18. The DISCO System

Peter Revesz

18.1 Introduction

DISCO, short for DATALOG *with Integer Set COnstraints*, is a constraint database system that implements DATALOG with *Boolean constraints* (see Chapter 7) on set variables that range over finite and infinite sets of integers. In this chapter we provide a short introduction to the DISCO system.

18.2 DISCO Queries

The syntax of DISCO is basically that of DATALOG, with a restricted form of recursion that guarantees termination. The domain of each DISCO variable is an element of the Boolean algebra in which 0 is the empty set, 1 is the set of integers, $\wedge$ is set intersection, $\vee$ is set union, and $'$ is set complement. The precedence operator $\leq$ is defined as the subset relation between sets.

Example 18.2.1. Consider the packet switching network in Figure 18.2.1. In this hierarchical network there are four layers each with four nodes. Each node represents a router and the directed edges represent connections across which packets can be sent. A set of packets arrive at the top layer and need to be dynamically routed to given nodes of the bottom layer, with some restrictions on the routing in the middle layers.

The connections of the network imply restrictions which can be represented by the relation `CONNECT`. One example of a rule in `CONNECT`, encoding the connections between the first and second layers (the $\{1\}$ refers to the outgoing layer) in the figure, is:

$$\begin{aligned}\texttt{CONNECT}(\{1\}, A, B, C, D, E, F, G, H) \text{ :- } & E \leq A, F \leq A \vee B, \\ & G \leq B \vee D, H \leq C \vee D, \\ & P \leq L.\end{aligned}$$

Suppose that some packets are to be sent from nodes in the top layer to the bottom layer. For example:

$$\begin{aligned}&\texttt{FROM}(A, B, C, D) \text{ :- } A = \{1, 2, 3\}, B = \{4, 5\}, C = \{6, 7\}, D = \{8, 9\}. \\ &\texttt{TO}(M, N, O, P) \quad \text{ :- } M = \{1\}, N = \{2, 6, 8\}, O = \{3, 4\}, P = \{5, 7, 9\}.\end{aligned}$$

To find out where each packet may be sent, using a DISCO query, we can then write:

G. Kuper et al. (eds.), *Constraint Databases*

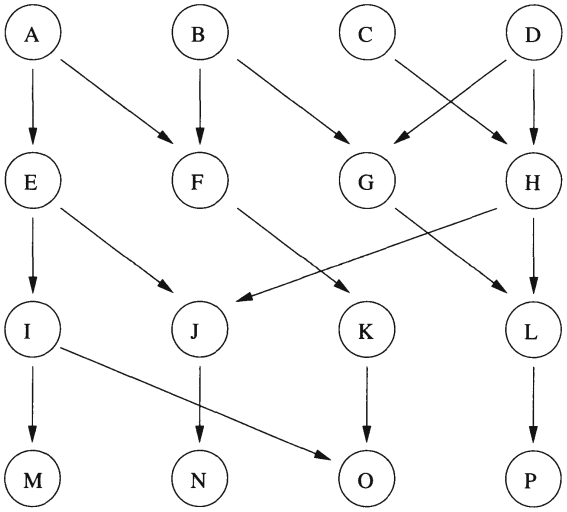

Fig. 18.2.1. A packet switching network

$$\texttt{PATH}(\{1\}, A, B, C, D) \texttt{ :- } \texttt{FROM}(A, B, C, D).$$
$$\begin{aligned}\texttt{PATH}(Z, V, W, X, Y) \texttt{ :- } & \texttt{PATH}(Z1, Q, R, S, T),\\ & \texttt{CONNECT}(Z1, Q, R, S, T, V, W, X, Y),\\ & \texttt{NEXT}(Z1, Z).\end{aligned}$$

where NEXT is the successor function, defined explicitly using rules such as

$$\texttt{NEXT}(\{1\}, \{2\}) .$$

Example 18.2.2. For another example, suppose that instead of requiring each packet to be sent to only one successor, we want each packet to be broadcast, that is, sent to all successors. Then the connections can be represented as follows:

$$\begin{aligned}\texttt{CONNECT}(\{1\}, A, B, C, D, E, F, G, H) \texttt{ :- } & A \leq E, A \leq F, B \leq F,\\ & B \leq G, C \leq H, D \leq G,\\ & D \leq H.\end{aligned}$$

with similar rules for the other two layers. The PATH query is the same as before.

18.3 Implementation

18.3.1 Converting to Relational Algebra

DATALOG rules are converted to relational algebra expressions in two different ways:

1. **EVAL function**. The translation of to relational algebra is standard except for a slightly different syntax for the selection operations.

2. **EVAL_INCR function**. This is similar to the EVAL, except that before conversion each rule is copied as many times as the number of defined relation symbols in its body. In the ith copy a Δ symbol is put before the ith defined relation.

Example 18.3.1. Consider Example 18.2.1. EVAL_INCR converts this query into:

$$\begin{aligned}\texttt{PATH}(\{1\}, A, B, C, D) \;&\texttt{:-}\; \texttt{FROM}(A, B, C, D).\\ \texttt{PATH}(Z, V, W, X, Y) \;&\texttt{:-}\; \Delta\texttt{PATH}(Z1, Q, R, S, T),\\ &\quad\; \texttt{CONNECT}(Z1, Q, R, S, T, V, W, X, Y),\\ &\quad\; \texttt{NEXT}(Z1, Z).\end{aligned}$$

This is translated into a relational expression as follows. The first rule is translated to:

$$\pi_{Z,A,B,C,D}\ (\sigma_{Z=\{1\}}\ \texttt{FROM}(A, B, C, D) \bowtie Z(Z))\ .$$

where $Z(Z)$ is the relation that has a single variable Z and no constraints.

The second rule is translated to:

$$\begin{aligned}&\pi_{Z,V,W,X,Y}\ \Delta\texttt{PATH}(Z1, Q, R, S, T) \bowtie \texttt{NEXT}(Z1, Z)\\ &\qquad \bowtie \texttt{CONNECT}(Z1, Q, R, S, T, V, W, X, Y)\ .\end{aligned}$$

The union of these expressions is a relational algebra expression for PATH.

DISCO queries are evaluated using the standard Naive and Semi-Naive methods for evaluating queries. This is illustrated by the following example.

Example 18.3.2. Example 18.3.1 is evaluated by DISCO as follows. The first iteration yields.

$$\begin{aligned}\texttt{PATH}(\{1\}, A, B, C, D) \;&\texttt{:-}\; A = \{1, 2, 3\}, B = \{4, 5\}, C = \{6, 7\},\\ &\quad\; D = \{8, 9\}.\end{aligned}$$

After the second iteration, we get (after renamings some variables for readability):

$$\begin{aligned}\texttt{PATH}(\{2\}, E, F, G, H) \;&\texttt{:-}\; E \leq \{1, 2, 3\}, F \leq \{1, 2, 3, 4, 5\},\\ &\quad\; G \leq \{4, 5, 8, 9\}, H \leq \{6, 7, 8, 9\}.\end{aligned}$$

We call a Boolean function g *monotone* if $x_i \leq y_i$, $i = 1, \ldots, n$ implies that $g(x_1, \ldots, x_n) \leq g(y_1, \ldots, y_n)$. A *monotone inequality constraint* is a constraint of the form $g(x_1, \ldots, x_n) \neq 0$ where g is monotone.

We now discuss the implementation of the DISCO algebra, for constraint databases and queries that contain only set-order and monotone inequality constraints. The key point is the following technique for eliminating existential quantifiers over such constraints.

Lemma 18.3.1. *Let $g_1, \ldots, g_l$ be monotone Boolean functions, and let y_i, z_i, v_i, w_i, u_i (for appropriate values of i) be constants or variables distinct from x (but not necessarily from each other). Then*

$$\begin{aligned}\exists x(&z_1 \leq x \wedge \cdots \wedge z_m \leq x \wedge x \leq y_1 \wedge \cdots \wedge x \leq y_k \\ &\wedge w_1 \leq u_1 \wedge \cdots \wedge w_s \leq u_s \\ &\wedge g_1(x, v_1, \ldots, v_n) \neq 0 \wedge \cdots \wedge g_l(x, v_1, \ldots, v_n) \neq 0)\end{aligned}$$

is equivalent to

$$\begin{aligned}&z_1 \leq y_1 \wedge \cdots \wedge z_j \leq y_i \wedge \cdots \wedge z_m \leq y_k \wedge w_1 \leq u_1 \wedge \cdots \wedge w_s \leq u_s \\ &\quad \wedge g_1((y_1 \wedge \cdots \wedge y_k), v_1, \ldots, v_n) \neq 0 \wedge \cdots \\ &\quad \wedge g_l((y_1 \wedge \cdots \wedge y_k), v_1, \ldots, v_n) \neq 0 \,.\end{aligned}$$

□

Example 18.3.3. Example 18.2.2 contains only set-order and monotone inequality constraints. The query is evaluated as follows

$$\begin{aligned}&\texttt{PATH}(\{1\}, A, B, C, D) &&:\!- B \wedge C \neq 0. \\ &\texttt{PATH}(\{2\}, E, F, G, H) &&:\!- F \wedge G \neq 0, F \wedge H \neq 0, G \wedge H \neq 0. \\ &\texttt{PATH}(\{3\}, I, J, K, L) &&:\!- J \wedge K \neq 0, J \wedge L \neq 0, K \wedge L \neq 0. \\ &\texttt{PATH}(\{4\}, M, N, O, P) &&:\!- N \wedge O \neq 0, N \wedge P \neq 0, O \wedge P \neq 0.\end{aligned}$$

The DISCO system also supports an interactive Semi-Naive evaluation method, where the user can decide whether to add new tuples at each step.

18.3.2 Optimization of Relational Algebra

DISCO uses the standard relational rewrite rules. The system pushes selections as far down the parse tree as possible, and then does the same with projection, performing selections before projections. Sequences of selections on the same relation are merged into one selection operation.

Example 18.3.4. We illustrate the operation of the DISCO optimizer on the following example.

$$\begin{aligned}&\pi_{x,y}(\sigma_{x \cap y \neq \emptyset}(\sigma_{y \cap \{4\} = \emptyset}(\sigma_{x \cap \{6,8\} = \emptyset}(\\ &\quad \sigma_{x \cap \{2,4\} = \emptyset}(A(x,z) \bowtie B(v,y) \bowtie C(x) \bowtie D(v,y)))))\end{aligned}$$

A parse tree is shown in Figure 18.3.1.

The first step is to perform selections as early as possible. To apply this step, DISCO finds which relations and selections have common variables. Suppose that in a (sub)formula a set of selections (S_1, S_2, ..., S_k) comes after the join of a set of relations (R_1, R_2, ..., R_l). Let V_i be the set of relations which have common variables with S_i, that is:

$$V_i = \{R_n | \text{ (variables in } R_n \text{)} \cap \text{ (variables in } S_i \text{)} \neq \emptyset\} \,.$$

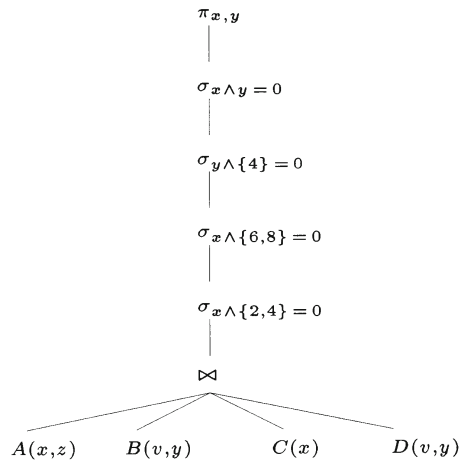

Fig. 18.3.1. The formula before optimization

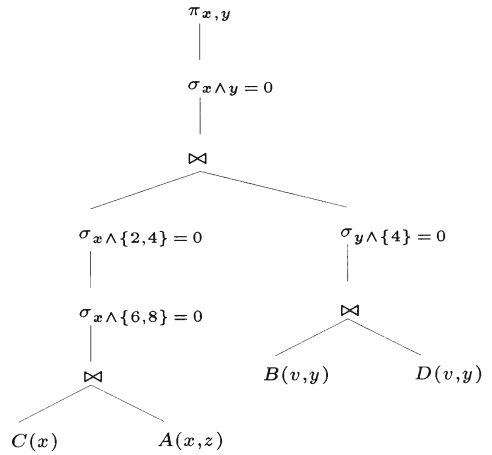

Fig. 18.3.2. The formula after the first optimization step

A selection (S_i) can be applied provided that the join of all the relations mentioned in V_i has already been computed. This gives an obvious limit for how far the selection S_i can be pushed down.

Whenever there are several selections to choose to move down, DISCO checks whether there is any i such that V_i is a subset of all the other V_js. In this case S_i will be pushed down ahead of all the other selections. If there is no such i, DISCO chooses some index i such that V_i has minimum cardinality. After a selection is pushed down, the V_is are updated and the process above is repeated. The result is shown in Figure 18.3.2. After this, projections are pushed down as far as possible, but not ahead of selections. The result is shown in Figure 18.3.3. Finally, cascades of selections are merged.

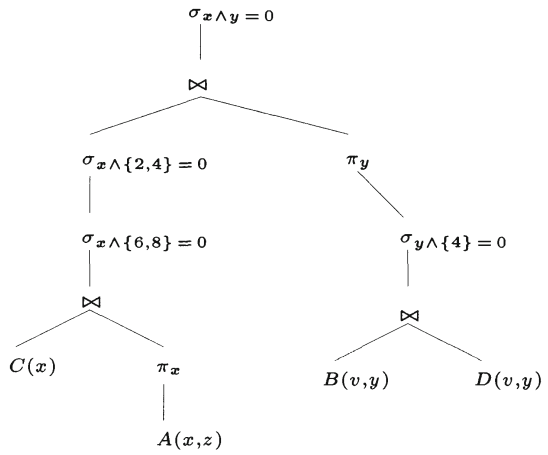

Fig. 18.3.3. The formula after the second optimization step

18.4 Extensibility of DISCO

Although DISCO implements a specific Boolean algebra, namely, the Boolean algebra of sets of integers, DISCO can be extended to Boolean algebras. This requires replacing a few of the basic DISCO routines: redefining the operators $\wedge$, $\vee$, and $'$, changing some of the data storage structures and modifying the Naive and the Semi-Naive evaluation methods with a new subsumption test.

Example 18.4.1. Consider the Boolean algebra in $\mathbb{R}^2$, where the operator $\wedge$ means intersection, $\vee$ means union, and $'$ means complement with respect to $\mathbb{R}^2$. A possible representation would be by sets of linear constraints.

Consider the relation $\texttt{RANGE}(X, A)$ that says that animal X occurs in the region A within a national park. Animals can be identified by single points in $\mathbb{R}^2$, along with some thematic data.

Suppose that animal $(12, 1997)$ is infected with a virus which spreads by contact with other animals. The following DATALOG program defines the set of animals which are in danger of being infected.

$$\begin{aligned}
&\texttt{INFECTION_AREA}(\texttt{A}) \texttt{ :- } \texttt{RANGE}((12, 1997), A).\\
&\texttt{INFECTION_AREA}(A) \texttt{ :- } \texttt{INFECTION_AREA}(A1), \texttt{RANGE}(X, A2),\\
&\qquad\qquad A1 \wedge A2 = 0, A1 \vee A2 = A.\\
&\texttt{IN_DANGER}(X) \qquad \texttt{:- } \texttt{INFECTION_AREA}(A1),\\
&\qquad\qquad \texttt{RANGE}(X, A2), A1 \wedge A_2 \neq 0.
\end{aligned}$$

18.5 Bibliographic Notes

The DISCO system was developed at the University of Nebraska. The first version of the system implemented *set-order* and positive *gap-order* constraints (see Chapter 7) [BR95]. The quantifier elimination technique

used in this system was described in [SRR94] and the Naive evaluation of DATALOG queries was considered in [Rev95a, Rev98d]. More details are given in [Rev97a]. The first version was also used to solve some genome assembly problems [Rev97b].

The second version of DISCO, described here, adds monotone Boolean inequality constraints and Boolean equality constraints over sets of integers [Sal98], but general Boolean equality and inequality constraints cannot be used together.

The quantifier elimination method for Boolean equality constraints is by Boole, while the one for set-order and monotone Boolean inequality constraints is described in [Rev98b]. More details on methods for quantifier elimination in the case of Boolean equality and inequality constraints can be found in [HMO95, MO96].

References

[BR95] J.-H. Byon and P. Revesz. DISCO: A constraint database system with sets. In *Proceedings of the 1st Workshop on Constraint Databases and Applications (CDB'95)*, volume 1034 of *Lecture Notes in Computer Science*, pages 68–83. Springer-Verlag, 1995.

[SRR94] D. Srivastava, R. Ramakrishnan, and P. Z. Revesz. Constraint objects. In *Proceedings of the 2nd International Workshop on Principles and Practice of Constraint Programming (PPCP'94)*, volume 874 of *Lecture Notes in Computer Science*, pages 218–228. Springer-Verlag, 1994.

[Rev95a] P. Z. Revesz. Datalog queries of set constraint databases. In *5th International Conference on Database Theory (ICDT'95)*, volume 893 of *Lecture Notes in Computer Science*, pages 425–438. Springer-Verlag, 1995.

[Rev98d] P. Z. Revesz. Safe query languages for constraint databases. *ACM Transactions on Databases Systems (TODS)*, 23 (1): 58–99, 1998.

[Rev97a] P. Z. Revesz. Problem solving in the DISCO constraint database system. In *Proceedings of the 2nd Workshop on Constraint Databases and Applications (CDB'97)*, volume 1191 of *Lecture Notes in Computer Science*, pages 302–315. Springer-Verlag, 1997.

[Rev97b] P. Z. Revesz. Refining restriction enzyme genome maps. *Constraints*, 2 (3/4): 361–375, 1997.

[Rev98b] P. Z. Revesz. The evaluation and the computational complexity of Datalog queries of Boolean constraint databases. *International Journal of Algebra and Computation*, 8 (5): 472–498, 1998.

[KKR95] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51 (1): 26–52, 1995.

[HMO95] R. Helm, K. Marriott, and M. Odersky. Spatial query optimization: From Boolean constraints to range queries. *Journal of Computer and System Sciences (JCSS)*, 51 (2): 197–210, 1995.

[MO96] K. Marriott and M. Odersky. Negative Boolean constraints. *Theoretical Computer Science (TCS)*, 160 (1/2): 365–380, 1996.