**Distribution Agreement**

In presenting this thesis or dissertation as a partial fulfillment of the requirements for an advanced degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis or dissertation in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis or dissertation. I retain all ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Signature:

_____ _____

Date

An Investigation into Managing SQL-Cardinality Constraints

By

Lesi Wang

Master of Science

Computer Science

_____ [Advisor's signature]

James J. Lu

Advisor

_____ [Member's signature]

Ojas Parekh

Committee Member

_____ [Member's signature]

Li Xiong

Committee Member

Accepted:

_____

Lisa A. Tedesco, Ph.D. Dean of the Graduate School

_____ Date

# An Investigation into Managing SQL-Cardinality Constraints

By

Lesi Wang
B.Sc., Beijing Normal University, 2002

Advisor: James J. Lu, Ph.D.

An abstract of
A thesis submitted to the Faculty of the Graduate School of Emory University
in partial fulfillment of the requirements for the degree of
Master of Science
in Computer Science
2008

# Abstract

## An Investigation into Managing SQL-Cardinality Constraints

By Lesi Wang

Constraint Satisfaction Problems (CSP) are commonly found in practice and finding effective representation language and efficient constraint solving techniques are important research areas. A recent development is the integration of CSP (specifically SAT) solvers with relational database systems to enable CSPs to be modeled using SQL, and solved within the database system. A key challenge in this integration is to keep the SAT encoding of SQL constraints small. In this work, we describe a divide-and-conquer technique for reducing the encoding of cardinality constraints. We present a number of experiments to show the improvements on performance.

# An Investigation into Managing SQL-Cardinality Constraints

By

Lesi Wang
B.Sc., Beijing Normal University, 2002

Advisor: James J. Lu, Ph.D.

A thesis submitted to the Faculty of the Graduate School of Emory University
in partial fulfillment of the requirements for the degree of
Master of Science
in Computer Science
2008

ACKNOWLEDGMENT

I would like to acknowledge all the people who help me with my thesis and my study at Emory.

I want to first express my special gratitude to my advisor, Dr. James J. Lu, who directed, encouraged and help me all along with my studies here.

I will also thank Dr. Li Xiong and Dr. Ojas Parekh who squeezed time to effort as my committee members.

Special thanks go for my colleagues Sebastien Siva, Dongfang Zhao and Robin Lohfert, who helped me a lot for the project.

Finally, I would like to thank my family and my friends: they supported me with every step in my life.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Modeling and solving complex constraints over sets of relational data is important in practice. The naive approach to developing such an application is to implement ad-hoc procedural code that accesses and exports the data through embedded SQL programming. A more elegant and reusable approach is to use declarative programming languages that integrate constraint modeling with database access in transparent ways. The most popular of these languages are rooted in deductive databases and constraint logic programming. A natural extension that generalizes relational databases with quantifier free constraints is constraint databases [5]. The main disadvantage of these solutions is that they require users to have knowledge beyond traditional RDB and SQL expertise.

An intriguing alternative, recently proposed by Cadoli and Mancini [1], is to extend SQL with the syntax necessary to express constraint satisfaction problems (CSP) specific constraints over relations. The key concept is the addition of a non-deterministic `GUESS` operator that declares a relation whose extension may be computed based on a set of constraints written in SQL.

The SQL Constraint Data Engine (SCDE) is an ongoing implementation

project to explore the feasibility of the GUESS concept by encoding input CSPs, written in SQL, into satisfiability problems (SAT). A SAT solver is employed to solve the encoded instances, with solutions from the SAT solver decoded and presented to the user by the SCDE. Several modifications to the syntax introduced in [6] have been incorporated to improve usability and SQL consistency. A CSP specification example in SCDE, shown in Figure 1.1, is to partition incoming class of Emory Oxford freshmen into seminar groups of equal size and with balanced characteristics. We will refer to the problem as SGE and use it as a running example.

The `COMPUTABLE` keyword specifies the solution table for the CSP. In this case, a record in the computed result specifies an assignment of a student, pid, to a group, gid. In the example, we assume 400 students of different ethnicities and home regions. Our objective is to divide them into 25 freshmen groups of 16 students with group characteristics that are as balanced as possible. That is, the number of students of each gender, ethnicity and home region should be similar across the 25 groups.

To represent the problem in a database, 5 tables are used.

```
people(id, gender, geocode,diversity)
groups(id):
gender(trait, min, max)
geocode(trait, min, max)
diversity(trait, min, max)
```

Table `people` describes characteristics of individual students; table `groups` contains the target groups that students will be assigned to; table `gender`, `geocode`, and `diversity` specify the numeric constraints associated with the different group characteristics. The constraint numbers are automatically calculated from `people` by calculating the number of students of a particular trait (e.g. female), and dividing by the number of groups. If the number divides

| Trait | min | max |
|---|---|---|
| Male | 6 | 7 |
| Female | 9 | 10 |

Table 1.1: Constraint table gender

evenly, then *min* equals *max* in the record, otherwise *min* is the floor of the quotient and *max* is the ceiling of the quotient; they differ by one at most. For example, if we have 174 male students and 226 females students, then the gender constraint table appears as in Table 1.1. Note that the *min* and *max* thus computed are satisfiable constraints when the characteristics are considered independently. When multiple characteristics are considered together, a solution where all groups characteristics differ by at most one may not be possible.

Figure 1.1 shows the corresponding constraints to SGE. Constraints C1 and C2 ensure that each student can only be assigned in one target group, and each target group has exactly 16 students, respectively. Constraints C11 through C14 check the group characteristics of each trait.

A solution to the problem is a set of records in `STUDGROUPS` such that each constraint is satisfied. A near solution is scored by the percentage of the number of records that violate the constraints. More formally, let $G = \{\text{gender}, \text{geocode}, \text{diversity}\}$, then for a computed instance $M$ of STUDGROUPS, $score(M)$ is given by the following equation. Remarks on notation: $S = M \underset{pid=id}{\bowtie} \text{people}$, $D(t, x, c) = |\mathcal{F}_{count(pid)}\ S \underset{\substack{s.t=t.trait \\ s.x=groups.id}}{\bowtie} t - t.c|$

$$Score(M) = \frac{\sum\limits_{g \in G} \sum\limits_{gid \in groups} min(D(g, gid, min), D(g, gid, max))}{\mathcal{F}_{count(distinct\ gid)} S}$$

Suppose for instance, two groups in $M$ would be 11 and 7 females, respectively, and the remaining group characteristics all meet the requirements of the constraints, for all groups. Then $score(M)$ would be

$min(|11-9|, |11-10|)/400 + min(|7-9|, |7-10|)/400 = 3/400 = 0.75\%$

The Student Grouping Example (SGE) is an extreme example of a CSP in which every constraint is a cardinality constraint. When translated into a SAT instance, cardinality constraints frequently produce unacceptably large number of clauses as well as large clauses using a standard encoding. In general, given a set of SAT variables ( $V_1$ .... $V_n$ ) the constraint that at most $m \leq n$ variables are true is a set of $\binom{n}{m+1}$ negative clauses of length m+1. Conversely, the constraint that at least m variables are true is a set of $\binom{n}{n-m+1}$ clauses of length n-m+1.

Cardinality constraints are common and important in CSP, and as the above example shows, can be easily represented in SQL by the aggregated `COUNT` function. Handling cardinality constraints efficiently in SAT solvers is an active area of research [8] [7], and is also the focus of this thesis. Specifically, we address the following question:

**Problem Statement:** What general heuristics are effective for reducing the size of the SAT encoding from SQL constraint?

We present a simple divide-and-conquer technique and study its performance on several problem instances related to the above example. Our approach breaks up the problem by exploring the numeric constraints in the data set. Assuming that cardinality constraints are naturally represented as data in tables like Table 1.1, our technique should be generally useful to systems such as SCDE with minimal modifications.

```
CREATE COMPUTABLE TABLE STUDGROUPS (
pid INTEGER FOREIGN KEY REFERENCES PEOPLE(id),
gid INTEGER FOREIGN KEY REFERENCES GROUPS(id),

--Each student must be assigned to exactly one target group
CONSTRAINT C1 CHECK (NOT EXISTS
 (SELECT p.id
 FROM people p
 WHERE 1 <> (SELECT COUNT(*)
             FROM studgroups s
             WHERE s.pid = p.id))),

-- Each target group will have exactly 16 students
CONSTRAINT C2 CHECK (NOT EXISTS
 (SELECT g.id
 FROM groups g
 WHERE 16 <> (SELECT COUNT(*)
             FROM studgroups s
             WHERE s.gid = g.id))),

-- The number of females and males in each group must be at least gender.min
CONSTRAINT C11 CHECK (NOT EXISTS
 (SELECT grp.id, gen.trait, gen.min
 FROM groups grp, gender gen
 WHERE gen.min > (SELECT COUNT(*)
                  FROM studgroups s, people p
                  WHERE s.pid = p.id
                  AND s.gid = grp.id
                  AND p.gender = gen.trait))),

-- The number of females and males in each group must be at most gender.max
CONSTRAINT C12 CHECK (NOT EXISTS
 (SELECT grp.id, gen.trait, gen.max
 FROM groups grp, gender gen
 WHERE gen.max < (SELECT COUNT(*)
                  FROM studgroups s, people p
                  WHERE s.pid = p.id
                  AND s.gid = grp.id
                  AND p.gender=gen.trait))),

-- The number of students from each geographic areas should be at least geocode.min
CONSTRAINT C13 CHECK (NOT EXISTS
 (SELECT grp.id, geo.trait, geo.min
 FROM groups grp, geocode geo
 WHERE geo.min > (SELECT COUNT(*)
                  FROM studgroups s, people p
                  WHERE s.pid = p.id
                  AND s.gid = grp.id
                  AND p.geocode = geo.trait))),

-- The number of females and males in each group must be at most geocode.max
CONSTRAINT C14 CHECK (NOT EXISTS
 (SELECT grp.id, geo.trait, geo.max
 FROM groups grp, geocode geo
 WHERE gen.max < (SELECT COUNT(*)
                  FROM studgroups s, people p
                  WHERE s.pid = p.id
                  AND s.gid = grp.id
                  AND p.geocode=geo.trait))),
--.............
-- Same constraints for hometown regions and other traits if required
--.............
 )
COMPUTE TABLE STUDGROUPS;
```

Figure 1.1: SCDE specification of Student Grouping

# Chapter 2

# Background

In this chapter, we provide an overview of the SCDE system and a related work on divide-and-conquer to improve performance.

## 2.1 System Overview

The SCDE architecture is shown in Figure 2.1. It has two parts: a database system (currently SQLite) and a SAT solver (currently SATO [3]). The SCDE takes a CSP specification as the input and exports the solution table. The input CSP is represented as a combination of internal tables and parse trees of the SQL constraints. Users interact with the system in several ways: a commandline tool for interactive problem development, a file input, a graphical user interface (under development), and application program interfaces. To encode the input problem as SAT, it first translates the computable table Q into an internal table, vbmap, where each possible record of Q is associated with an unique integer *id*. These unique *id*'s are used to construct clauses written in standard formats accepted by all SAT solvers. The most common way of specifying cardinality constraints in SCDE is by the SQL statement pattern shown in Figure 2.2.

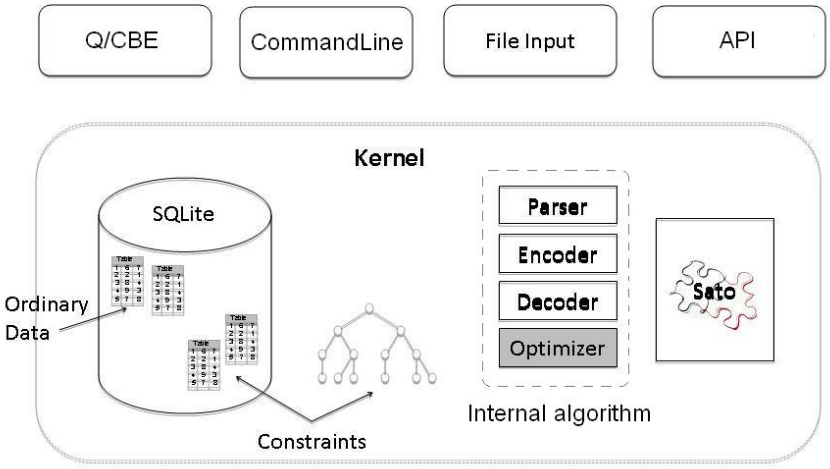


Figure 2.1: Structure of SCDE

```
CHECK (NOT EXISTS
    (SELECT *
    FROM Q, B1, ...., Bk
    WHERE <c>
              AND m op (SELECT COUNT(*)
              FROM Q, B'1,...., B'j
              WHERE <c'>)
    ))
```

Figure 2.2: SCDE Cardinality syntax

In the pattern, Q is the computable table, B's are the base tables, $m$ is an expression that evaluates to a number and and $op$ is a comparison such as $<$ and $=$. The SCDE translates such a constraint to a clause of the form:

$$(V_1 \ldots V_n)\ op\ m$$

Each $V_i$ is the SAT variable associated with a tuple of the computable table in the result of the nested subquery. Recall the data of Table 1.1 and constraint C2 of SGE from Figure 1.1. If $V_1 ... V_n$ are all variables associated with assigning females to group1, then the translated constraint would appear as

$$(V_1 \ldots V_n) \leq 10 \text{ and } (V_1 \ldots V_n) \geq 9$$

9 is the minimum number that should be in any group of a solution that considers only the gender constraint. As mentioned, a SAT representation of these constraints typically require unacceptably large boolean formulas. To put this in context, if given the problem of distributing 65 students among 10 groups evenly. Each group should be assigned at least 6 students and at most 7 students. These produce to $\binom{65}{65-7+1}$= 82,598,880 positive clauses of length 59 and $\binom{65}{6+1}$= 696,190,560 negative clauses of length 7 by the standard encoding. Finding heuristics to reduce this complexity is one of the purposes of the optimizer (shown in Figure 2.1 by the gray box), and is the focus of this thesis.

## 2.2 Case study

The SCDE has been applied to several case studies including the ACC Basketball Scheduling Problem [6]. The basic problem is to schedule a double round-robin basketball tournament for 9 teams. Table 2.1 summarizes the performance of several variants of the original problem using the SCDE. Since the schedule is a double round-robin tournament, each team must play every other team twice. The number of games that must occur between a match-up and a rematch is indicated in parentheses. Problem `generic` contains the basic requirements of a double round-robin tournaments with a rematch interval of 9 games. Problem `P` is the simplified version of the original problem. Problem `All` contains all constraints of the original problem but with varying requirements for the rematch intervals. `Orig` represents the original problem without modifications. It is intuitively clear that the smaller the spacing requirement, the easier it is to solve the problem. This intuition is reflected in the solving time. The experiments show that compiling adds a modest but consistent overhead to the overall time to find a solution. It is not surprising that a difficult to solve problem instance,

| | generic | P(5) | P(6) | All(4) | All(5) | orig |
|---|---|---|---|---|---|---|
| clauses | 180K | 174K | 175K | 175K | 176K | 182K |
| compile time | 5s | 5s | 5s | 8s | 7s | 7s |
| solve time | 0s | 177s | 820s | 403s | 577s | NA |
| solution found | yes | yes | no | yes | no | NA |

Table 2.1: ACC performance

such as `Orig`, is also difficult for SCDE.

## 2.3 Improvement of SAT solver

Cardinality constraints are not the only source of large encodings in SCDE specifications. A problem that contains many attributes and data values can also produce a large number of clauses. An example is a course scheduling problem which contains teaching preference of professors for courses and time. Suppose 4 base tables with information about professors, courses, time and rooms are given.

```
professor(pid ,name, rank, department, research_areas, #courses)
course(cid, department, topics, term)
time(tid, duration)
rooms(rid, capacity)
```

Assume the computable table is composed from the keys of each base table:

`schedule`(pid, cid, tid, rid).

Possible constraints on the solution include:

1. Each course cannot be assigned to different professors to teach
2. Course topics should match professors research interest and expertise.
3. Room and time has no conflicts.

Previous studied in [4] and [6] introduced an approach to improve the solution time by incrementally encoding the problem as a series of smaller problems. The basic idea is to divide the schema of the computable table into several sub-schema, solve the most important sub-schema and its associated constraints first, then use its solutions to guide the search for solutions to the next sub-schema. We call the approach Attribute Decomposition. For the example above, the computable table is divided into 3 sub-tables:

```
P1(pid, cid),
P2(pid, cid, tid),
P3(pid, cid, tid, rid).
```

The choice of attributes for each sub-problem is based on a simple analysis of the ways in which the attributes are connected to each other in the SQL constraints. In general, an attribute that is syntactically related to many other attributes (i.e. appear together in a logical expression) is given a higher weight. The weights of all the attributes are used to determine the choices of the schema for the initial sub-problem. Experiments show that Attribute Decomposition yields significant performance improvements. With a setting of 20 professors, 35 courses, 12 time slots, and 8 rooms, the brute-force method solve a solution within 5.553 for 692,255 clauses, while Attribute Decomposition uses 2.829 and only 85 + 92 + 579 clauses are involved. In another experiment with the same settings, the brute-force one uses 44.67 seconds to compute 5,571,459 clauses, but Attribute Decomposition only takes 1.218 for a total number of 173 + 92 + 579 clauses [4].

To verify the robustness of the attribute selection heuristics of [4], we adapted an algorithms that had been applied to informative retrieval[2] to calculate the importance of the attributes. In this algorithm, an undirected graph is built to represent the linguistic associations among terms that appear in constraints.

The terms include table and attribute names, and are represented as nodes. The associations are represented as edges. Each node is initialized with two numbers: informative score (i-score) which represents the volume of information associated with the term, and representative score (r-score) that indicates the ability that a term can reduce a problem. The two scores are iteratively updated until they stabilize, and the final aggregate score is taken to be the sum of the r-score and the difference of the i-score and the initial i-score. Several comparative studies including problems of airline scheduling, basketball and course scheduling showed that the two ranking approaches agreed in their results [6].

# Chapter 3

# Divide and conquer by traits

The Attribute Decomposition approach to divide a SCDE problem works well when the computable table has a non-trivial number of attributes. For computable tables that have only two or three attributes, as is the case with our student grouping example (SGE), this approach is inapplicable. In addition, the assumption that analyzing constraints can reveal useful information about sub-problems does not necessarily hold. As our example demonstrates, specific numbers associated with cardinality constraints are frequently stored as data (e.g. the gender constraint table). Finally nearly all the constraints for the example are written identically. This makes differentiation based on syntactic analysis not particularly useful. As mentioned, the size of encoding given a cardinality constraint is on the order $\binom{f(n)}{g(m,n)}$, where $n$ is the number of possible tuples for the computable table, and m is the value associated with the constraint that often indicates the number of records (of a certain type) that must, or must not, appear in the solution. Hence the key to reducing the size

| n | groups | min | negative clauses | max | positive clauses | time | total |
|---|---|---|---|---|---|---|---|
| 39 | 10 | 3 | 7410 | 4 | 5757570 | 0 | 5764980 |
| 39 | 5 | 7 | 16313115 | 8 | 1059575660 | 0 | 1075888775 |
| 40 | 11 | 3 | 8580 | 4 | 7238088 | 0 | 7246668 |
| 40 | 9 | 4 | 88920 | 5 | 34545420 | NA | 34634340 |
| 40 | 7 | 5 | 639730 | 6 | 130504920 | 0 | 131144650 |
| 45 | 12 | 3 | 11880 | 4 | 14661108 | 0 | 14672988 |
| 45 | 6 | 7 | 48870360 | 8 | 5316978810 | 0 | 5365849170 |
| 45 | 7 | 6 | 8552313 | 7 | 1508872365 | NA | 1517424678 |
| 45 | 8 | 5 | 1191960 | 6 | 363036960 | 0 | 364228920 |
| 45 | 12 | 3 | 11880 | 4 | 14661108 | 0 | 14672988 |
| 50 | 6 | 8 | 599306400 | 9 | 61633669020 | NA | 62232975420 |
| 50 | 9 | 5 | 2072700 | 6 | 898959600 | 195 | 901032300 |
| 50 | 15 | 3 | 18375 | 4 | 31781400 | NA | 31799775 |
| 50 | 16 | 3 | 19600 | 4 | 33900160 | NA | 33919760 |
| 50 | 17 | 2 | 850 | 3 | 3915100 | 0 | 3915950 |
| 50 | 19 | 2 | 950 | 3 | 4375700 | 261 | 4376650 |
| 50 | 21 | 2 | 1050 | 3 | 4836300 | NA | 4837350 |
| 50 | 26 | 1 | 26 | 2 | 509600 | 0 | 509626 |
| 51 | 25 | 2 | 1275 | 3 | 6247500 | NA | 6248775 |
| 51 | 26 | 1 | 26 | 2 | 541450 | NA | 541476 |

Table 3.1: Tests for the size of the SAT encoding and the performance of SATO

of an encoding is by lowering the values of n and m. This is the basis of our approach described in this chapter.

As an initial step experiment to understand how large of a formula SATO can handle efficiently, we experimented with various values for n and m. Table 3.1 shows some of the results. The values for *min* and *max* are the ceiling and floor for the size of each group, respectively. *NA* indicates that no solution was found quickly (within 10 minutes), and 0 indicates that a solution was found immediately.

All complete SAT Solvers adopt the basic Davis-Putnam Longman-Loveland (DPLL) procedure. While some solvers incorporates more advanced search heuristics, it is likely that all have limits that are comparable or within some

constant factor of SATO's performance.

The data reveals some patterns, but are unfortunately inconclusive. A comparison of numbers of variables versus time consuming to provide a solution is shown in Figure 3.1. We find very few instances are solvable when number of variables exceeds 800 (e.g. 40 students and 20 groups), except for very small problems. For number of variables smaller than 800, the difficulty of the problem becomes harder to predict. For our experiments (discussed in Chapter 4), we generally aim to divide each problem until number of variables is substantially smaller than 800, and when associated with the number of groups, n is generally assigned to be 40.

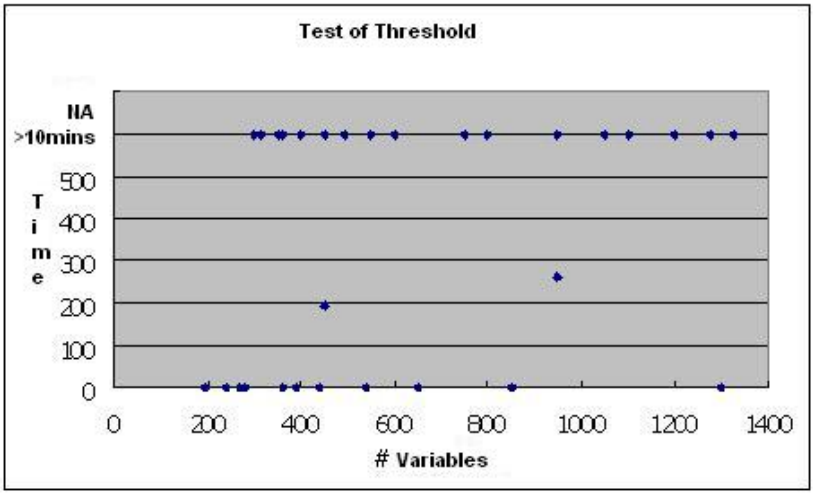


Figure 3.1: Comparison of #variables and solving time

To obtain smaller values of n and m, the straightforward way is to divide the students into sufficiently small subsets, obtain solutions for each subset, and

then combine these to form the overall solution. One possibility is to divide the initial data set randomly into evenly sized subsets and calculate all constraints at once, but a small set of experiments show that this is infeasible. When we use an example of 400 students with 2 traits for gender, 8 traits for geocode (i.e., home region), and 7 traits for diversity, 75% of the subsets produce no solutions and the total time to calculate a unsolvable subset costs more than 30 minutes.

A better approach, it turns out, is to divide students along traits. Each trait induces a sub-problem that can be solved independently from the other traits. This process can be repeated for each sub-problem. Merging of solutions occur in the reverse of the order in which the traits are considered.

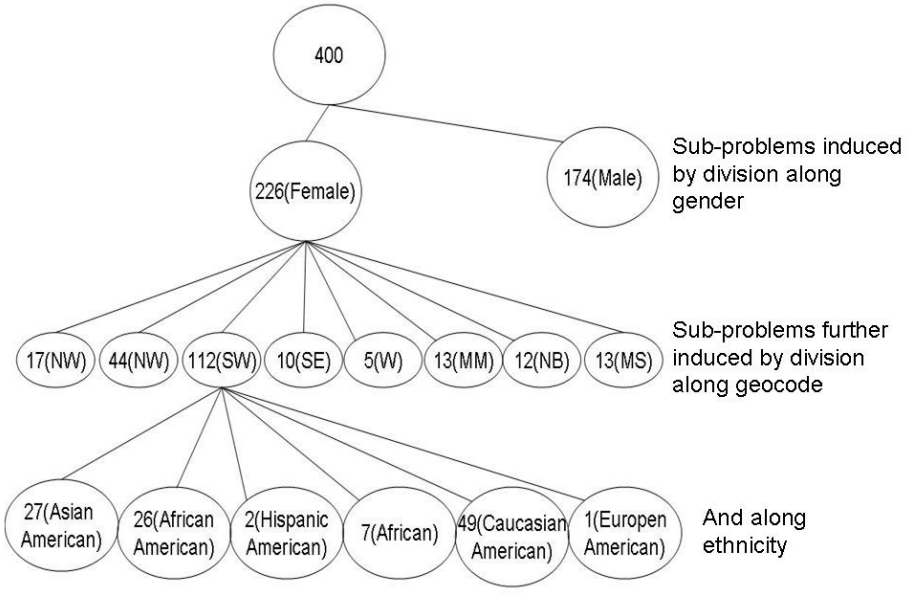


Figure 3.2: Divide by trait values

For SGE shown in Figure 1.1, if we first divide along gender as shown in

Figure 3.2, this produces 2 sub-problems of 226 females and 174 males. Note that by dividing on gender, the gender constraints no longer apply on the sub-problems; they are absorbed by the constraint C2 that specifies the size of each group, but modified to reflect the size of the new problems. Since the data set of the sub-problem may not divide evenly into the number of groups, usually the new group size constraints also need to be represnted as a range. The SCDE specification for the sub-problem is shown in Figure 3.3. Similarly, the cardinality constraints specified in the base tables for diversity and geographic origins would need to be adjusted appropriately. If we continue to divide the sub-problems along other traits, then the final sub-problem would only be constrained by two basic constraints: that each student is assigned to exactly one group, and the size of each group.

As a comparison of the reduction in the number of clauses, a subset that has only 27 Asian, SW and female students over 25 groups (and with only the two basic constraints) can be encoded in 8,800 clauses, a substantial reduction to the 3.84408E+19 clauses in the original problem.

To merge the solutions of the sub-problems, we adopt a simple greedy algorithm that sorts the groups with respect to the group sizes. The group of a smaller size is combined with a group of a larger size. A more precise descriptions of the complete divide-and-conquer algorithm is shown in Algorithm 1, which is on the next page. We will refer to this by the name DC.

In the algorithm, we assume a set of base tables. Input $B$ denotes the distinguished base table that represent the elements to be partitioned. In particular, $B$ contains individual traits that form the basis for the group characteristics.

Note that merging is performed iteratively where each iteration merges a solution with its neighbor. The merged solutions are put back into the beginning part of the list $S[]$, and the next round of merging only traverses half of the list.

**Algorithm 1** DC($P$) //DC: divide-and-conquer

**Input:** $P$ is the input problem $(Q, B, R, T)$ with $Q$: the computable table $B$: base table of elements to be partitioned $R$: constraints associated with the problem $P$ $T$: an ordered list of individual characteristics of $B$

```
1: if size(B) < limit(α,β) then
2:    encode and solve P by SCDE
3:    decode and return the solution;
4: else
5:    let S[]: a list to hold temporary solutions (S[][] is the list of groups for
      each solution)
6:    let C: the first (i.e., the most important) characteristic in T
7:    // divide and solve P along the traits of C
8:    T' = T − C;
9:    R' = R with cardinality constraints associated with C removed
10:   i = 0;
11:   for each trait v in C do
12:      B' = σ_{B.C=v}(B);
13:      P' = (Q, B', R', T');
14:      S[i] = DC(P');
15:      i = i + 1;
16:   end for
17:   // merge the solutions
18:   s = number of traits in C;
19:   repeat
20:      z = 0;
21:      for i = 0, j = 1; i < s − 1; i = i + 2, j = j + 2 do
22:         sort the groups in S[i] in increasing order
23:         sort the groups in S[j] in decreasing order
24:         for k = 1 to the number of groups do
25:            S[z][k] = merge S[i][k] with S[j][k]
26:         end for
27:         z = z + 1;
28:      end for
29:      s = ⌈ s / 2 ⌉;
30:   until s = 1;
31:   return S[0];
32: end if
```

```
CREATE COMPUTABLE TABLE STUDGROUPS (
pid INTEGER FOREIGN KEY REFERENCES PEOPLE(id),
gid INTEGER FOREIGN KEY REFERENCES GROUPS(id),

--Each student must be assigned to exactly one target group
CONSTRAINT C1 CHECK (NOT EXISTS
 (SELECT p.id
 FROM people p
 WHERE 1 <> (SELECT COUNT(*)
             FROM studgroups s
             WHERE s.pid = p.id))),

-- Each target group have no more than ⌈total/groups⌉ students
CONSTRAINT C3 CHECK (NOT EXISTS
 (SELECT g.id
 FROM groups g
 WHERE ⌈total/groups⌉ < (SELECT COUNT(*)
             FROM studgroups s
             WHERE s.gid = g.id))),

-- Each target group will have no less than ⌊total/groups⌋ students
CONSTRAINT C4 CHECK (NOT EXISTS
 (SELECT g.id
 FROM groups g
 WHERE ⌊total/groups⌋ > (SELECT COUNT(*)
             FROM studgroups s
             WHERE s.gid = g.id))),

-- The number of students from each geographic areas should be at most geocode.max
CONSTRAINT C13 CHECK (NOT EXISTS
 (SELECT grp.id, geo.trait, geo.max
 FROM groups grp, geocode geo
 WHERE geo.max < (SELECT COUNT(*)
                  FROM studgroups s, people p
                  WHERE s.pid = p.id
                  AND s.gid = grp.id
                  AND p.geocode = geo.trait))),

-- The number of females and males in each group must be at least gender.min
CONSTRAINT C14 CHECK (NOT EXISTS
 (SELECT grp.id, geo.trait, geo.min
 FROM groups grp, geocode geo
 WHERE gen.min > (SELECT COUNT(*)
                  FROM studgroups s, people p
                  WHERE s.pid = p.id
                  AND s.gid = grp.id
                  AND p.geocode=geo.trait))),
--.............
-- Same constraints for home regions and other traits if required
--.............
 )
COMPUTE TABLE STUDGROUPS;
```

Figure 3.3: SCDE specification of sub-grouping SCDE specification for sub-problems after dividing by gender

Hence, if $s$ the number of traits and $g$ the number groups, then $log_2(s)g$ number of merges are required overall. If the number of traits is not even, then we assume that the result of merging the last solution in S[] is just the solution

itself.

# Chapter 4

# Experiments

## 4.1 Case studies

To experiment, we created several data sets shown in Table 4.1. Each data set has 3 characteristics: gender, ethnicity and home regions that have 2, 3 and 4 traits respectively. The trait Ordering column describes the order in which the division is applied to the original problem.

| Data set | Gender F/M | Ethnicity NW/SW/NE/NW | Home region W/B/A | Trait Ordering | # groups |
|---|---|---|---|---|---|
| 1(200) | 95/105 | 54/54/52/40 | 51/106/43 | G.D.S. | 10 |
| 2(200) | 105/95 | 45/65/50/40 | 20/65/115 | G.D.S. | 10 |
| 3(300) | 146/154 | 81/80/79/60 | 143/83/74 | G.D.S. | 20 |
| 4(300) | 141/159 | 92/42/38/128 | 77/158/65 | G.D.S. | 20 |
| 5(400) | 214/186 | 90/130/100/80 | 40/130/230 | D.G.S. | 25 |
| 6(400) | 201/199 | 108/106/106/80 | 192/110/98 | G.D.S. | 25 |

Table 4.1: Data sets

As summary comparison of the time to solve each problem by the divide-and conquer approach is shown in Figure 4.1 and the corresponding result data

details are in Table 4.2. By comparison, when the entire problem is encoded and submitted to SATO - the brute-force approach, only data 1 produced a solution, after 16 hours of computing time. Computing time for all other data sets failed to terminate after 24 hours.

The computing time required for divide-and-conquer is consistent and does not change much in relation to the problem size. The reason that data set 1, which requires much more time than other, larger data sets to compute is because of the existence of a sub-group that contains 35 students. This sub-problem alone took 67 seconds to solve. With respect to the score of each solution, we see that it does not grow significantly as the size of problem increase. The worst case result is data set 5 with a score of 15.25%. A closer inspection of the solution shows that group characteristics are quite even, however; the maximum variance for any trait is 3 among the 25 groups. Column Ordering is the order of input trait to Algorithm DC(P), is the order of input trait, where S is gender, D is diversity and G is the geocode.

| Data set | Time(s) | Score |
|---|---|---|
| 1 | 73 | 0.00% |
| 2 | 3 | 5.50% |
| 3 | 7 | 10.67% |
| 4 | 7 | 9.00% |
| 5 | 8 | 15.25% |
| 6 | 8 | 5.00% |

Table 4.2: Performance table

A more detailed analysis of the effect of trait ordering is shown in Table 4.3 data set 1. We observe the following: a characteristic with a smaller difference among its traits are generally better balanced with respect to the specified constraint in the solution. This suggests a heuristic for ordering traits: dividing the data set along the trait with the largest variance first may provide a more balanced solution overall. For a CSP that involves multiple cardinality constraints

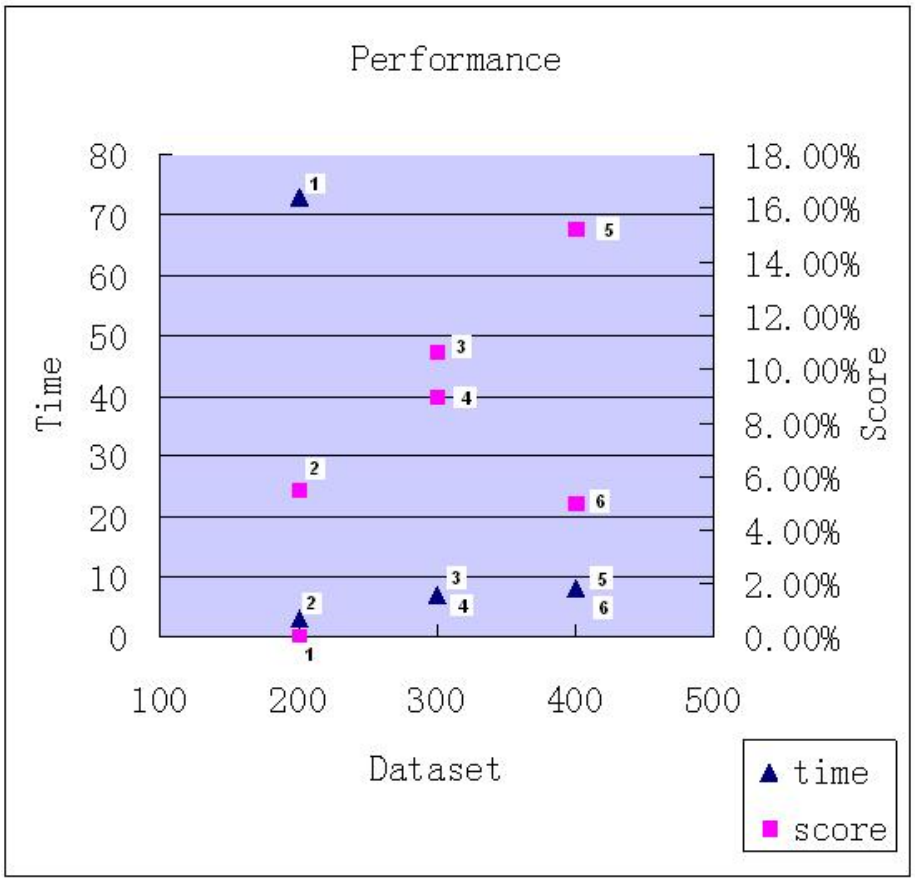


Figure 4.1: Time/Score performances

over multiple traits, this ordering may be automatically calculated or specified by the user.

## 4.2 A Real World Example

Returning to the original Emory Oxford SGE, which motivated our work in this thesis, the 400 students are to be partitioned among 25 classes, and they come from 8 home regions and are made up of 7 ethnicity groups.

| Data set | Order | Score |
|---|---|---|
| 1 | G.D.S | 0 |
| 2 | G.S.D | 5% |
| 3 | D.G.S | 20% |
| 4 | D.S.G | 5% |
| 5 | S.D.G | 10% |
| 6 | S.G.D | 15% |

Table 4.3: Ordering comparision

Table 4.4 shows a time comparison of our divide-and-conquer approach to the brute-force method. For the problem, we only included min-cardinality constraints. If we add the max-cardinality constraints to the problem, the brute-force method almost never return with an answer within a reasonable amount of time. Even with only the minimum constraints, the table shows that the brute-force approach still took far longer than the divide-and-conquer approach which included both and max- and min-cardinality constraints. And while the overall scores are similar, the variance of each characteristic among the groups are generally smaller by the divide-and-conquer approach. This is shown in Table 4.5. For each characteristic in the table, the associated traits are represented as numbers in the second column. For example, 1 represents female for gender. The columns Gmin and Gmax represent the min- and max-cardinality constraints, respectively, for each group characteristic considered in isolation. That is, if the characteristic is the only constraint for the problem, then an ideal solution will have, for each group, at least Gmin and at most Gmax elements with that characteristic. The next two columns, DCmin and DCmax, show the actual range of values in the solution produced by Algorithm DC(P). For instance, the the smallest number of females in a group is 5 and the largest number is 9. The last two columns, BFmin and BFmax represent the same information using the brute-force approach.

From the data, it is intuitively clear that the solution of the divide-and-

| | brute-force | divide-conquer |
|---|---|---|
| time(s) | 261 | 25 |
| score | 49.75% | 42.5% |

Table 4.4: Solutions comparison for Oxford students

| characteristic | trait | Gmin | Gmax | DCmin | DCmax | BFmin | BFmax |
|---|---|---|---|---|---|---|---|
| gender | 1 | 6 | 7 | 5 | 9 | 4 | 9 |
| | 2 | 9 | 10 | 8 | 12 | 7 | 12 |
| geocode | 1 | 7 | 8 | 6 | 9 | 5 | 11 |
| | 2 | 3 | 4 | 1 | 6 | 2 | 6 |
| | 3 | 1 | 2 | 0 | 4 | 1 | 4 |
| | 4 | 0 | 1 | 0 | 4 | 0 | 2 |
| | 5 | 0 | 1 | 0 | 2 | 0 | 2 |
| | 6 | 0 | 1 | 0 | 2 | 0 | 3 |
| | 7 | 0 | 1 | 0 | 3 | 0 | 3 |
| | 9 | 1 | 1 | 0 | 2 | 0 | 3 |
| diversity | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| | 2 | 4 | 5 | 4 | 6 | 2 | 10 |
| | 3 | 2 | 3 | 2 | 3 | 1 | 6 |
| | 4 | 0 | 1 | 0 | 5 | 0 | 3 |
| | 5 | 1 | 2 | 0 | 5 | 1 | 3 |
| | 6 | 6 | 7 | 5 | 7 | 4 | 10 |
| | 7 | 0 | 1 | 0 | 1 | 0 | 1 |

Table 4.5: Comparison of ranges for Oxford students

conquer are better balanced. While the greatest differences in the group characterstics using the brute-force method are 8, 6, and 6 (for `diversity 4, 6`, and geocode 1), the largest differences using the divide-and-conquer are 5, 5, and 5 (for `geocode 2`, `diversity 4` and `5`).

# Chapter 5

# Conclusion and future work

We have introduced a simple divide-and-conquer technique to reduce the size of the SAT encoding of cardinality constraints in SQL. The technique divides a problem along traits of elements to be grouped based on constraints of the problem. This reduces both the size of the data and the number of constraints to be solved in the resulting sub-problem. Experimental comparison with the brute-force approach on a prototypical grouping problem shows that our technique can provide solutions faster and more accurately than the brute-force approach.

Many questions remain. We discuss here a few of the more important ones.

1. Our experiments have focused on problems that contain only cardinality constraints. It is not clear, however, how Algorithm DC(P) can be integrated with other heuristics in the presence of other types of constraints. In particular, it is possible that there exist non-cardinality constraints that are incompatible with the subproblems that Algorithm DC(P) generates. Designing techniques to simultaneously address both cardinality and non-cardinality constraints equitably is an important research issue.

2. The merging step in Algorithm DC(P) is, by itself, an interesting con-

straint satisfaction problem. Currently, the greedy algorithm only takes into account the overall size of groups when merging. Consequently, smaller group characteristics may not be well balanced. An interesting question is how the balance of both sets of constraints can be effectively maintained during merge. Relatedly, analyzing and establishing a bound for the best balance across all groups is an interesting theoretical question.

3. Some recent work on cardinality constraints have introduced more compact SAT encodings [8]. A comparative study of the performance between such alternative encoding and our divide-and-conquer may provide useful guidelines for optimization techniques in SCDE.

# Bibliography


[1] M. Cadoli , T. Mancini: Combining relational algebra, SQL, constraint modelling, and local search. *Theory Pract. Log. Program., 7(1-2), 37-65*, 2007.

[2] J. Liu and X. Dong and A. Halevy.: Answering structured queries on unstructured data. *WebDB 06 Chicago, Illinois US.*, 2006.

[3] H. Zhang.: SATO: An efficient propositional prover. *CADE., (272-275)*, 1997.

[4] R. Lohfert, J. Lu and D. Zhao: SATO: Solving SQL Constraints by Incremental Translation to SAT. *The 21st International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE'08), Proceedings LNAI 5027, Wroclaw, Poland*, 2008.

[5] P. C. Kanellakis and D. Q. Goldin.: Constraint Programming and Database Query Languages. *In TACS., 96-120*, 1994.

[6] S. Siva, J. Lu and L. Wang.: A SQL Database System for Solving Constraints. *CIKM08 Ph.D. Workshop*, 2008.

[7] J.P Marques Silva, I. Lynce. Towards Robust CNF Encodings of Cardinality Constraints, *Principles and Practice of Constraint Programming (CP)*, 2007.

[8] C. Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints, *Principles and Practice of Constraint Programming (CP)*, 2005.