DISCO: A Constraint Database System with Sets

Jo-Hag Byon and Peter Z. Revesz

Department of Computer Science and Engineering
University of Nebraska–Lincoln, Lincoln, NE 68588, USA
email: {byon, revesz}@cse.unl.edu

Abstract. This paper describes the implementation of a constraint database system with integer and set of integers data types. The system called DISCO allows Datalog queries and input databases with both integer gap-order [30] and set order constraints [31]. The DISCO query language can easily express many complex problems involving sets. The paper also presents efficient running times for several sample queries.

1 Introduction

Recently there has been much interest in constraint databases that generalize relational databases by allowing infinite relations that are finitely represented using constraint tuples (ex., [23, 3, 4, 8, 17, 21, 25, 28]).

DISCO (short for Datalog with Integer and Set order COnstraints) is a constraint database system being developed at the University of Nebraska. DISCO implements a particular case of constraint query languages for which a general framework was proposed in [23] analogously to the constraint logic programming framework of Jaffar and Lassez [18].

The particular type of constraints implemented in DISCO are integer gap-order and set order constraints. The incorporation of these constraints into databases was described theoretically in [30, 31, 34] but not implemented before.

DISCO combines the advantages of constraint logic programming and database systems. DISCO provides a non-procedural, logic-based query language and allows users to express the database inputs in a compact and often the only natural way, i.e., using constraints. Like many other database systems, DISCO also works by a translation to a procedural, algebraic language and a bottom-up evaluation that is guaranteed to terminate with some constraint database output. These features make DISCO applicable in various problems in computer-aided design, scientific databases and other areas where set-type data are used.

The current version of DISCO also incorporates several optimization methods. The running time of DISCO as measured on some traditional problems like boolean satisfiability is quite reasonable compared with other solutions. Since DISCO has a DEXPTIME-complete data complexity, any implementation will have some bad worst cases. (Data complexity measures the computational complexity of evaluating fixed size query programs as the input database size varies [7, 38].) However, the running times of most user queries may be faster in
the average case. Developing good benchmark problems for constraint database systems is a topic of research in the future.

The paper is organized as follows. Section 2 describes the DISCO query language. Section 3 presents the outline of the implementation of DISCO, including the data structures, the optimization methods used and the algorithm for query evaluation. In Section 4 we present the testing results on some example queries. In Section 5 we mention related work. Finally, in Section 6 we list some open problems to improve the system that we are working on.

2 The DISCO Query Language

The syntax of the query language of DISCO, denoted $Datalog^{<z, \subseteq, \in_r(z)}$, is that of traditional Datalog (Horn clauses without function symbols) where the bodies of rules can also contain a conjunction of integer or set order constraints. That is, each program is a finite set of rules of the form: $R_0 :-- R_1, R_2, ..., R_l$. The expression $R_0$ (the rule head) must be an atomic formula of the form $p(v_1, ..., v_n)$, and the expressions $R_1, ..., R_l$ (the rule body) must be atomic formulas of one of the following forms:

1. $p(v_1, ..., v_n)$ where $p$ is some predicate symbol.
2. $v \theta u$ where $v$ and $u$ are integer variables or constants and $\theta$ is a relational operator $=, \neq, <, \leq, >, \geq, <_g$ where $g$ is any natural number. For each $g$ the atomic constraint $v <_g u$ is used as shorthand for the expression $v + g < u$.
3. $V \subseteq U$ or $V = U$ where $V$ and $U$ are set variables or constants.
4. $c \in U$ or $c \notin U$ where $c$ is an integer constant and $U$ is a set variable or constant.

Atomic formulas of the form (2) above are called gap-order constraints and of the form (3-4) are called set order constraints. In this paper we will always use small case letters for integer variables and capital letters for set variables. Set variables always stand for a finite or infinite set of integers.

**Remark:** In $Datalog^{<z, \subseteq, \in_r(z)}$ the left hand side of any $\in, \notin$ constraint must be a constant. Without this restriction the query language is not evaluable [33].

Next we give as an example of $Datalog^{<z, \subseteq, \in_r(z)}$ the query for testing the satisfiability of a propositional formula in conjunctive normal form.

**Example 2.1** We assume that the input propositional formulas are in conjunctive normal form and contain only the propositional variables $x_1, x_2, ...$

We describe each input propositional formula $\phi$ using three EDB (extensional database [37]) relations: $No\_vars, No\_clauses, Clause$. The unary EDB relations $No\_vars$ and $No\_clauses$ describe respectively the number of distinct variables and clauses in $\phi$.

Each clause of $\phi$ is represented by a constraint tuple of the $Clause$ EDB relation. In each constraint tuple the first argument gives the clause number and the second argument gives the elements of the clause. The second argument is a