面向非等值连接负载的数据生成器研究与实现

作 者 姓 名：李捷荧
指 导 教 师：赵海 教授
单 位 名 称：计算机科学与工程学院
专 业 名 称：计算机科学与技术

东 北 大 学
2016 年 6 月

The Research and Implementation of Data Generator Oriented to Non-Equality Join Workload

by Li Jieying

Supervisor ： Professor Zhao Hai

Northeastern University

June 2016

毕业设计（论文）任务书

毕业设计（论文）题目：

面向非等值连接负载的数据生成器研究与实现

设计(论文)的基本内容：

⑴本文研究的是数据生成，旨在实现一种面向非等值连接负载的数据生成器。

⑵根据 sigmod11 的“Data Generation using Declarative Constraints”实现基于基数约束的数据生成器。

⑶根据前人工作的不足，提出关于基于负载的数据生成问题的新思路并实现。

⑷与 sigmod11 所述数据生成器实验对比，验证我们工作的有效性和正确性。

毕业设计（论文）专题部分：

题目： _____

设计或论文专题的基本内容：

学生接受毕业设计（论文）题目日期

第 周

指导教师签字：

年 月 日

面向非等值连接负载的数据生成器研究与实现

摘要

因为数据隐私、模拟真实数据等要求，数据生成器在数据库管理系统测试、数据库应用测试和应用驱动的基准测试中有重要作用，在数据挖掘应用测试中也有所应用。

面向非等值连接负载的数据生成问题是数据生成领域的一大难题，目前针对此问题尚未有合适的数据生成算法。对此本文提出一种基于数据生成与负载特征约束解耦思想的算法 DecAlgo(decoupling algorithm)，此算法支持面向非等值连接负载的数据生成，对于部分等值连接负载和等值负载本文同样支持。本文首先给出每个属性的生成函数，然后根据负载特征约束之间的依赖关系和影响关系调整属性生成函数的定义域，最后再依据属性的生成函数和定义域生成满足负载特征约束的查询实例。本文实现了并发和流式的数据生成，相比 PGM 数据生成器，DecAlgo 算法能够在表层次上实现并行生成，能够高效地利用计算资源且生成的数据可直接落盘，因此消耗的存储资源少。通过针对非等值连接负载的实验以及与 PGM 数据生成器针对传统查询负载的对比实验展示了 DecAlgo 算法的通用性和高效性。实验结果显示针对非等值连接负载 DecAlgo 的误差率在 2%内；针对传统查询负载 DecAlgo 的误差率平均在 5%内；DecAlgo 的数据生成性能仅受磁盘带宽影响。

本文第一章介绍了数据生成器的应用背景；第二章阐述了相关工作；第三章介绍了 DecAlgo 算法所用到的数学方法；第四章介绍了 DecAlgo 算法的流程；第五章是实验；第六章总结全文并展望。

关键词：非等值连接负载；数据生成；负载特征约束；解耦

The Research and Implementation of Data Generator Oriented to Non-Equality Join Workload

Abstract

Because of data privacy, simulating original data, etc, Data generator in the database management system testing, database application testing and benchmarking plays an important role. Also, it could be used in testing of data mining application.

It is a hard problem which use the non-equality join workload to generate data in data generation, and there is no applicable algorithm to solve the problem. This thesis presents the algorithm DecAlgo(decoupling algorithm) that can handle the problem and is based on such thought which decouples data generation from workload constraints, It can not only support non-equality join workload but also part equality workload. The algorithm specify generation function for each attribute first, then alter domains of functions according to dependence and influence which are between workload constraints, finally the algorithm use functions and related domains to generate database instance that satisfy all workload constraints. It specify all value concurrently in the schema level without prior data by making use of computing efficiently compared to the PGM, after generating one row data it flush it into disk, so the cost of memory resource is low. The result from non-equality join workload experiment and the contrast experiment with PGM in traditional query workload shows the universality and efficiently of our work. DecAlgo's error rate is under 2% when the non-equality join workload as input, DecAlgo's average error rate is under 5% in the condition of traditional query workload, the data generation performance of DecAlgo is only affected by the disk bandwidth.

In the Chapter one, it gives the background of application; it introduces related works in the Chapter two; it makes a introduction about math method which is used in DecAlgo; this thesis presents the DecAlgo algorithm in Chapter four and shows the experiment in Chapter five; Chapter six concludes the paper and prospects.

Keywords： non-equality join workload; data generation; workload constraint; decouple

目录

第1章 绪论

1.1 研究背景

数据库的发展主要有三种推动力：硬件的进步、数据类型多样化、应用场景的变化。随着半导体技术的不断发展，通过摩尔定律可知硬件处理能力提升飞快，带动数据库性能提升巨大、支持的功能增多；计算机技术的发展，使得原本现实社会所产生的数据类型更多，数据规模更大，以此驱动数据库的发展；而新应用场景的出现带动数据库发展出新的类别：传统事务处理、分布式事务处理、分析性、流处理、键值型等，例如分布式数据库在对大数据的存储和对高并发处理上

表 1.1 主流数据库及其类别

DBMS	Database Model
Oracle	Relational DBMS
MySQL	Relational DBMS
Microsoft SQL Server	Relational DBMS
MongoDB	Document store
PostgreSQL	Relational DBMS
DB2	Relational DBMS
Microsoft Access	Relational DBMS
Cassandra	Wide column store
Redis	Key-value store
SQLite	Relational DBMS
Elasticsearch	Search engine
SAP Adaptive Server	Relational DBMS
Teradata	Relational DBMS
Solr	Search engine
HBase	Wide column store
Hive	Relational DBMS
FileMaker	Relational DBMS
Splunk	Search engine
SAP HANA	Relational DBMS
Neo4j	Graph DBMS

有天然优势，而流处理类型的数据库则是服务于流应用，最近几年流行起来的数据挖掘和数据分析则需要分析性数据库的支撑。客户为了评测系统对于个人业务需求的可用性，需要对候选数据库系统进行测试，通常需要依赖公开 benchmark，如事务处理性能委员会（Transaction Processing Performance Council, TPC）制定的 benchmark 系列。但 TPC 系列代表的应用不能反映所有应用的需求，为了灵活测试数据库性能、可用性、鲁棒性、可扩展性，良好的数据生成器是不可少的工具之一。尽管目前很多 benchmark 也携带数据生成器，但是其可用性受限于对实际应用需求支持不够，无法定制化等。为了支持灵活、可扩展以及与现实应用相符的数据生成，涌现了大批数据生成器的研究工作，包括以下四个应用场景：

数据库管理系统测试（DBMS Testing）：当今数据库管理系统的一个常用测试方法是先生成一组含有数据的测试数据库，然后在生成数据库上执行相关 SQL 查询。当数据库系统开发人员根据计划或实际需求开发了一个新的数据库管理系统组件，项目测试人员对该组件进行功能测试和性能测试时需要具有某种特征的数据，在此数据上检验开发的组件是否达到预期的效果。不同的功能模块，对数据特征的要求不同，如为了测试混合哈希连接的代码模块，需要一个在外连接属性上具有高倾斜度特征的数据库实例；当研究内存管理器和多哈希连接算子的相互作用时，具有特定中间结果集的数据库实例是必需的。

数据隐私规避和数据库应用测试（Data Masking and Database Application Testing）：如今形形色色、大大小小的公司都在生产经营中使用数据库服务，由于无相关资金预算、缺乏相关技术人员等某些限制，对于一些中小公司来说进行数据库的应用测试代价很高。因而最佳选择是将此测试工作外包给专业的公司或机构。由于需要对内部商业数据保密或保护客户隐私等约束，不能将真实数据交给外包公司以避免泄漏，此时需要有一个数据生成器能生成满足测试目的且保证数据特征与真实数据库相似的数据库实例。例如在一个顾客-产品-订单数据库中，客户年龄和购买商品种类间存在相关性，要求生成的数据库实例也应具有此相关性。

基准测试（Benchmarking）：基准测试是指通过设计科学的测试方法、测试工具和测试系统，能够对测试对象的某项性能指标进行定量和可对比的测试。当今市面上的主流数据库除了支持基本的功能外，在某些方面还具有自己的特殊优势。对于数据库客户来说难以决定要从这些备选中选择哪一个，客户可以阐述应用需求，提出自己关心的性能指标，然后委托专业机构、公司或是自己进行基准测试。标准基准测试如 TPC-H 并不一定就适用客户当前的应用场景，因而其自带的数据生成器 dbGen 有可能无用武之地，所以需要定制一个满足客户需求的

特定数据生成器。

表1.2 TPC的几个benchmark

Benchmark Name	Remark
TPC-C	An On-Line Transaction Processing benchmark
TPC-DC	The new decision support benchmark standard
TPC-E	A new On-Line Transaction Processing (OLTP) workload developed by the TPC-C
TPC-H	An ad-hoc, decision support benchmark
TPC-Energy	A new TPC specification which augments the existing TPC Benchmarks with Energy Metrics developed by the TPC

数据挖掘应用测试（Testing of Data Mining Applications）：利用数据生成器产生某些已知特征，使用数据挖掘工具检验是否能够发现这些特征，以此作为评估数据挖掘工具优劣的一个指标。

通过对以上四种应用场景的阐述，在数据库相关的开发、测试和应用工作中，数据生成器的应用意义深远。

1.2 研究内容与主要贡献

本文研究针对非等值连接负载的数据生成问题（Data Generation Problem），从负载的角度出发，旨在设计高效数据生成算法以及提供灵活的数据生成工具。数据生成问题定义如下：

定义 1.2（数据生成问题）：给定数据库表结构（schema）和一组相关基数约束集合，生成一个满足所有基数约束的数据库实例。

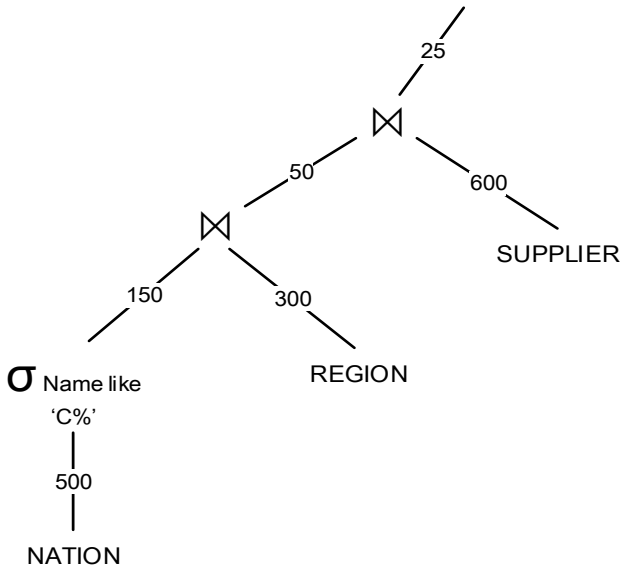


图 1.1 查询约束树

定义 1.3（负载特征，Workload-Feature） 一个 query 的负载表现特征是指在这个 query 执行过程中，各个阶段中间结果集的大小，如图 1.2 所示，NATION 表有 500 条记录，经过选择操作过滤后得 150 条记录，整个 query 最终返回 25 条记录。

定义 1.4（基数约束，Cardinality Constraint） 针对查询计划中某个约束表达式指定返回的中间结果集大小。

例 1.1 query：select * from A where a1<=p1 and a2=p2，基数约束：

a1<=p1 and a2=p2,180，其中，180 代表执行完选择表达式后的中间结果集，即查询负载。

判定数据生成问题已被验证是 NEXP 完全问题（NEXP-complete）[3]，数据生成问题的解决难度非常高。尽管，已有工作已经在这个领域给出一些解决方案，但还是存在可扩展性、可用性等问题。但本文从应用测试的角度出发，发现非等值连接是常见的测试负载之一，因此尝试解决已有数据生成问题之外，尝试解决具有非等值连接负载的数据生成问题，这个工作目前尚未有人涉及。

本文的主要研究目标为设计并实现支持非等值连接负载的数据生成器，同时有效支持传统数据生成器负载。针对现有工作[3]在数据生成过程中属性间的耦合导致生成效率低、可用性不高的缺点，本文提出一种基于数据生成与属性间关联关系解耦思想的算法 DecAlgo(decoupling algorithm)。DecAlgo 算法通过将数据表中各属性值的生成过程函数化，并在此条件下尽可能近似地给出满足所有负载 query 基数约束的参数。对于非等值运算符的基数约束，本文利用属性的生成函数给出近似解；对于具有等值运算符的基数约束，本文通过随机指定等值段位置和非等值解的冲突判断结合，采用启发式算法最大程度地给出近似解，最终计算出所有满足负载的查询约束参数，实现高并发数据生成。本文提出的 DecAlgo 算法复杂度低，仅与基数约束中涉及的属性维度有关；在给出数据表中各属性的生成函数与定义域后，各表甚至各属性便可以完全并发地生成数据；同时本文算法对内存空间的要求低，仅与数据表中属性个数相关，可充分利用计算资源。本文的主要贡献概括如下：

（1） 面向非等值连接负载的数据生成问题尚无相关工作，本文第一次提出在支持已有的数据生成工作的同时，支持对具有非等值连接负载的数据生成。

（2） DecAlgo 生成过程可在属性层次上实现完全并发，对内存要求低，仅与属性个数相关，能够充分利用计算资源。

1.3 组织结构

依据研究内容，本文的组织结构如下：

第一章介绍了本文的研究背景、研究内容和主要贡献点及文章的组织结构。

第二章阐述本文研究内容的相关工作。

第三章提出关于求解基数约束调整参数的算法，命名为 IntegAvg，在 DecAlgo 算法的多个步骤中均需要使用到 IntegAvg。

第四章介绍本文的算法 DecAlgo 实现流程。

第五章给出实验，展示：DecAlgo 在传统负载下的误差率和查询时间；对比 DecAlgo 和已有数据生成器 PGM[3]；输入为非等值约束和部分等值约束条件下，验证 DecAlgo 的准确性和生成性能；通过实验说明 DecAlgo 目前的局限之处。

第六章总结本文的主要研究内容，并提出本文工作的未来展望。

第2章 相关工作

现阶段针对数据库的数据生成相关研究主要分为三类：基于负载特征的工作，基于数据特征的工作和从其它特征出发的工作。

2.1 基于负载特征数据生成器

从负载特征出发生成数据的代表性工作有[1-6]。QAGen[1]的架构如图 2.1 所示，其针对输入的每一个 query 使用 Symbolic Query Engine 生成一个对应的符号数据库（Symbolic Database）且将其存于 DB 中，再借助 Data Instantiator 将符号数据

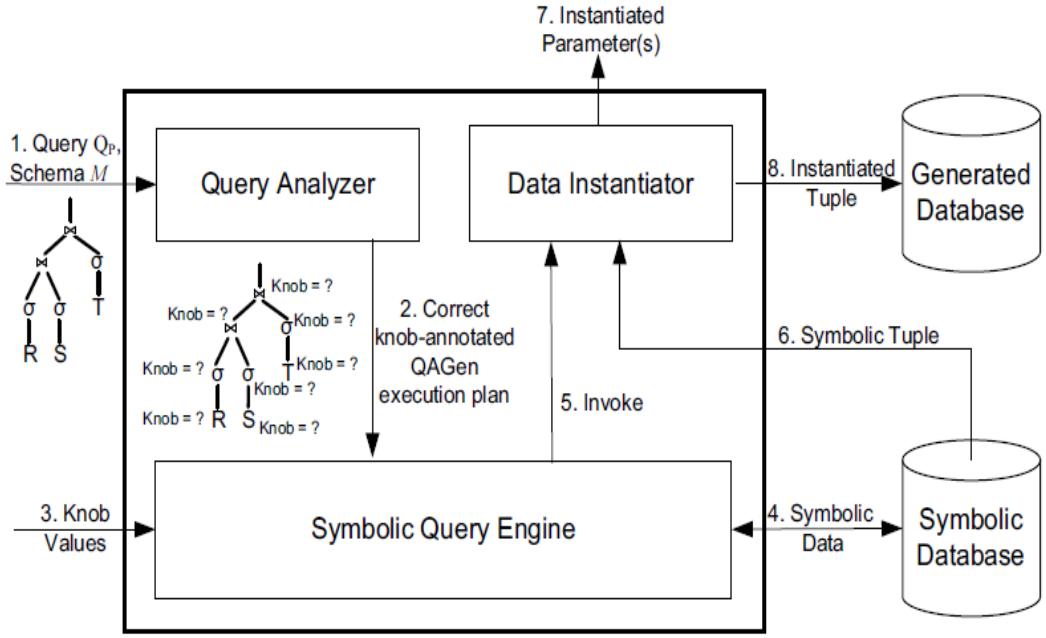


图 2.1 QAGen 架构

库实例化，即将数据库中的符号替换为符合相应表达式的值，最终生成一个单独的数据库实例。针对每个 query 生成独立数据库，但多个 query 不能生成一个数据库实例，其实用性受限；另外，将符号数据库完全放入内存对内存要求较高；再者，它不能支持多样的复杂多表多属性基数约束；最后，算法时间复杂度较高，实验结果表明当生成大规模数据的时候，处理时间将以天为单位。

基于 QAGen[1]的 MyBenchmark[2]的架构如图 2.2 所示，其同样使用了符号查询处理技术来生成符号数据库再将其实例化，这项工作的主要贡献点在于使用 SQP Engine 为每个 query 生成相应的符号数据库后，利用 Symbolic database integrator 尽可能地融合符号数据库，最终尽量达到只剩一个符号数据库的目的。最后利用 Data instantiator 实例化符号数据库。但是其局限在于只能尽力而为并不能保证最终仅生成一个数据库实例，因此存在一定的可用性问题。

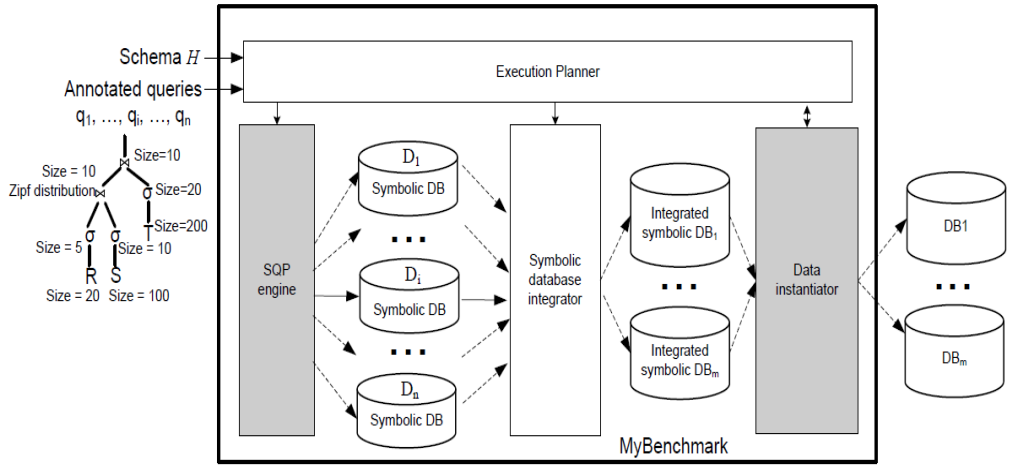


图 2.2 MyBenchmark 架构

在工作[3]中，它提出新的算法处理多个 query 以生成一个数据库实例，核心思想是利用概率图模型（probabilistic graphical models）使生成的数据概率上接近。算法概览如图 2.3 所示，其主要思路如下：

（1）利用输入的基数约束构建出一个马尔科夫网；

（2）将马尔科夫网转变为弦图；

（3）根据三个公式列出满足约束的方程组，求出弦图中所有极大团的边缘概率分布，然后求出整个弦图的联合概率分布；

（4）最终利用求出的联合概率分布取样生成数据。

INPUT: A data generation problem involving $R(A_1, \ldots, A_n)$ and constraints $C_1, \ldots, C_m$ and $|R| = N$
OUTPUT: A generative probability distribution $p(\mathcal{X})$
1. Construct the Markov network $G = (\mathcal{X}, E)$
2. Add edges to G to get a chordal graph $G_c = (\mathcal{X}, E_c)$
3. Identify maximal cliques $\mathcal{X}_{c1}, \ldots, \mathcal{X}_{cl}$ in G_c
4. Solve for marginal distributions $p(\mathcal{X}_{c1}), \ldots, p(\mathcal{X}_{cl})$
5. Use chordal graph property to construct $p(\mathcal{X})$ from the marginals $p(\mathcal{X}_{c1}), \ldots, p(\mathcal{X}_{cl})$

图 2.3 利用概率图模型的算法概览

其主要不足是需要针对基数约束进行大量人工预处理；多维度数值型属性的基数约束生成算法复杂度太高；生成过程冗余较多；大数据量下的外键生成时间代价很高；其属性的域不能太大。这些缺陷使得这该算法实用性有待提高。另有工作[5,6]利用最大熵原理进行基数估计，但此方法难度太大，并且以现有的解决方法无法处理复杂约束。

2.2 基于数据特征数据生成器

从数据特征出发设计实现数据生成器的有工作[7-10]。工作[7]阐述了一个可以

满足表内、表间依赖关系并生成数据库实例的灵活框架模型，避免研究者们各自开发数据生成器从而导致生成的数据分布无法重现、分析等问题；该框架采用DGL（Data Generation Language）描述待生成数据库的数据分布特征。

工作[8]介绍了一种具有通用、高扩展性的数据生成器，能够为 OLTP（On-Line Transaction Processing），OLAP（On-Line Analytical Processing）和数据流应用产生测试数据。该工具利用图模型来指导数据生成，可以支持比较复杂的表内和表间的关联关系，对于精度要求较高、数据量大的需求也能部分满足。

工作[9]提供一个基于 XML 的数据约束描述语言 SDDL（Synthetic Data Description Language），虽然概念上与 DGL[7]相似但有些重要区别，如 SDDL 基于 XML 而 DGL 语法类似 C 语言，DGL 为流数据生成器服务而 SDDL 更易于理解和模块化。SDDL 可以在多核的环境下尽可能地并发生成数据。工作[10]研究了大数据系统下的数据生成问题，阐述了一种数据生成工具包，利用伪随机数字生成器实现数据的有效并行生成，以期减少大数据生成时间耗费。

2.3 其它工作

还有其它一些不属于前面两个分类的数据生成器工作。工作[11]的方法建立在非均匀取样和数据合成的基础之上，目标是为数据流应用（Dataflow Programs）生成样例数据（Example Data）。该技术在雅虎的网页分析应用中表现不错。工作[12]在原始数据库基础上生成模拟数据库，利用从原始数据库中精心挑选的统计特性生成数据，既保证模拟数据库与原始数据库的负载性能特征一致，也可以规避原始数据库的隐私性。

工作[13-15]用于生成非关系型数据。其中工作[13]生成 XML 数据，而工作[14,15]生成图数据；工作[16]根据一个 query 和一个输出，生成一个可能的数据库实例，而生成的 query 在该数据库实例上操作结果即为定义的输出。

与数据生成工作相关还有 query 的生成，相关见工作[17-19]。工作[17]根据一个数据库实例和一个输出，利用 QRE（Query Reverse Engineering）生成满足输出结果的 query；工作[18,19]在一个数据库实例上生成满足基数约束的 query，工作[18]使用了抽样和空间剪枝的技术，工作[19]使用了启发式算法提高搜索效率。

2.4 本章小结

本章将数据库领域内的数据生成工作进行简单的分类、总结介绍：首先介绍从查询负载角度出发的数据生成工作，然后是从数据本身特征出发的数据生成工作，最后介绍一些为非传统数据库生成数据或间接与数据生成问题相关的工作。无论哪一类数据生成器都没有成熟的用于支持不等式约束的数据生成工作。

第3章 IntegAvg

不同于以往的工作，在输入负载前确定约束表达式的常数项，为了将属性间的耦合打开，本文首次提出对此常数项参数化，即通过生成函数和定义域共同确定其数值，将此确定常数项及此常数项对应的定义域序的方法命名为 IntegAvg，DecAlgo 算法中需计算的 index 点即为定义域的序 K，本章将分别介绍：

（1） 单表单属性约束调整参数的计算方法 IntegAvgSS；

（2） 单表多属性约束调整参数的计算方法 IntegAvgSM；

（3） 多表多属性约束调整参数的计算方法 IntegAvgMM。

3.1 IntegAvgSS

IntegAvgSS 描述的是针对单表单属性情况，非等值基数约束和等值基数约束可抽象为属性对应生成函数上的相关计算。对于非等值约束，其中基数抽象为函数不等于某个值的定义域个数，某个值即为约束中的调整参数。对于等值约束，可以理解为在函数的定义域上出现了大小为约束基数的等值段，导致函数的值在此等值段上相等，约束的调整参数即为此时函数的值。以下将从不含有等值段和含有等值段两种角度来介绍 IntegAvgSS 的流程。

定义 3.1（等值段） 定义域上的重复区间，使得此区间对应的属性取值相同。

3.1.1 不含等值段情况

问题 3.1 有函数y = f(x)，定义域为：start = c_0、step = c_1，c_0、c_1为常数。要求在上述函数和定义域确定的值域（大小为 M，即有 M 个 step）上比常数 c 小的元素有 k 个，c_0、c_1、k 和 M 均为已知。求常数 c 的大小和 c 的序 K（即 c 对应的定义域序号）。

针对问题 3.1 本文采取方式如下：

（1）将函数y = f(x)的定义域修正为[$c_0 - \frac{c_1}{2}$, $c_0 + (M-1) * c_1 + \frac{c_1}{2}$]；

（2）计算函数y = f(x)在修正后定义域上的最大值 max 和最小值 min；

（3）给出 c 的一个预测值：c = ((max - min) / M) * k + min；

（4）计算 c 值的近似序 K：令 f(x) = c，解得 x 的解集。根据 x 的解集给出 c 的近似序；

（5）如果 c 的序达到精度要求或者迭代次数达到上限则退出算法，返回 c 的值和 c 的序；否则给出 c 的新预测值（K 为第 4 步计算出来的 c 的序）：min + ((c-min)/K)*k（K>k 时）或者 c + ((max-c)/(M-K))*(k-K)（K<k 时），返回第 4 步。

第 3 步建立在 step 为常数的前提下，第 4 步可以根据 y=c 和 y=f(x)的图像上

下位置关系给出 c 的近似序。

例子 3.1 假设精度要求为 0.1%，迭代次数上限为 20，k=5，函数 y=2x+7，start=3，M=10，则值域为{13，17，21，25，29，33，37，41，45，49}，则返回的 c 值和 K 应为 30 和 5。下面为使用算法的计算过程：

（1）定义域修正为[2，22]；

（2）max = 51，min = 11；

（3）c = (51 – 11) / 10 * 5 + 11 = 31；

（4）令 2x + 7 = 31，解得 x = 12，根据 x = 12，可以计算 c 的序 K 约为：(12 – 2) / (22 - 2) * 10 = 5；

（5）(5 – 5) / 5 < 0.1%，达到精度要求，算法结束。

3.1.2 含有等值段情况

问题 3.2 有函数y = f(x)，定义域为：start = c_0、step = c_1，c_0、c_1为常数，等值段开始序号 i，等值段长度 r。要求在上述函数和定义域确定的值域（大小为 M，即有 M 个 step）上比常数 c 小的元素有 k 个，c_0、c_1、k 和 M 均为已知。求常数 c 的大小和 c 的序 K（即 c 对应的定义域序号）。

针对问题 3.2 本文采取方法如下：

（1）根据定义域将原函数转为分段函数；

（2）计算所有函数在修正后定义域上的最大值和最小值，然后给出综合 max 和 min；

（3）给出 c 的一个预测值：c = ((max - min) / M) * k + min；

（4）计算 c 值的序：分别令分段函数 f(x) = c，解得 x 的解集，根据 x 的解集给出 c 在本函数上的近似序，再综合给出 c 的全局近似序；

（5）如果 c 的序达到精度要求或者迭代次数达到上限则退出算法，返回 c 的值和 c 的序；否则给出新的预测值 c（K 为计算出来的 c 的序）：min + ((c-min)/K)*k（K>k 时）或者 c + ((max-c)/(M-K))*(k-K)（K<k 时），返回第 4 步。

第 2 步中即选择出所有分段函数的最大 max 和最小 min 为全局 max 和全局 min，第 4 步求近似序参照问题 3.1.1 解决办法。

3.2 IntegAvgSM

3.2.1 不含等值段的情况

问题 3.3 有若干函数y = $f_i(x_i)$，x_i的定义域为$D_i = \{x|x = start_i + n * step_i\}$（其中$start_i$和$step_i$为常数，n=0,1,2,……,M-1）。现有函数Y = $\sum f_i(x_i)$，但是所有x_i取值时 n 必须相同（因为这些函数对应同一张表上的不同属性），因

此可知 Y 的值域大小为 M，要求在 Y 的值域上比 c 小的元素有 k 个（k 和 M 是已知的）。求常数 c 的大小和 c 的序 K。

针对问题 3.3 本文采取如下解决办法：

（1）利用化归思想将多个函数的变量转化为同一个；

（2）计算函数 Y=f(t)在定义域上的最大值 max 和最小值 min；

（3）给出 c 的一个预测值：c = ((max-min)/(M+1))*k + min；

（4）计算 c 值的序：令 f(t)=c，解得 t 的解集。根据 t 的解集给出 c 的近似序 K；

（5）如果 c 的序达到精度要求或者迭代次数达到上限则退出算法，返回 c 的值和 c 的序 K；否则再给出新的预测值 c：min + ((c-min)/K)*k（K>k 时）或者 c + ((max-c)/(M+1-K))*(k-K)（K<k 时），返回第 4 步。

由于这些函数是对应同张表上的不同属性，所以第 1 步可将变量转换为同一个，且定义域的大小一致。

3.2.2 含有等值段的情况

在问题 3.3 的基础上，x_i的定义域$D_i = \{x|x = s_i + n * step_i\}$（其中$s_i$和$step_i$为常数，n=0,1,2,……，t,……,t,……,M'）。定义域的大小依然为 M，但是定义域中存在重复的值。

解决思路：将原函数转化为分段函数，假设所有函数上含有重复区间定义域的个数为 n 个，那么分段函数的个数小于等于 2n+1，以重复区间作为分段依据且重复区间自成一段。其余思路与 IntegAvgSS 类似，这里不再赘述。

3.3 IntegAvgMM

3.3.1 不含等值段的情况

问题 3.4 有若干函数$y = f_i(x_i)$，x_i的定义域为$D_i = \{x|x = start_i + n * step_i\}$（其中$start_i$和$step_i$为常数，n=0,1,2,……）。现有函数$Y = \sum f_i(x_i)$，因此可知 Y 的值域大小为$\prod(M_i)$（其中$M_i$为$y = f_i(x_i)$定义域的大小）。求给一个常数 c，要求在 Y 的值域上比 c 小的元素有 k 个。

下面我们将给出针对问题 3.4 的解决办法：

（1）对所有函数$y = f_i(x_i)$的定义域进行修正；

（2）分别计算所有函数的 min 和 max，然后给出综合 min 和 max；

（3）给出 c 的一个预测值：c = ((max-min)/(M+1))*k + min；

（4）计算 c 值的序：通过空间积分给出 c 的序 K；

（5）如果 c 的序达到精度要求或者迭代次数达到上限则退出算法，返回 c

的值和 c 的序；否则再给出新的预测值 c（K 为计算出来的 c 的序）：min + ((c-min)/K)*k（K>k 时）或者 c + ((max-c)/(M+1-K))*(k-K)（K<k 时），返回第 4 步。

第 1 步的定义域修正规则与 IntegAvgSS 相同；第 4 步的计算方法将通过下面例子 3.2 来介绍。

例子 3.2 有两个函数$y = x_1{}^2$，start = 1，step = 1；M = 8 和$y = -x_2$，start =1，step = 1；M = 5，则$Y = \sum f_i(x_i) = x_1{}^2 - x_2$；$x_1$={1,2,……,8}，$x_2$={1,2,……,5}，修正后的定义域为$x_1$=[0.5, 8.5]，$x_2$=[0.5, 5.5]，假设此时 c=28。则计算过程转换为$\int_{0.5}^{8.5}\int_{0.5}^{5.5}(x_1{}^2 - x_2 \le 28)\mathrm{d}x_2\,dx_1$，下图 3.3.1 是通过 Mathematica 的计算结果。

```
In[109]:= e1 = x1^2 - x2 ≤ 28
Out[109]= x1^2 - x2 ≤ 28

In[110]:= Integrate[Boole[e1], {x1, 0.5, 8.5}, {x2, 0.5, 5.5}]
Out[110]= 25.3313

In[112]:= RegionPlot[#, {x1, 0.5, 8.5}, {x2, 0.5, 5.5}] &/@{e1}
```

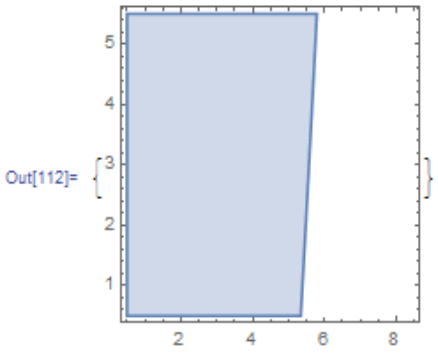


图 3.1 IntegAvgSM 空间积分示意图

由图 3.1 可知 28 的序为：(25.3313/((8.5-0.5)*(5.5-0.5))) * (5*8) = 25，而 28 的实际序也为 25。

3.3.2 含有等值段的情况

在问题 3.4 的基础上，x_i的定义域$D_i = \{x|x = start_i + n * step_i\}$（其中$start_i$和$step_i$为常数，n=0,1,2,……，t,……,t,……,M'）。定义域的大小不变依然为 M，但定义域中存在重复值时。

我们可以将其理解为等值段拉伸了原空间，并且不同定义域（不同维度）上的拉伸有可能还会存在交织。而这些拉伸将会导致原空间变大，这时候我们可以通过补偿性积分来弥补。具体过程如下：

（1）不考虑定义域中重复的值，针对 n=0,1,2,……,M'表示的定义域正常作

积分预算，得到值v_0；

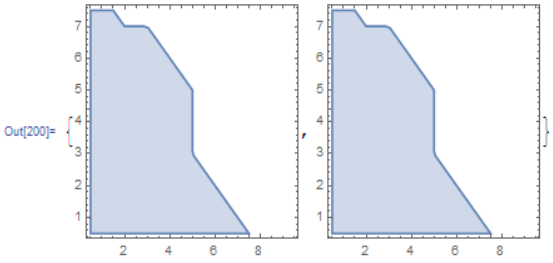
（2）用各重复区间的值去截原空间得到一条线（二维）或一个面（三维）…，然后用截得的这条线的长度或者这个面的面积…乘以该重复值被拉伸的长度，得到值v_1^1、v_1^2、v_1^3…；

（3）两两检测第二步中截得的所有线或者所有面…是否相交，如果相交，则用相交点（值为 1）或者线的长度…乘以该各自维度上被拉伸的长度，得到值v_2^1、v_2^2、v_2^3…；

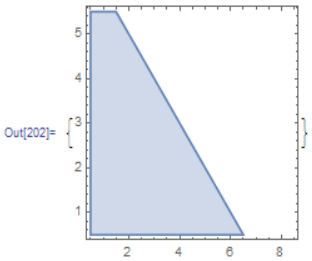
（4）最终积分值为：V= $v_0 + v_1^1 + v_1^2 + v_1^3 + \cdots + v_1^p + v_2^1 + v_2^2 + v_2^3 + \cdots + v_2^q$。

第 3 步源于当两者相交不独立时，其中一个长度或面积发生变化会导致另一个也发生变化。以下通过例子 3.3 介绍补偿性积分的具体做法。

```
In[198]:= e1 = x + y < 7 /. y → y - 2 UnitStep[y - 3] * UnitStep[7 - 3 - x] /. x → x - UnitStep[x - 0] * UnitStep[7 - 0 - y]
Out[198]= x + y - UnitStep[x] UnitStep[7 - y] - 2 UnitStep[-3 + y] UnitStep[4 - x + UnitStep[x] UnitStep[7 - y]] < 7

In[199]:= e2 = x + y < 7 /. x → x - UnitStep[x - 2] * UnitStep[7 - 2 - y] /. y → y - 2 UnitStep[y - 2] * UnitStep[7 - 2 - x]
Out[199]= x + y - 2 UnitStep[5 - x] UnitStep[-2 + y] - UnitStep[-2 + x] UnitStep[5 - y + 2 UnitStep[5 - x] UnitStep[-2 + y]] < 7

In[200]:= RegionPlot[#, {x, 0.5, 9.5}, {y, 0.5, 7.5}] & /@ {e1, e2}
```



```
In[201]:= e3 = x + y < 7
Out[201]= x + y < 7

In[202]:= RegionPlot[#, {x, 0.5, 8.5}, {y, 0.5, 5.5}] & /@ {e3}
```



```
In[203]:= Integrate[Boole[e1], {x, 0.5, 9.5}, {y, 0.5, 7.5}]
Out[203]= 31.

In[204]:= Integrate[Boole[e2], {x, 0.5, 9.5}, {y, 0.5, 7.5}]
Out[204]= 31.

In[205]:= Integrate[Boole[e3], {x, 0.5, 8.5}, {y, 0.5, 5.5}]
Out[205]= 17.5
```

图 3.2 含有等值段 IntegAvgMM 示意图

例子 3.3 先给一个二维的情形：x∈[0.5,8.5]，y∈[0.5,5.5]，求当 x+y<7 时的空间大小。但是此时，x 在定义域 2 处存在长度为 2 的等值段（即 x=2 被拉开了一个单位），y 在定义域 3 处存在长度为 3 的等值段（即 y=3 被拉开了两个单位），如图 3.2 所示。补偿积分的 3 步结果分别是 17.5、(5-0.5)*1+(4-0.5)*2、1*2，3 步结果之和为 31，符合计算结果。

3.4 本章小结

本章介绍了从算法步骤中所需用到的数学计算方法抽象出来的 IntegAvg，分别从单表单属性，单表多属性，多表多属性抽象对应 IntegAvgSS、IntegAvgSM 和 IntegAvgMM。IntegAvgSS 和 IntegAvgSM 的主要思路是通过逼近思想和分段函数来解决，IntegAvgMM 则将离散的数据连续化，利用高等数学的积分求解。然后通过几个简单的例子表现算法的具体做法。

第4章 DecAlgo

本章描述的是面向 query 的数据生成器的算法流程，主要针对非等值连接负载，并在一定程度上支持等值连接负载。当等值连接负载的数量控制在一定范围内时，算法也表现良好。

4.1 预备定义

在介绍算法流程之前，首先对流程中需要使用到的概念、定义进行介绍。

4.1.1 算法输入

该算法的输入是一组 query 所涉及的表结构以及针对 query 中间结果集的基数约束，输出是一个数据库实例和满足基数约束的调整参数。

4.1.2 依赖关系

本文将基数约束分成两类：等值基数约束和非等值基数约束。等值基数约束是指针对等值约束表达式的基数约束；非等值基数约束指针对非等值约束表达式的基数约束。

每个 query 的执行计划就是一个查询树，那么针对该 query 各个约束表达式的基数约束显然也是这样的树形依赖关系，上层节点依赖于下层节点。本文将具体的依赖关系分成如下四类（–>前面的为下层节点，后面的为上层节点，–>后面的依赖于前面的）：

（1）非等值基数约束 –> 非等值基数约束；

（2）非等值基数约束 –> 等值基数约束；

（3）等值基数约束 –> 非等值基数约束；

（4）等值基数约束 –> 等值基数约束。

以下通过例子 4.1 对依赖关系做个介绍。

例子 4.1 query：select * from A where a1 <= p1 and a2 = p2，可得此 query 中存在依赖关系 a1<=p1 –> a2=p2。

4.1.3 影响关系

定义 4.1（影响关系） 基数约束 A 对基数约束 B 有影响可分为两种情况：A 是一个等值基数约束，B 是一个非等值基数约束，A 相关的约束表达式中涉及的属性与 B 相关的约束表达式中涉及的属性有交集；基数约束 B 依赖于基数约束 A。

只要两种情况中任意一种成立即可认为 A 对 B 有影响。关于第一种情况，

由于算法的大致处理流程（本章后面会介绍）与数据库语法树的执行流程类似，类似由叶子向根节点执行的流程，且 A 和 B 的两者属性有交集，所以 A 对 B 有影响。第二种情况则是由于语法树的原因，很好理解。

例子 4.2 query：select * from A where a1<=p1 and a2=p2，有影响关系：a1<=p1 影响 a2=p2。

4.1.4 属性分组

定义 4.2（属性分组）如果属性 a 与属性 b 同在某个等值约束表达式中，那么属性 a 与属性 b 同属于一个属性组。

当 a 和 b 同属一个属性分组且 b 和 c 也同属一个属性分组，则 a、b、c 同属一个属性分组，即属性分组存在传递性。

例子 4.3 query：select * from A,B,C where a1 = b1 + p3 and b2 = c1 + p4 and c2 = p5，则有两个分组，分组 1：a1，b1；分组 2：b2，c1。

4.1.5 约束分层

定义 4.3（约束分层） 基数约束层次化，某个基数约束属于 i 层当且仅当其依赖于 i-1 层基数约束，不依赖于任何基数约束的基数约束属于 0 层。

4.2 算法流程

如果两个 query 的执行计划相同，并且在同一个数据库实例上整个执行过程中各个阶段的中间结果集大小也完全相同，那么本文认为这两个 query 的负载表现特征是一致的。从另一个角度可以这么说：一个系统在该数据库实例上处理这两个 query 的难度是相同的。因此本文提出以下假设：

假设 4.1 针对 query 的约束表达式加一个常数的参数调整是不影响 query 的负载表现特征的。

例子 4.4 query：select * from A inner join B on A.a1 = B.b1 + p3 where A.a2 > p1 and B.b2 > p2;（p1、p2、p3 为三个常数），根据我们的经验，该 query 的执行流程如下：

（1） 先从 A 表中选择出满足约束表达式 A.a2 > p1 的记录；

（2） 再从 B 表中选择出满足约束表达式 B.b2 > p2 的记录；

（3） 对上面两个中间结果集作连接运算，返回最终结果。

针对三个约束表达式的基数约束分别为：r1、r2、r3，这三个基数约束分别指上述三个中间结果集的大小。如果我们改变了 p1、p2、p3 的大小，并且依然保证 r1、r2、r3 的大小不变，此时该 query 的负载表现特征明显是没有发生任何改变的。

表 4.1 算法流程

输入：基数约束，表结构（schema）
1. 对基数约束进行预处理
2. 求基数约束间的依赖关系
3. 求基数约束间的影响关系
4. 为每个属性随机产生生成函数
5. 将等值基数约束蕴含信息注入相应定义域中
6. 计算满足所有基数约束的调整参数
7. 根据生成函数及定义域生成数据

表 4.1 阐述了本文 DecAlgo 的流程，因为本文对不同类型的基数约束依赖关系并不完全支持，所以需要对输入的基数约束中的两种情况进行说明和预处理。

情况 1：根据前面已经介绍的依赖关系定义，我们可以将基数约束依赖关系分成 4 类。对于“等值基数约束 –> 非等值基数约束”和“等值基数约束 –> 等值基数约束”依赖关系，如果–>后面的基数约束涉及的属性集合含于前面的基数约束涉及的属性集合，那么必须将前后基数约束的大小调成一致。虽然对这种特殊情况不能很好地支持，但是这种基数约束指定是不符合实际应用场景的。通过以下例子来说明：

例子 4.5 query：select * from A, B where a1 = b1 + p1 and a1 > p2，基数约束为：a1 = b1 + p1, r1 –> a1 > p2, r2。

此时本文要求两个约束对应的基数 r1 和 r2 必须相等。在真实应用场景中，不会出现这种情况，原因是如果–>后面的基数约束涉及的属性集合真含于前面的基数约束涉及的属性集合，生成的语法树应该是先执行–>后面的基数约束相关的约束表达式判断，这样便与给定的基数约束依赖关系矛盾；如果两个集合相同，那么也应该是同时执行约束表达式判断的。因此本文在实现中，默认这种情况不会出现。

情况 2：对于未指定依赖关系的基数约束，但是算法运行时需要指定的（即在语法树中两个约束表达式可以同时进行判断，这是由行式存储导致的。如 a1 <= p1 and a2 = p2, 180; a1 = b1 + p3 and a2 + b2 > p4, 980），可将其具体分为如下几种类别：

（1）全部为非等值基数约束

（2）含有一个等值基数约束

（3）含有多个等值基数约束

在第一种类别下，该组基数约束只显式地保留最后一个基数约束参与算法后续处理，本组的剩余约束仅在计算调整参数步骤时才考虑，通过在计算调整参数时令它们在定义域都为真实现。

在第二种类别下，该组基数约束只显式地保留唯一的等值基数约束参与算法后续处理，本组其余约束仅在计算调整参数时考虑，通过在计算调整参数时令它们在定义域都为真实现。

在第三种类别下，保留该组的所有等值基数约束参与算法后续处理，其余约束仅在计算调整参数时考虑，通过在计算参数时令它们在定义域都为真，所有保留的等值基数约束的基数都指定为该组基数约束的基数值）。例如：对于 a1 <= p1 and a2 = p2, 180 应只保留 a2 = p2, 180。对于 p1 的计算时，满足让 a1 <= p1 在定义域上都为真即可。

4.3 为属性产生生成函数

为输入的表结构中每个属性随机产生一个生成函数：$y = c_0 x + c_1$。其中c_0、c_1是随机产生的常数，对于一个整形的属性，它们应该都是整数；而对于一个非整形的属性，它们可能为小数。当然，还得随机给出这些生成函数的定义域$start_i$和$step_i$，其中$start_i$为定义域的起点，$step_i$为步距，初始 M 等于相应属性所在表的大小。

因为属性之间的生成函数独立随机生成，且在未将等值基数约束信息注入定义域（即修改定义域）前，各属性之间不应该存在有某种关联，例如两个属性定义域的各自某一区间所对应的属性值存在常数差或倍数差。为了解决这个问题，本文在生成函数的实现时尽可能让所有函数的c_0和c_1各不相同，且c_0和c_1的值都从一定范围内的质数随机取得。

4.4 调整定义域

首先将经过预处理步骤的基数约束进行层次化，接着依次对每层中的基数约束按照表 4.2 进行处理。

表 4.2 定义域调整算法

Algorithm 1: 定义域调整算法

input: 生成函数 *f*，依赖关系，影响关系，分组信息，单层最大迭代次数 *k*
output: 生成函数

1. 对基数约束分层，共 *n* 层；
2. *i* ← 0；
3. **for** *i* < *n* **do**
4. *i* 层等值约束→ *EC*；
5. *iter* ← 1；

```
6.    EC 影响的非等值约束→ INC;
7.    for nc∈INC do
8.      求满足 nc 的定义域范围;
9.    for ec∈EC do
10.     规划 ec 的等值段位置;
11.   EC 影响的非等值约束→ INC;
12.   EC 影响的等值约束→ IEC;
13.   for nc∈INC do
14.     if nc 精度低于阈值 then
15.       相关属性→ TMP;
16.   for ec∈IEC do
17.     if ec 等值段位置越界 then
18.       相关属性→ TMP;
19.   if f 误差小于 BEST then
20.     BEST ← f;
21.   所有相关分组属性→ TMP;
22.   if TMP 不为空 && iter <= k then
23.     TMP 相关等值约束→EC;
24.     iter++; goto 9;
25.   if iter = k then
26.     BEST → f;
27.   i++;
```

4.4.1 计算非等值基数约束 index 点

即定义域调整算法中的求满足非等值约束的定义域范围。需要注意的是，在对任何基数约束作相关操作前，我们需要先确定其有效定义域，对于 0 层的基数约束，其有效定义域当然为初始定义域即相应表大小，而对于非 0 层的基数约束，其定义域在初始定义域和其依赖的所有基数约束基础上共同确定。

具体计算方法参照我们在第 3 章给出的 IntegAvg 描述，单表单属性的基数约束抽象成 IntegAvgSS，单表多属性的抽象为 IntegAvgSM，多表多属性的抽象为 IntegAvgMM。待计算的 index 点即为 IntegAvg 中 c 的序 K。

4.4.2 等值基数约束 index 点指定

即定义域调整算法中的规划等值段位置。我们首先介绍 index 点的指定顺序：多表多属性，单表单属性。之所以这么安排顺序，因为多表多属性更容易引起冲突（包括等值冲突和与非等值约束 index 点的冲突），对于单表多属性的等值基数约束，本文不打算支持，原因在后面会解释。接下来分别阐述针对多表多属性和单表单属性的操作流程。

多表多属性操作流程：先确定各属性上的等值段长度，在各属性上随机一个 index 点，将相应等值段长度插入。

根据数据库理论，我们将等值约束的基数作为约束相关属性的笛卡尔积，为了能够更加合理，属性的乘积因子之间存在倍数关系，其来源于属性所在表大小

之间的比例大小。

$$\Pi a_i x = c \ , \ 1 \le i \le n \tag{4.1}$$

$$\frac{c}{\Pi a_i x} = x_n \ , \ 1 \le i \le n-1 \tag{4.2}$$

式子(4.1)保证各等值段笛卡尔积不超过约束基数，式子(4.2)是在式子(4.1)已求解完的基础上单独求解最后一个属性的等值段长度，原因是数据库表的记录数必然为整数，式子(4.1)的解有可能需要进行近似取整，为了保证基数准确，将前面所有属性取整后的偏差全部转移至最后一个属性。下面通过例子 4.6 直观地介绍求等值段长度的具体做法。

例子 4.6 有基数约束 a1 = b1 + p3, 基数为 100，a1 所在 A 表大小为 1000 条记录，b1 所在 B 表大小为 20,000 条记录，则 A、B 两表大小比例为 1:20，假设 a1 的等值段长度为 x，则可得 x*20x=100，求得 x≈2，则 b1 等值段为 100/2=50。

在求出等值段的长度后接下来是为每个等值段寻找相应的插入点，即在约束相关属性的定义域上找到等值段的开始点。我们采用在定义域上随机的方法，但是随机到的 index 点在插入等值段后，需要保证尽量不能和在此定义域上的其他已存在等值段产生冲突，我们称之为等值冲突，也尽量不影响非等值基数约束的精度。以下介绍等值冲突和解决办法。

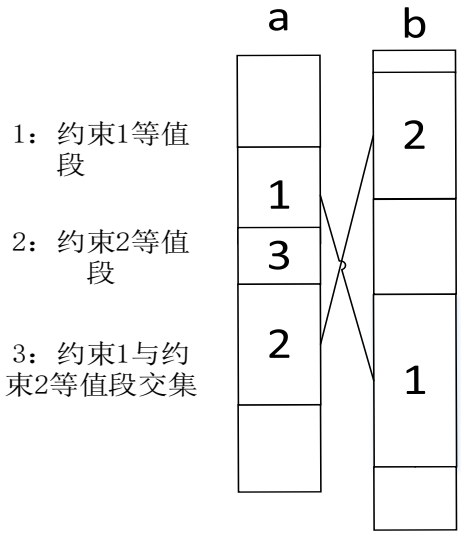


图 4.1 等值冲突示意图

图 4.1 中 a 和 b 分别表示不同数据表的两个属性的元组情况，其中 3 为等值段 1 和 2 的交集部分。现考虑此种情况，a 上等值段 1 长度为 10（即有 10 条记录），b 上等值段 1 长度为 30，输入的相关约束 1 的基数为 300；a 上等值段 2 长度为 15，b 上等值段 2 长度为 20，输入的相关约束 2 的基数为 300；图中 3 部分的长度为 3，则约束 1 的实际基数为 300+3*20=360，约束 2 的实际基数为

300+3*30=390。此即为等值冲突，将影响等值基数约束的准确性。

因此我们提出解决等值冲突的思路：

（1）各属性上的等值段随机组合（即涉及的属性上都得拿一个等值段出来），判断是否有某个组合的计算值与之前等值基数约束相关的约束表达式有冲突。如果有冲突的话，则重新随机一个 index（多次都不满足，则取 index 的历史最优值）。各个属性上的 index 点是逐个指定的；

（2）判断该属性上的单表单属性等值约束是否与该 index 点等值段冲突。

在指定 index 点时只需查看该属性相关的多表多属性等值基数约束，其实对于冲突的判断可以更严苛（n-1 判断），但是无论如何都无法保证精确。

以下通过例子 4.7 来解释为何需要尽可能不影响非等值基数约束的精度：

例子 4.7 某个属性的生成函数为 y = 2x + 7, start = 3, step = 2, M = 10, (i = 4, r = 2; i = 6, r = 2)即在定义域 i=4 和 i=6 的位置分别存在长度为 2 的等值段，则值域为：13, 17, 21, **25**, **25**, 29, **33**, **33**, 37, 41。此时如果 k = 5，那么我们无法返回一个值使得上述值域中比其小的元素有 5 个，因为取 25 时则计算结果 c 的序 K 可为 4 也可为 5。

针对随机 index 点，为了能够更快更好地取到符合条件的 index 点，可以采取尽可能远离所有已计算出的非等值基数约束的 index 点，比如仅在这些 index 点分割的各段定义域中间 80%区间中随机等值约束的 index 点或者尽可能选择较小的 index 点，避免因为存在的等值段过多而被“挤出”定义域。

单表单属性操作流程：此时等值段长度显而易见不必求解，约束的基数即为等值段长度，在随机 index 点时关于等值冲突问题和非等值基数约束的精度问题的处理与多表多属性的处理完全相同。

针对单表多属性的操作流程：这种情形的处理是最麻烦的，因为等值段的动态增加会使之前确定的 index 点发生改变。并且在现实应用中，针对单表多属性的等值基数约束也是很少的，所以本文打算不支持该情形。

等值约束 index 点指定小结：先处理多表多属性等值基数约束，再处理单表单属性，不支持单表多属性情形；index 点逐个属性添加进去，每随机一个 index 点需要做相关单表单属性和多表多属性等值基数约束的冲突判断且随机 index 点时应尽可能远离非等值基数约束的 index 点（不仅仅是本层次的）。

4.4.3 精度判断

即定义域调整算法中的 9-24 步，由于 4.4.2 中指定等值基数约束的 index 点时添加的等值段可能影响非等值基数约束的精度，因此需要做两个判断：

（1）因此在指定完本层次的等值基数约束 index 点后需要对其影响的非等

值基数约束的精度进行判断。在新的定义域条件下使用 IntegAvg 算法对相应 c 的序 K 进行计算，并和输入的基数求精度，与输入精度对比判断。

（2）判断各等值基数约束的等值段 index 是否依然在前驱非等值基数约束确定的定义域中，即不被“挤出”定义域。

如果上述两个判断都符合，则层次增 1（若不存在该层次，则返回新的含有等值段的定义域）；否则对于 a 和 b 不符合要求的相关属性所在的属性组中所有属性相关的本层次等值基数约束都重新进行 4.4.2 步骤的操作，多次都不满足，则取历史最优值。

第5章 实验结果与分析

5.1 实验环境

本章的实验环境基于如下表 5.1 所示。其中操作系统为 Windows10 Pro；处理器为 Intel i5-4460，四核，单核最大主频 3.4GHz；内存为第三代 8GB；硬盘每分钟 7200 转并带有 64MB 缓存；开发环境有 Eclipse，JDK1.8，MySQL5.6 和 Mathematica10。

表 5.1 实验环境

实验环境	详细信息
操作系统	Windows 10 Professional 10586
CPU	Intel i5-4460
内存	8GB DDR3 1600MHz
硬盘	缓存：64MB 转速：7200rpm
开发环境	Eclipse JDK1.8 MySQL5.6 Mathematica10.0

5.2 实验数据

本章实验分为四部分，为表述方便，本文数据生成器简称为 DecAlgo，基于概率图模型数据生成器简称为 PGM[3]。实验内容主要体现为以下几个方面：

（1）针对经过改造的传统查询负载，DecAlgo 数据生成误差率和查询响应时间；

（2）针对经过改造的传统等值查询负载，DecAlgo 和 PGM 的误差率、生成时间和存储消耗对比；

（3）针对输入为单表单属性，单表多属性和多表多属性非等值负载时，DecAlgo 生成数据的正确性和效率;

（4）通过单表单属性实验说明 DecAlgo 数据生成器目前的局限性。

5.2.1 DecAlgo 数据生成效率和查询响应时间约束定义

实验 1 将选取传统 bechmark 中的 TPC-H 的 Q1-Q10（不含 Q4，Q7，Q8）7 个 query 改造后作为负载，从 7 个 query 中提取约束作为输入，通过 TPC-H 自带的数据生成器 dbgen 分别生成 1G 和 0.1G 的数据，以此生成数据指定 7 个查询负载即规定查询算子的中间结果集大小。表 5.2 为提取出来的作为实验 1 的输入约束信息。表结构中属性设置与表 5.3 相同。

表 5.2 TPC-H 的 7 个 query 约束信息

编号	约束表达式	1G 中间结果集	0.1G 中间结果集	依赖信息
1	lineitem.lshipdate<=p1	5956891	5956890	
2	part.ptype=p2	40058	4005	
3	part.psize+region.rname+partsupp.pssupplycost	460	46	3 ->2
4	customer.cmktsegment=p4	30142	3014	
5	Orders.oorderdate<p5	727305	72730	
6	Lineitem.lshipdate>p6	3241776	324177	
7	customer.ccustkey-orders.ocustkey=p7	1500000	150000	7->5 7->4
8	lineitem.lorderkey-orders.oorderkey=p8	6001215	600121	8->5 8->6
9	orders.oorderdate>=p9	1046266	104626	
10	orders.oorderdate<p10	681331	68133	
11	lineitem.lsuppkey-supplier.ssuppkey=p11	6001215	600121	11->6
12	customer.cnationkey-supplier.snationkey=p12	150000	15000	
13	lineitem.lshipdate>=p13	4336142	433614	
14	lineitem.lshipdate<p14	909455	90945	14->13
15	lineitem.ldiscount<p15	3819096	381909	
16	lineitem.ldiscount>p16	3427089	342708	
17	lineitem.lquantity<p17	752249	75224	17->15 17->16
18	part.pname<p18	10664	1066	
19	part.ppartkey-lineitem.lpartkey=p19	319404	31940	19->18
20	orders.oorderdate>=p20	1103335	110333	
21	orders.oorderdate<p21	453734	45373	
22	customer.ccustkey-orders.ocustkey=p22	57069	5706	22->20 22->21
23	lineitem.lorderkey-orders.oorderkey=p23	228772	22877	23->20 23->21

5.2.2 DecAlgo 与 PGM 等值约束对比实验约束信息定义

由于 PGM 对等值约束支持较好，针对 TPC-H 的 8 个 query 进行了专门的实验验证；而本文的关注点在于解决具有非等值约束的数据生成问题，但在等值约束上也能支持。由于 PGM 需要大量人工 query 改造，因此实验 2 只选择在 TPC-H 的 Q2 和 Q5 进行误差率、数据生成性能和存储消耗对比。

```
select
        s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone,
        s_comment
from
        part, supplier, partsupp, nation, region
where
        p_partkey = ps_partkey and s_suppkey = ps_suppkey and p_size = [SIZE]
        and p_type like '%[TYPE]' and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey and r_name = '[REGION]'
        and ps_supplycost = (
            select
                min(ps_supplycost)
            from
                partsupp, supplier, nation, region
            where
                p_partkey = ps_partkey and s_suppkey = ps_suppkey
                and s_nationkey = n_nationkey and n_regionkey = r_regionkey
                and r_name = '[REGION]'
            )
order by
         s_acctbal desc,n_name, s_name, p_partkey
```

图 5.1 TPC-H 的 Q2

```
select
        n_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue
from
        customer,
        orders,
        lineitem,
        supplier,
        nation,
        region
where
        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and l_suppkey = s_suppkey
        and c_nationkey = s_nationkey
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = '[REGION]'
        and o_orderdate >= date '[DATE]'
        and o_orderdate < date '[DATE]' + interval '1' year
group by
        n_name
order by
        revenue desc;
```

图 5.2 TPC-H 的 Q5

由于 PGM 对输入需要进行处理，对 Q2 和 Q5 改造后提取约束，对所涉及到的表进行改造并添加基数约束，表 5.3 和表 5.4 分别为实验 2 所需的表结构和基数约束。

表 5.3 实验 2 表结构

表名	表大小	属性列	备注
PartSupp	800	Partsuppkey	主键
		Ps_availqty	
		Ps_supplycost	
		Ps_partkey	外键，参照 Part 表的 P_partkey
		Ps_suppkey	外键，参照 Supplier 表的 S_suppkey
Part	200	P_partkey	主键
		P_name	
		P_type	
		P_size	
		P_retailprice	
Supplier	10	S_suppkey	主键
		S_name	
		S_address	
		S_acctbal	
		S_nationkey	外键，参照 Nation 表的 N_nationkey
Nation	25	N_nationkey	主键
		N_name	
		N_regionkey	外键，参照 Region 表的 R_regionkey
Region	5	R_regionkey	主键
		R_name	
Customer	150	C_custkey	主键
		C_name	
		C_address	
		C_phone	
Orders	1500	O_orderkey	主键
		O_totalprice	
		O_orderdate	

续表 5.3 实验 2 表结构

		O_custkey	外键，参照 Customer 表的 C_custkey
Lineitem	6000	L_lineitemkey	主键
		L_linenumber	
		L_discount	
		L_tax	
		L_partsuppkey	外键，参照 PartSupp 表的 Partsuppkey
		L_orderkey	外键，参照 Orders 表的 O_orderkey

表 5.4 实验 2 基数约束

约束编号	约束表达式	基数
1	P_size=5	20
2	P_size+P_type=3	40
3	3*P_type=12	4
4	R_name=7	3
5	Ps_supplycost=2	400
6	Ps_supplycost+S_address*P_retailprice=84	25
7	N_name-R_name=1	5
8	R_name=2	2
9	O_orderdate=7	300
10	O_orderdate=5	600
11	O_orderdate=1	200

5.2.3 DecAlgo 非等值实验约束定义

实验3从完全无等值约束和有部分等值约束两种情况来验证DecAlgo的可用性和有效性。每种情况再细分为单表单属性，单表多属性和多表多属性三种类别约束。目前没有关于多表间非等值连接的 benchmark，但非等值连接负载在实际中有所应用，如 LBS 中可通过用户信息表与地点信息表之间的非等值连接约束进行范围查询，可用非等值连接约束对 DBMS 进行测试，因此本文设定的表结构和基数约束将尽可能体现非等值约束出现的各种情况。为了表达简便，以下表属性列的数据类型都为整型，对于浮点型数据类型本文同样支持。

表 5.5 为单表单属性约束的表结构，表 5.6 和表 5.7 分别为无等值约束和包含

部分等值约束的单表单属性基数约束。

表 5.5 单表单属性表结构

表名	表大小	属性列
A	500	a1

表 5.6 无等值约束的单属性约束

约束编号	约束表达式	基数
1	A.a1>p1	123
2	A.a1<p2	321
3	A.a1>=p3	207
4	A.a1<=p4	441

表 5.7 包含等值约束的单属性约束

约束编号	约束表达式	基数
1	A.a1>p1	123
2	A.a1<p2	321
3	A.a1>=p3	207
4	A.a1<=p4	441
5	A.a1=p5	40
6	A.a1=p6	25

表 5.8 为单表多属性约束的表结构，表 5.9 和表 5.10 分别为无等值约束和包含部分等值约束的单表多属性输入基数约束。

表 5.8 单表多属性表结构

表名	表大小	属性列
A	500	a1, a2, a3, a4, a5

表 5.9 无等值约束的单表约束

约束编号	约束表达式	基数	依赖关系
1	A.a1>p1	123	
2	A.a2<p2	321	
3	A.a3>=p3	207	
4	A.a4<=p4	441	

续表 5.9 无等值约束的单表约束

约束编号	约束表达式	基数	依赖关系
5	A.a2+2*A.a3<=p5	150	5 依赖 2
6	5*A.a3-A.a5>p6	255	

表 5.10 包含等值约束的单表约束

约束编号	约束表达式	基数	依赖关系
1	A.a1>p1	123	
2	A.a2<p2	321	
3	A.a3>=p3	207	
4	A.a4<=p4	441	
5	A.a2+2*A.a3<=p5	150	5 依赖 2
6	5*A.a3-A.a5>p6	255	
7	A.a1=p7	40	
8	A.a2=p8	40	
9	A.a5=p9	80	
10	A.a4=p10	120	

表 5.11 为多表多属性约束的表结构，表 5.12 和表 5.13 分别为无等值约束和包含部分等值约束的多表输入基数约束。

表 5.11 多表结构

表名	表大小	属性列
A	1000	a1, a2, a3
B	2000	b1, b2, b3
C	10000	c1, c2, c3

表 5.12 无等值约束的多表约束

约束编号	约束表达式	基数	依赖关系
1	A.a3>=p1	200	
2	B.b2>p2	200	
3	A.a3-B.b3>p3	190	依赖 2 和 4
4	B.b1<p4	590	
5	C.c1>p5	1280	

续表 5.12 无等值约束的多表约束

约束编号	约束表达式	基数	依赖关系
6	B.b3+3*C.c3>p6	360	依赖 2，4 和 5
7	A.a3-2*B.b3<p7	90	
8	A.a1<=p8	1000	不参与

表 5.13 包含等值约束的多表约束

约束编号	约束表达式	基数	依赖关系
0	A.a2=p0	40	
1	C.c2=p1	130	
2	A.a1-B.b1=p2	270	
3	B.b2-C.c1=p3	240	依赖 1
4	A.a3>=p4	200	
5	B.b2>p5	200	
6	A.a3-B.b3>p6	190	依赖 5,7
7	B.b1<p7	590	
8	C.c1>p8	1280	
9	B.b3+3*C.c3>p9	360	依赖 5,7,8
10	A.a1=p10	420	
11	B.b1=p11	230	
12	A.a3-2*B.b3<p12	90	
13	A.a1<=p13	1000	不参与

5.2.4 DecAlgo 缺陷验证实验约束定义

通过该组实验验证算法章节对于等值冲突的论述验证；并且帮助说明前文所说的 DecAlgo 对于等值约束支持存在的局限性。表 5.14 为表结构，表 5.15 为约束。约束 1 和 2 将表的定义域划分为三个区间，其中区间最长为 198，以在最长区间内的等值约束 3 的基数为变量，展示三个约束整体误差率变化情况。

表 5.14 实验 3 表结构

表名	表大小	属性列
A	500	a1

表 5.15 实验 3 输入约束

约束编号	约束表达式	基数	备注
1	A.a1<p1	123	
2	A.a1>p2	179	
3	A.a1=p3	100	基数变化，且依赖于 1 和 2

5.3 实验结果及分析

5.3.1 传统负载上 DecAlgo 实验结果分析

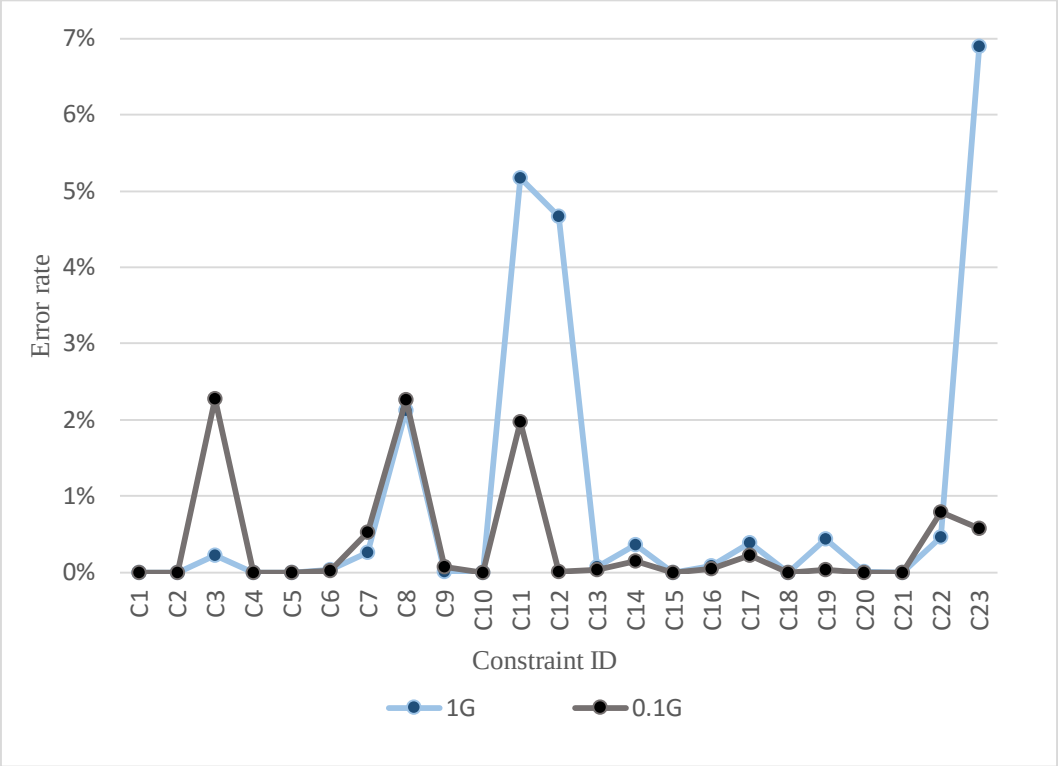


图 5.3 传统负载下 DecAlgo 误差率

图 5.3 的结果是 DecAlgo 依据 dbgen 生成的 1G 数据和 0.1G 数据，根据这些数据指定中间结果集大小为输入的误差率，可以看出绝大多数约束的误差率在 5%以下。图 5.4 的输入约束是改造后的 TPC-H 的 8 个 query，约束基数是以 dbgen 生成 0.01G 数据时指定的中间结果集大小，用改造后的 8 个 query 在 dbgen 生成的数据上执行，得到 dbgen 的查询响应时间，这 8 个 query 在 DecAlgo 生成的数据上执行，得到 DecAlgo 的查询响应时间，虽然 DecAlgo 查询响应时间不如 dbgen 生成的数据库好，但是查询时间的绝对值并不高，仍然可以接受。

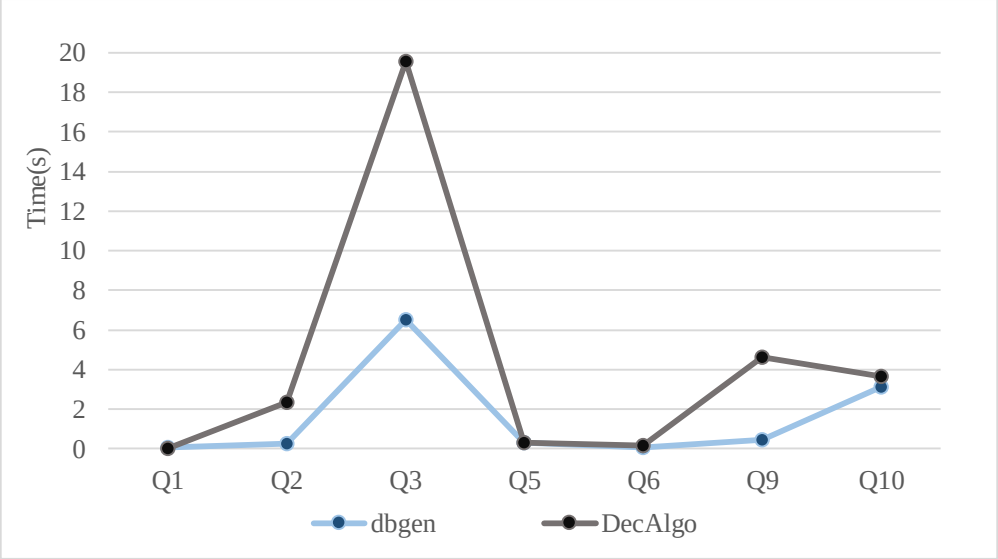


图 5.4 DecAlgo 与 dbgen 查询时间

5.3.2 DecAlgo 与 PGM 对比实验分析

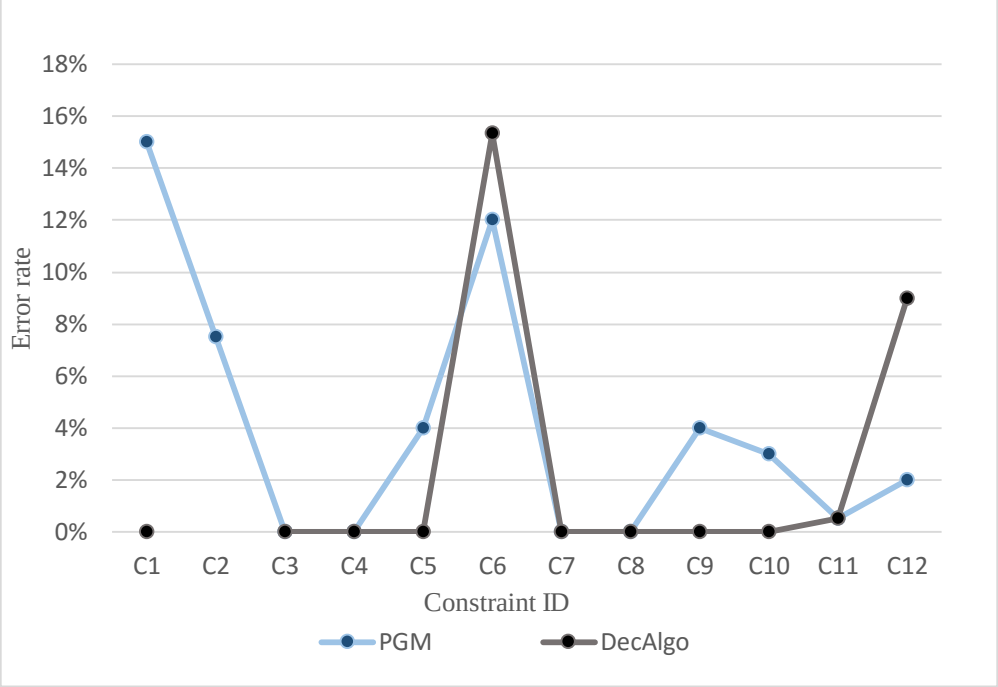


图 5.5 PGM 和 DecAlgo 误差率对比

图 5.5 是以表 5.3 和表 5.4 的负载约束作为输入生成数据的误差率结果，误差率=（约束基数理论值-约束基数实际值）/约束基数理论值。

从图 5.5 中可看出 DecAlgo 在多数约束上误差不高于 PGM，DecAlgo 在约束 2 缺失误差率是因为 DecAlgo 不支持单表多属性等值约束计算。

由于 PGM 基于概率图模型，因此较小的基数可能出现结果有点波动，为此本文在实验时对其进行多次实验，取其中较好的结果进行对比。

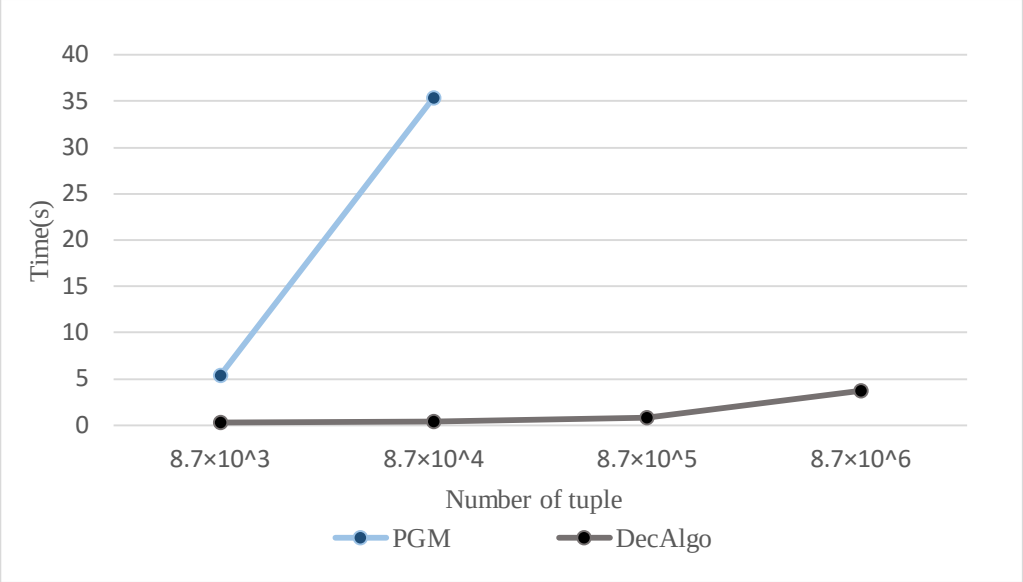


图 5.6 DecAlgo 和 PGM 生成时间对比

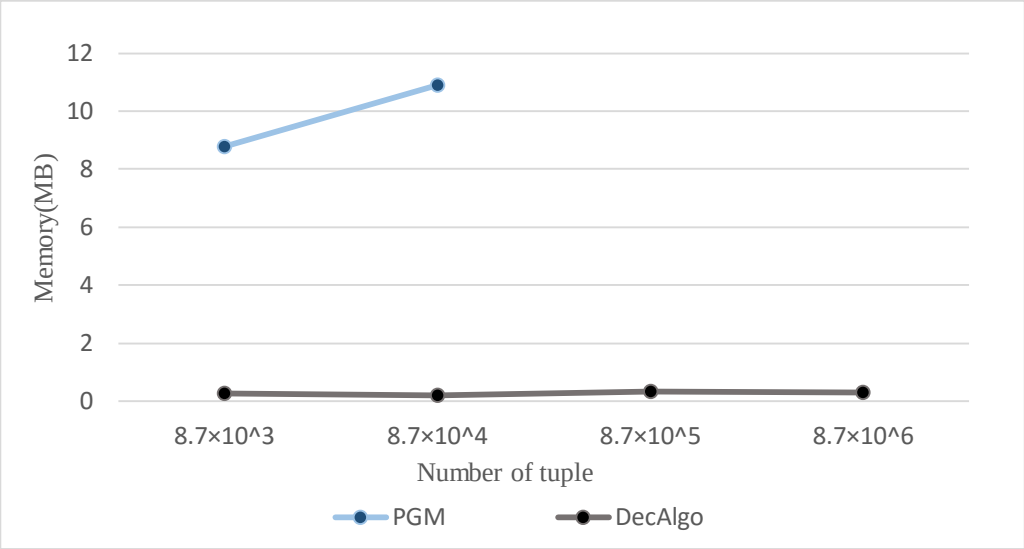


图 5.7 DecAlgo 和 PGM 存储消耗对比

图 5.6 和图 5.7 是在表 5.3 和表 5.4 基础上，对输入的表大小和基数同时扩大为原来 10 倍的实验结果。横坐标是指各表记录数之和，生成时间分别为 DecAlgo 修改完定义域后数据生成时间和 PGM 求得联合概率分布后的数据生成时间。图 5.6 中纵坐标数值为原始数据开平方后的数值，存储消耗是指程序运行过程中的最大内存消耗。

从图 5.6 中可以看出 PGM 在生成 8.7 万记录时，生成时间呈现一个爆炸性增长，从原来的十秒级别一跃到千秒级别，而 DecAlgo 则依旧维持在秒级别，PGM 在生成 200 万记录时所需的时间已经超过 11 小时，而 DecAlgo 时间花费依然在秒级别，在更大数据量时对 PGM 的实验由于时间耗费太大没有进行。

DecAlgo 在生成数据时可完全并发执行，因此在总记录数增大时，生成时间依然变化不大，当生成 870 万条记录时折线有所上扬是由于磁盘带宽瓶颈导致，但相比 PGM 时间消耗仍然很小，优势非常明显。PGM 尽管能够部分并发生成，但由于其在生成外键时需要对参照表中的记录进行匹配，当数据量增大时，时间耗费非常庞大，造成时间爆炸性增长。

图 5.7 中纵坐标为生成过程中的最大存储消耗，DecAlgo 由于可以直接落盘，因此存储消耗小。而 PGM 生成外键时需要存储之前的部分生成结果，因此存储消耗随着生成数据量的增大也增大。可以看出 DecAlgo 在传统查询负载条件下，能够高效生成低误差率的数据。

5.3.3 DecAlgo 验证实验分析

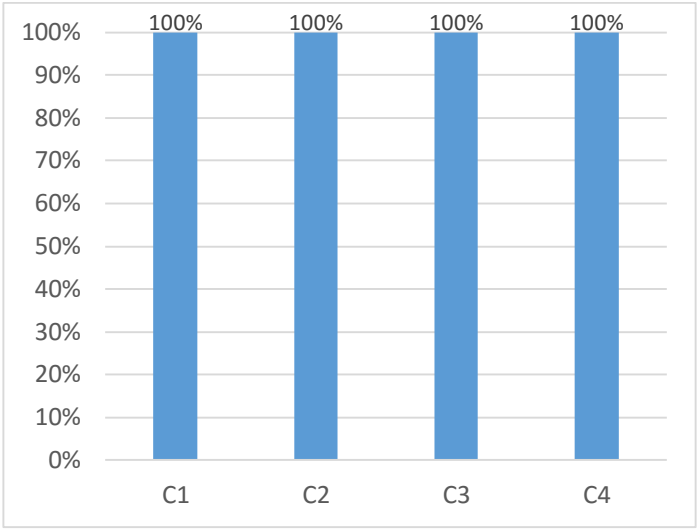


图 5.8 无等值约束的单表单属性准确率

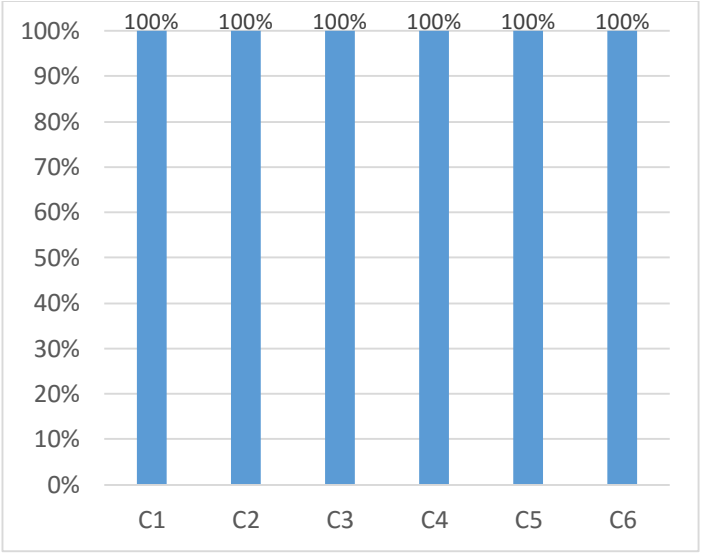


图 5.9 无等值约束的单表多属性准确率

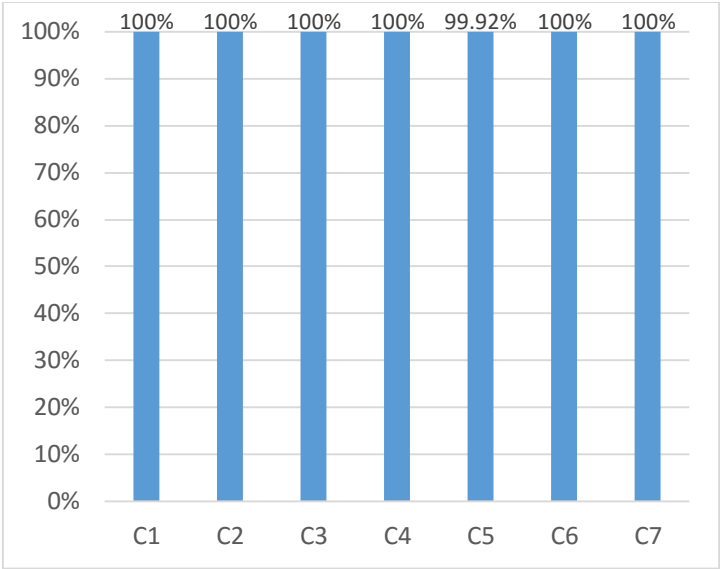


图 5.10 无等值约束的多表准确率

图 5.8，图 5.9 和图 5.10 是表 5.5，表 5.8，表 5.11，表 5.6，表 5.9 和表 5.12 作为输入的实验结果。准确率定义为生成基数与输入基数的比值。

可以看出当无等值约束时，单表约束和多表约束的准确率接近 100%，根据本文算法，我们在仅输入非等值约束条件下期望准确率最高能够达到 100%，因为此时不存在等值冲突，而等值段的加入是理论上唯一能够带来误差的因素。

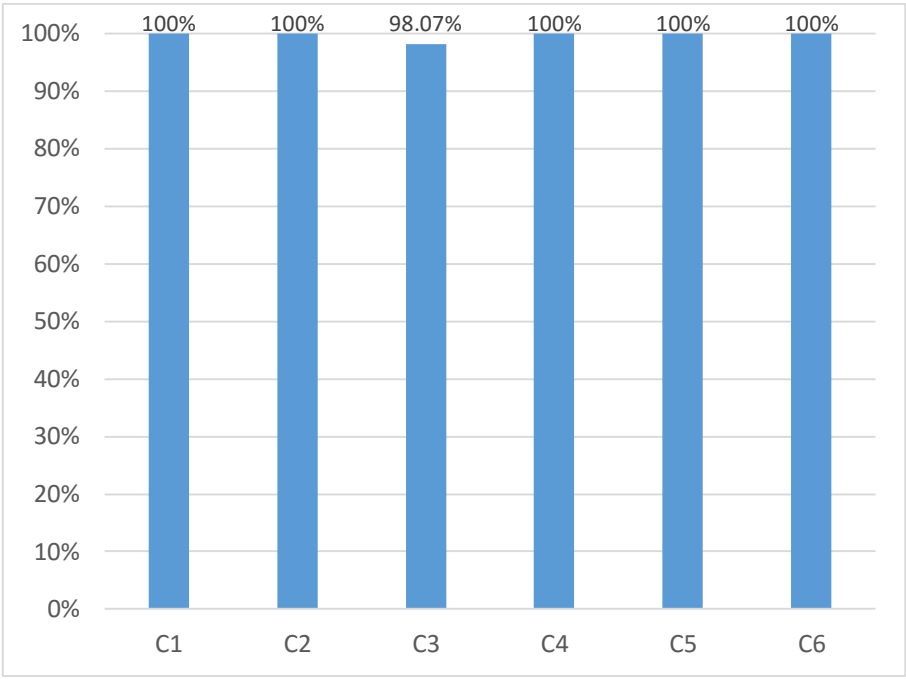


图 5.11 包含等值约束的单表单属性准确率

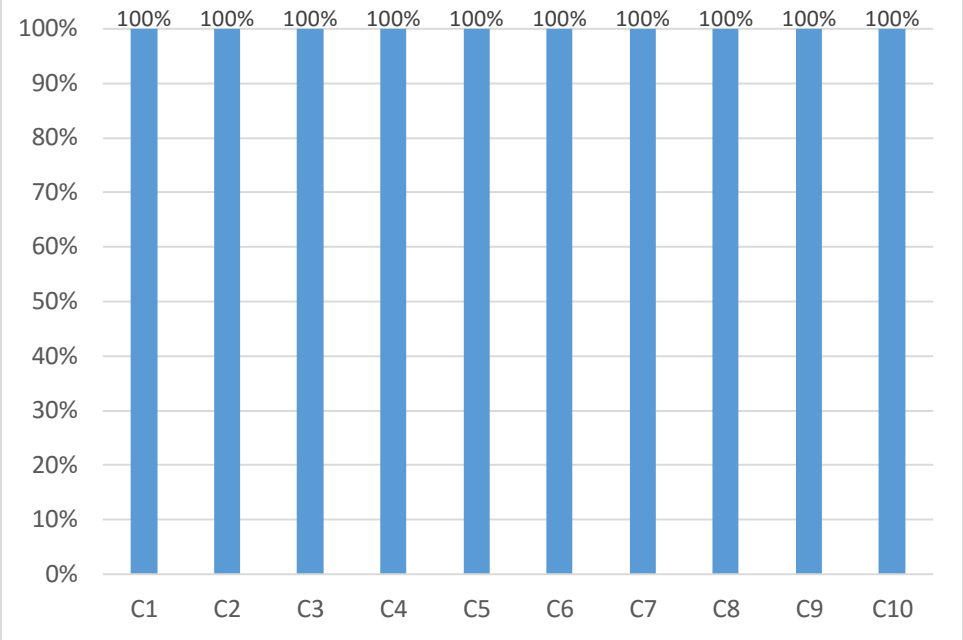


图 5.12 包含等值约束的单表多属性准确率

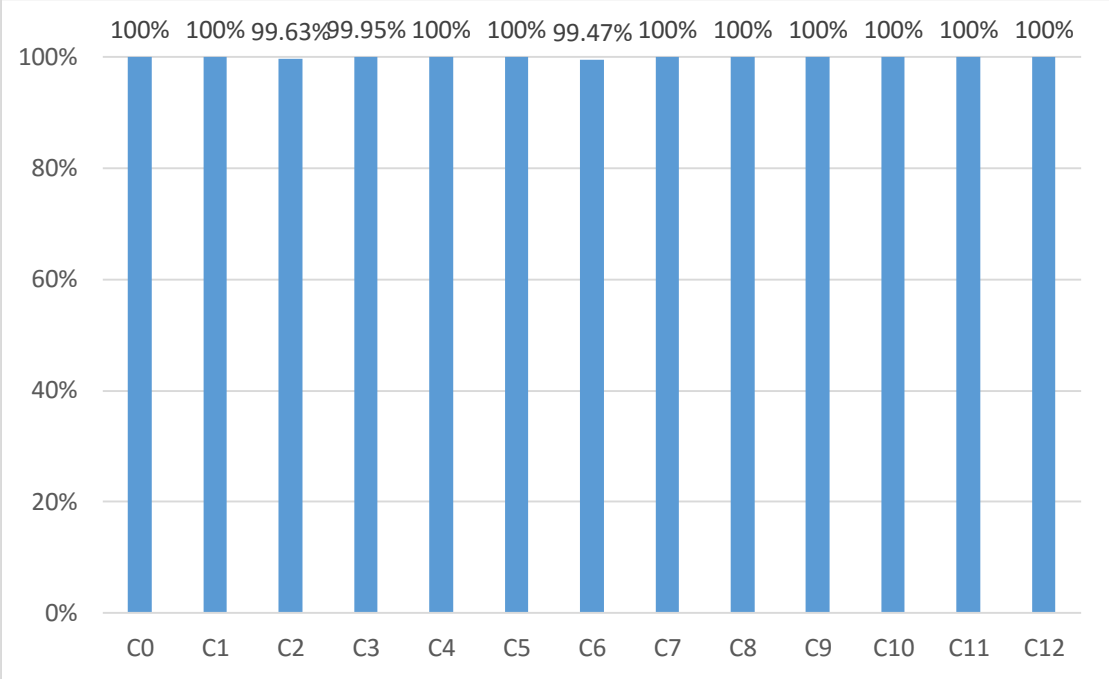
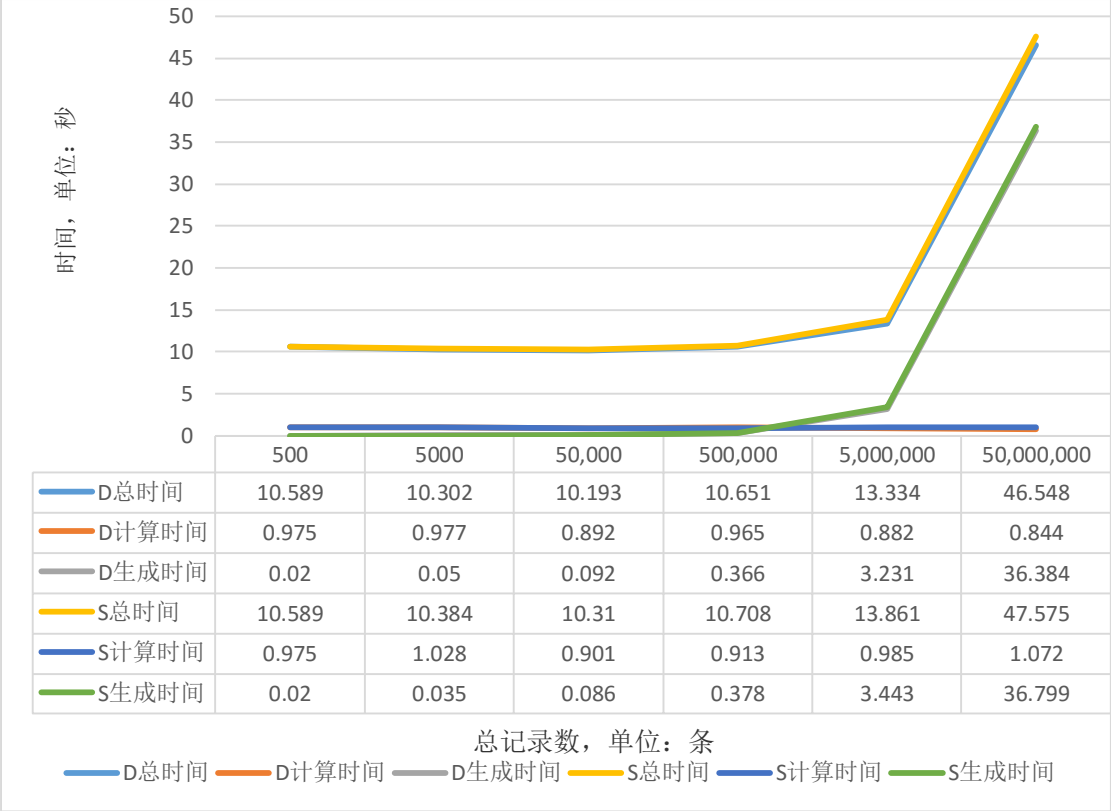


图 5.13 包含等值约束的多表准确率

图 5.11，图 5.12 和图 5.13 是表 5.5，表 5.8，表 5.11，表 5.7，表 5.10 和表 5.13 作为输入的实验结果。

可以看出尽管存在等值约束，但 DecAlgo 在单表和多表约束上的准确率依然非常高，在绝大多数约束上准确率可以达到 100%，表明本文算法能够准确、高效地解决非等值约束和部分等值约束。



	500	5000	50,000	500,000	5,000,000	50,000,000
D总时间	10.589	10.302	10.193	10.651	13.334	46.548
D计算时间	0.975	0.977	0.892	0.965	0.882	0.844
D生成时间	0.02	0.05	0.092	0.366	3.231	36.384
S总时间	10.589	10.384	10.31	10.708	13.861	47.575
S计算时间	0.975	1.028	0.901	0.913	0.985	1.072
S生成时间	0.02	0.035	0.086	0.378	3.443	36.799

图 5.14 DecAlgo 两种情况时间对比

图 5.14 在图 5.12 的表结构和基数约束输入基础上，对比 DecAlgo 在表大小和约束基数同等扩大为原来的 10 倍（以 D 开头标识）和只有表大小扩大为原来的 10 倍（以 S 开头标识）两种情况的各项时间结果。从图中可以看出两种情况的各项时间的折线基本重合，表明两种情况的时间耗费相差不大，可以认为是一致的，据此我们认为之前通过将表大小和约束基数同等扩大 10 倍以求得生成时间的做法合理。

5.3.4 DecAlgo 简单局限实验分析

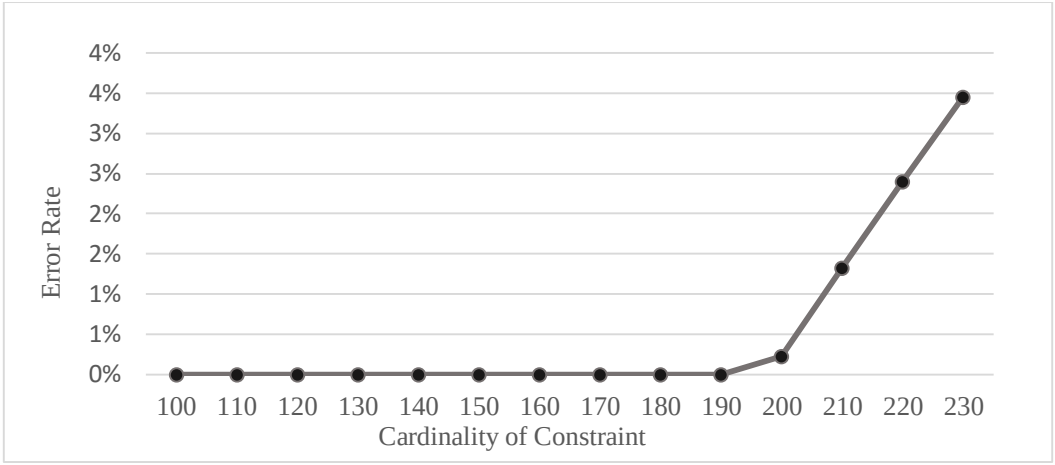


图 5.15 实验 4 误差率

图 5.15 是表 5.14 和表 5.15 作为输入的实验结果，其中整体误差率定义为：所有约束基数的误差之和/所有约束的理论基数之和。

图 5.15 中折线前期误差为 0，因为此时等值约束的基数还没到达最大区间长度时，DecAlgo 容易找到一个合适点插入等值段。

当等值约束的基数超过 198 后，误差率开始上升，因为此时定义域范围边界点会落在等值段上，无法找到一个仍能满足非等值约束和等值约束的插入点，导致非等值约束精度受损。尽管如此，此时整体误差率仍然较低，且误差上升趋势呈线性，不至于因为超过一点就导致误差率剧烈上升，对于数据生成问题，即使是 20%的误差率仍然可以容忍。

5.4 本章小结

本章首先介绍了实验环境，接着是四部分实验的输入表结构和基数约束，最后是实验结果和相应分析。

通过在传统查询负载下，DecAlgo 的误差率和查询响应时间实验，DecAlgo 与 PGM 在部分传统查询负载上的误差率、生成性能和存储消耗对比实验，表明本文工作的通用性和高效性。

通过在已有非等值约束基础上包含部分等值约束和无等值约束两种情况的实验，验证 DecAlgo 生成数据的准确性和高效率。

最后通过实验也说明 DecAlgo 数据生成器对于等值约束的支持程度。

第 6 章 总结与展望

6.1 总结

随着社会和科技的不断发展进步，数据库从最初的为金融行业记账服务，到现在应用于各行各业。为了灵活测度数据库性能、可用性、鲁棒性、可扩展性，良好的数据生成器是不可少的工具之一，因此研究数据生成很有意义。

面向非等值连接负载约束的数据生成问题是数据生成领域的一大难题，目前尚未有针对此问题的数据生成算法，对此本文提出一种基于数据生成与属性间关联关系解耦思想的算法 DecAlgo。本文首先为每个属性定义生成函数，然后根据负载特征约束之间的依赖关系和影响关系对属性生成函数的定义域进行调整，最后再依据属性的生成函数和定义域生成满足基数约束的查询实例。本文数据生成实现了完全并发，生成时不需要依赖于已生成的数据和属性之间的关联，能够高效地利用计算和存储资源。通过针对常用的传统查询负载的实验，与基于概率图模型(probabilistic graphical models, PGM)的数据生成器[3]查询负载的对比实验，针对非等值负载的实验，证明本文工作的通用性、有效性和高效性。针对非等值连接负载 DecAlgo 的误差率在 2%以内；针对传统负载 DecAlgo 的误差率平均在 5%内，并且 DecAlgo 的数据生成性能仅受磁盘带宽影响。

6.2 展望

本文存在着一些需要改进的地方，主要体现在以下两点：

（1）在规划等值段位置时，本文通过在范围内随机指定插入等值点的方法实现，但是当此范围内等值段较多时，随机的方法可能导致可插入部分碎片化严重，可以采用插入时尽量从小的位置开始，在应对范围内等值段较多时效果会较好。

（2）由于单表多属性等值约束插入等值段时容易与单表单属性和多表多属性等值约束的等值段产生冲突，目前本文还不支持此类冲突。后续工作中将尝试解决。

参考文献

1. C. Binnig, D. Kossmann, E. Lo, et al. QAGen: Generating Query-Aware Test Databases[A]. SIGMOD[C], Beijing, 2007, pages 341-352.
2. E. Lo, N. Cheng, and W.-K. Hon. Generating Databases for Query Workloads[A]. VLDB[C], Singapore, 2010, pages 848-859.
3. A. Arasu, R. Kaushik, and J. Li. Data Generation using Declarative Constraints[A]. SIGMOD[C], Athens, 2011, pages 685-696.
4. J. Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference[M]. San Francisco: Morgan Kaufmann, 1988.
5. C. Re and D. Suciu. Understanding cardinality estimation using entropy maximization[A]. PODS[C], Indianapolis, 2010, pages 53-64.
6. U. Srivastava, P. J. Haas, V. Markl, et al. ISOMER: consistent histogram construction using query feedback[A]. ICDE[C], Atlanta, 2006.
7. N. Bruno and S. Chaudhuri. Flexible Database Generators[A]. VLDB[C], Trondheim, 2005, pages 1097-1107.
8. K. Houkjaer, K. Torp, and R. Wind. Simple and Realistic Data Generation[A]. VLDB[C], Seoul, 2006, pages 1243-1246.
9. Joseph E. Hoag, and Craig W. Thompson. A Parallel General-Purpose Synthetic Data Generator[A]. SIGMOD[C], Beijing, 2007, pages 19-24.
10. Alexander Alexandrov, Kostas Tzoumas, and Volker Markl. Myriad: Scalable and Expressive Data Generation[A]. PVLDB[C], Istanbul, 2012, pages 1890-1894.
11. C. Olston, S. Chopra, and U. Srivastava. Generating Example Data for Dataflow Programs[A]. SIGMOD[C], Providence, 2009, pages 245-256.
12. Wentian Lu, Gerome Miklau, and Vani Gupta. Generating Private Synthetic Databases for Untrusted System Evaluation[A]. ICDE[C], Chicago, 2014, pages 652-663.
13. S. Cohen. Generating XML Structure using Examples and Constraints[J]. PVLDB, 2008, 1(1): 490-501.
14. J. Leskovec, D. Chakrabarti, J. M. Kleinberg, et al. Realistic, Mathematically Tractable Graph Generation and Evolution, using Kronecker Multiplication[A]. PKDD[C], Portugal, 2005, pages 133-145.
15. S. Lattanzi and D. Sivakumar. Affiliation Networks[A]. STOC[C], Bethesda,

2009, pages 427-434.

16. C. Binnig, D. Kossmann, and E. Lo. Reverse Query Processing[A]. ICDE[C], Istanbul, 2007, pages 506-515.
17. Quoc Trung Tran, Chee Yong Chan, and Srinivasan Parthasarathy. Query Reverse Engineering[J]. VLDBJ, 2014, 23(5):721-746.
18. Chaitanya Mishra, Nick Koudas, and Calisto Zuzarte. Generating Targeted Queries for Database Testing[A], SIGMOD[C], Vancouver, 2008, pages 499-510.
19. N. Bruno, S. Chaudhuri, and D. Thomas. Generating Queries with Cardinality Constraints for DBMS Testing[J]. IEEE Trans. Knowl. Data Eng., 2006, 18(12):1271-1275.

致谢

在此毕业设计（论文）完成之际，谨向我的指导教师赵海教授和华东师范大学的张蓉副教授致以衷心的感谢。感谢赵海老师和张蓉老师在我毕业设计期间给予的悉心指导、热诚帮助和大力支持。导师严谨的治学态度和渊博的学识使我受益匪浅。

对我的学长李宇明和大学同学们表示特别感谢，李宇明学长在我毕设期间给予了我耐心的解答指导，在四年的大学学习生活中，同学们在很多方面给予我很大的帮助和宝贵的支持。另外，对毕业设计期间在同一实验室共同学习、工作同学深表感谢，感谢大家与我一起度过三个月快乐的毕业设计生活。

最后，我谨向所有关心和帮助过我的老师、同学和家人致以真诚的谢意。