

create d-edge (n', m)

if $f(m) = p$ then if $f($new$[m] = f(n')$ then mark n' else create d-edge $(n'$,new$[m])$

Example 10 The D-graph $G_{DB \cup \{s(X) \leftarrow p(X,Y), q(Y)\}}$ after the evaluation of the safe selection rule $\{s(X) \leftarrow p(X, Y), q(Y)\}$ are shown in Figure 5(c).

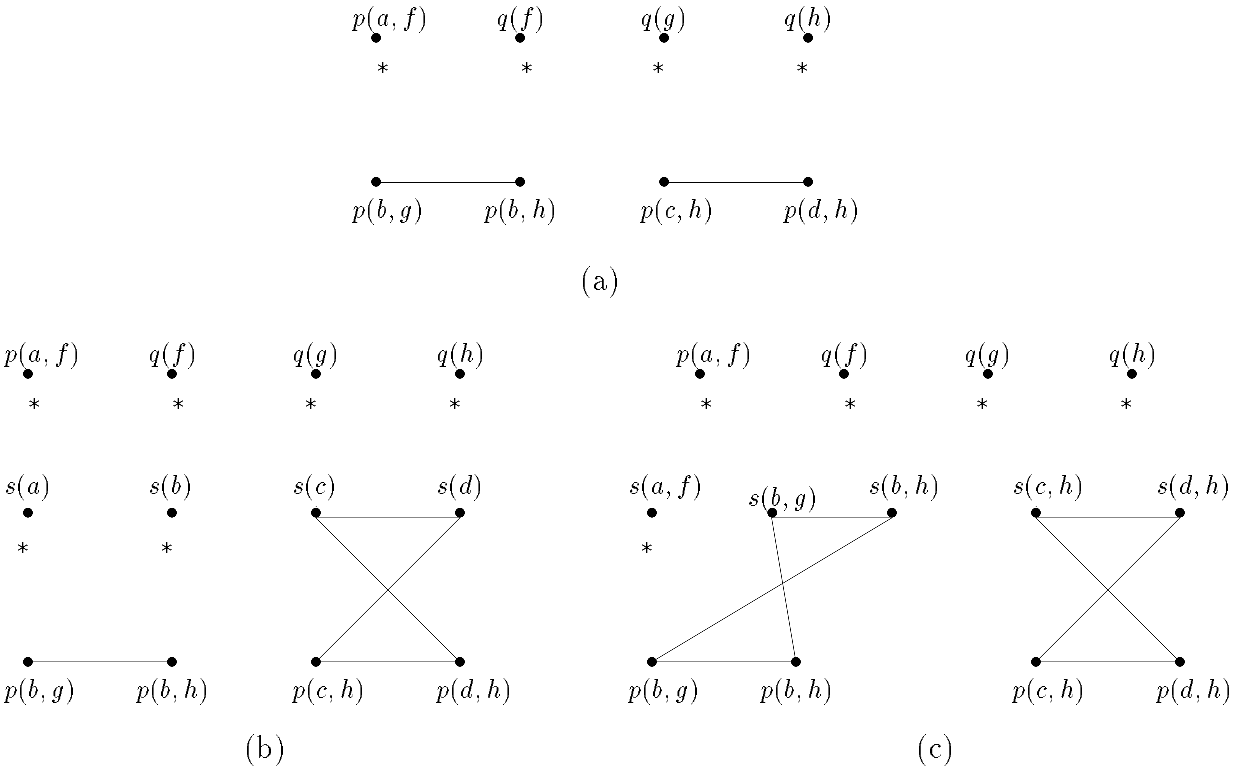


Figure 5: (a) The D-graph G_{DB} of the Database in Example (b) The D-graph $G_{DB \cup \{s(X) \leftarrow p(X,Y)\}}$, (c) The graph $G_{DB \cup \{s(X) \leftarrow p(X,Y), q(Y)\}}$.

6 Conclusion

In this paper, we identified a tractable class of disjunctive deductive databases. We defined 2-disjunctive databases and presented polynomial time algorithms to compute general queries in such databases. One advantage of our algorithm is that by introducing a parameter k, we can control the depth of the graph search and thus the user can get a better response time by specifying the length of disjunctive answers if he is interested in shorter answers only. We also classified some classes of rules that are evaluable in polynomial time and presented algorithms to evaluate them. Our future research would be on finding larger and more powerful subclasses of disjunctive deductive databases that are tractable.

References

[CL90] M. Cadoli and M. Lenzerini. The complexity of closed world reasoning and circumscription. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 550–555. MIT Press, July 1990.

[Dal92] M. Dalal. Some tractable classes of disjunctive logic programs. Technical report, Rutgers University, Department of Computer Science, 1992.

[Dem89] R. Demolombe. An efficient evaluation strategy for non-horn deductive databese. In *Proceedings of IFIP*, 1989.

[FM91] J. A. Fernandez and J. Minker. Bottom-up evaluation of hierarchical disjunctive deductive databases. In K. Furukawa, editor, *Logic Programming proceedings of the 8th International Conference*, pages 660–675. MIT Press, 1991.

[GM86] J. Grant and J. Minker. Answering queries in indefinite databases and the null value problem. In P. Kanellakis, editor, *Advances in Computing Research*, pages 247–267. 1986.

[GM92] J. Grant and J. Minker. The impact of logic programming on databases. *Communications of the ACM*, 35(3):66–81, March 1992.

[HP88] L. J. Henschen and H. Park. Compiling the gcwa in indefinite database. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 395–438. Morgan Kaufmann Pub., Washington, D.C., 1988.

[Imi91] T. Imielinsky. Incomplete deductive databases. *Annals of Mathematics and Artificial Intelligence*, 3:259–294, 1991.

[IV89] T. Imielinsky and K. Vadaparty. Complexity of query processing in databases with or-objects. In *Proceedings of the 7th ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 51–65, Philadelphia, Pennsylvania, March 1989.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.

[LS91] K.C. Liu and R. Sunderraman. A generalized relational model for indefinite and maybe information. *IEEE Transactions on Knowledge and Data Engineering*, 3(1):65–77, 1991.

[LYW92] J. Lobo, C. Yu, and G. Wang. Computing the transitive closure in disjunctive databases. Technical report, University of Illinois at Chicago, Department of Electrical Engineering and Computer Science, 1992.

[Var86] M. Vardi. Querying logical databases. *Journal of Computing and Systems Sciences*, (33):142–160, 1986.

[YC89] L. Y. Yuan and D.A. Chiang. A sound and complete query evaluation algorithm for relational databases with disjunctive information. In *Proceedings of the 8th Symposium on Principles of Database Systems*, pages 66–74, March 1989.

[YH85] A. Yahya and L. J. Henschen. Deduction in non-horn databases. *Journal of Automated Reasoning*, 1(2):141–160, 1985.

A Appendix

Before we describe the algorithm, we need to define the data structures and operators used in Q-graph traversals.

$$\text{visited}[n] = \begin{cases} \text{true} & \text{if node } n \text{ is marked and has been visited} \\ \text{d-edge} & \text{if node } n \text{ is not marked and has been visited through a d-edge} \\ \text{j-edge} & \text{if node } n \text{ is not marked and has been visited through a j-edge} \\ \text{false} & \text{if node } n \text{ has not been visited yet} \end{cases}$$

p-sets[n]: set of partial sets where each partial-set is a subset of an s-set of a simple or cyclic Q-path that contains node n.
initialize(): for all nodes n p-sets[n] is set to $\{\emptyset\}$, visited[n] and done[n] are set to *false*.

Algorithm: computes s-sets of all simple and cyclic Q-paths starting at node n in a Q-graph $G_{DB\cup\{\leftarrow Q\}} = \langle V, D, J, f, mark, S\rangle$.

0. initialize()

1. *S-SETS*$\leftarrow \{\emptyset\}$

2. **if** $\exists(n,m) \in J$ **then**

alternate-edge-dfs(m,n,j-edge)

S-SETS$\leftarrow$p-sets[m]

3. **return***(S-SETS)*

procedure *alternate-edge-dfs(m,n,edge-type)*
{visits node m from node n, performs a depth-first search and stores in p-sets[m] the set of partial s-sets of Q-paths that contain node m}

0. *P-SETS*$\leftarrow \{\emptyset\}$

1. visited[m]$\leftarrow$*edge-type*

2. **if** *edge-type* = d-edge and $\nexists(m,n) \in J$ **then**

alternate-edge-dfs(j-neighbor(m),m,j-edge)

P-SETS$\leftarrow$p-sets[j-neighbor(m)]

3. **if** *edge-type* = j-edge **then**

if *mark*(n) **then** *P-SETS*$\leftarrow \{\{S((m,n))\}\}$

else for each $v \in$ d-neighbors(m) **do**

if visited[v] = j-edge and done[v] = false **then** *P-SETS*$\leftarrow$*P-SETS*$\cup\{\{S((m,n))\}\}$

if visited[v] = d-edge and done[v] = true and p-sets[v] $\neq \{\emptyset\}$ **then**

P-SETS$\leftarrow$*P-SETS*$\cup\{s \cup \{S((m,n))\} \mid s \in$ p-sets[v]$\}$

else if visited[v] = false **then**

alternate-edge-dfs(v, m,d-edge)

P-SETS$\leftarrow$*P-SETS*$\cup\{s \cup \{S((m,n))\} \mid s \in$ p-sets[v]$\}$

4. p-sets[m]$\leftarrow$*P-SETS*

5. done[m]$\leftarrow$ true

6. **return()**