Софийски университет
„Св. Климент Охридски”

Факултет по математика
и информатика

Стоян Милчев Велев

Оптимизация на SQL заявки чрез генетичен алгоритъм

АВТОРЕФЕРАТ

на

Дисертация

за присъждане на образователната и научна степен Доктор
по научната специалност 01.01.12 Информатика

Научен ръководител: доц. д-р Владимир Димитров

Научно жури:
проф. д.т.н. Цветан Семерджиев (ИИКТ БАН)
проф. д-р. Аврам Ескенази (ИМИ БАН)
доц. д-р. Тодор Гюров (ИИКТ БАН)
доц. д-р. Мария Нишева (ФМИ СУ)
доц. д-р. Владимир Димитров (ФМИ СУ)

София, ноември 2011 г.

Дисертационният труд е обсъден и насочен за публична защита на 14.11.2011 г. на заседание на катедра "Компютърна Информатика" към Факултета по математика и информатика (ФМИ) на Софийски университет (СУ) "Св. Климент Охридски" .

Публичната защита на дисертационния труд ще се състои на г. от часа в на открито заседание. Материалите по защитата са на разположение в библиотеката на ФМИ към СУ (София, бул. Джеймс Баучър №5).

Пълният обем на дисертацията е 173 страници. Дисертацията се състои от увод, 6 глави, заключение, авторска справка, списък с цитираната литература и 3 приложения. Цитираната литература включва 87 заглавия (всички на английски език). Списъкът от публикации на докторанта по същината на дисертацията включва 4 заглавия.

СЪДЪРЖАНИЕ

ОБЩА ХАРАКТЕРИСТИКА НА ДИСЕРТАЦИОННИЯ ТРУД

Актуалност на проблема и мотивация

Ролята на системите за управление на бази данни (СУБД) в софтуерната индустрия трудно може да бъде преувеличена. Възможността за надеждно съхранение на данни и бързото им извличане е самата основа на информационните технологии – трудно е да си представим софтуерно приложение, система или например Интернет без тези възможности.

Проблемът за оптимизацията на SQL заявки е централен в областта на базите данни и е обект на множество съвременни изследвания. Скоростта на обръщенията към БД обикновено са определящи за скоростта на приложението като цяло, а за тази скорост генерирането на ефективни планове за изпълнение на заявките е от ключово значение. Оптимизацията на заявки към БД е един от най-важните оптимизационни проблеми в компютърната наука.

Релационните СУБД, освен бързото си налагане и доминиране в последните десетилетия, и днес продължават да са най-широко приложимите, а благодарение на свойствата си са и единствените приложими в определени области. Това вероятно няма да се промени и в близко бъдеще, въпреки навлизането на нови типове СУБД като нерелационните и in-memory.

Едно от основните предимства на релационните СУБД е гъвкавостта при извличане на данните чрез заявки на мощен декларативен език. Изпълнението на заявките в реално време, независимо от обемите данни, е от ключово значение за приложенията, затова и оптимизацията на заявките продължава да е област на активни изследвания. Ролята на оптимизаторите на заявки става особено критична в последните години заради високото ниво на сложност характеризиращо съвременните приложения.

Оптимизирането на съединенията винаги е било на фокус заради цената на операцията, и докато при класическите приложения на релационни бази заявките рядко съединяват повече от 8-10 релации, в много от съвременните приложения на БД възниква нуждата от ефективно изчисление на порядъци повече съединения (десетки и стотици) – в системите за вземане на решения (Decision Support Systems, DSS) и Аналитична онлайн обработка (On-line Analytical Processing, OLAP), Data Warehousing (DW), мултимедийните БД, уеб-базираните БД, и Географски информационни системи (GIS), в паралелни и разпределени системи, в дедуктивните БД, в логическото програмиране, в задачите за удовлетворяване на ограничения, обектноориентираните БД (ООБД) и други. Класическите методи, алгоритми и системи за оптимизация на заявки са неприложими или неефективни за заявки с голям брой съединения (над 10-12), каквито често възникват в такъв тип приложения.

За решаване на проблема с оптимизацията на заявки с голям брой съединения съществуват редица *частни* решения с полиномиална сложност. Решенията с полиномиална сложност обаче налагат твърде съществени ограничения върху вида на заявките, типа на използваната функция за оценка на сложността, конкретния вид на методите за реализация на съединенията и т.н., т.е. те са неприложими в общия случай. Общата задача за намиране на оптималния ред за изпълнение на последователност от съединения за съжаление е $\mathcal{NP}$-пълна. Формално доказателство на този факт за циклични заявки е представено за първи път в [9]. За малки заявки е възможно да се извърши пълно обхождане и да се намери глобалният оптимум, но, тъй като размерът на пространството на решенията нараства експоненциално по броя на

съединените релации, за по-големи заявки задачата за подредба на съединенията не може да бъде решена точно в общия си вид.

Цел и задачи на дисертацията

Целта на настоящия труд е обзор и анализ на областта на оптимизирането на заявки към релационни СУБД, проектиране на генетичен алгоритъм за оптимизиране на заявки с голям брой съединения, създаване на оптимизатор на SQL заявки използващ разработения алгоритъм, както и получаване на експериментални резултати от представянето на алгоритъма.

От така поставената цел произтичат следните задачи:

- Да се направи обзор на принципите и методите за обработка на SQL заявки в СУБД – от текст на заявка до план за нейното изпълнение; принципната архитектура на модулите на СУБД, натоварени с обработката на заявките; стандартните фази, през които преминава обработката

- Да се направи обзор и класификация на съществуващите алгоритми за подреждане на съединения, да се анализират техните свойства, ограничения и приложимост

- Да се проектира генетичен алгоритъм за оптимизиране на заявки с голям брой съединения преодоляващ част от известните недостатъци на съществуващите класически алгоритми

- Да се създаде библиотека за реализиране на генетични оптимизатори

- Да се проектира и реализира експериментален оптимизатор способен да обработва много големи заявки (със стотици съединения) и да работи в реално време

- Да се създаде система за автоматизиране на експерименти с генетични алгоритми приложени към задачата за оптимизиране на заявки

- Да се получат експериментални резултати от представянето на реализирания оптимизационен алгоритъм (в това число да се подберат параметри, които максимизират качеството на решенията за типични заявки) и да се проведе сравнително изследване с аналогични алгоритми описани в литературата

Обем и структура на дисертацията

Обемът на дисертацията е 173 страници, като тя се състои от 8 глави, списък на цитираната литература и 3 приложения. Използваната литература включва 87 заглавия, всички на английски език. Списъкът с публикации съдържа 4 заглавия, всичките с единствен автор и на английски език. Посочено е едно известно цитиране.

Глава 1 въвежда в проблемната област, дефинира задачата за намиране на оптималната подредба на съединенията и разглежда нейната сложност. Определят се целта и задачите на дисертационния труд.

Глава 2 ни запознава с основите на оптимизирането на заявки към БД: разглежда се принципната архитектура на един ценови оптимизатор на заявки и се изяснява функционалността и взаимодействието на основните му модули. Представят се методите за достъпване на релации и реализация на съединения, профилни формули за основните операции на релационната алгебра, ценови модели на заявки към БД, както и основните техники за определяне на размера на релации и разпределение на стойностите на атрибутите. Дефинира се пространството на решенията и неговото най-важно подпространство – това на ляворекурсивните решения и се изследват техните свойства.

Глава 3 представлява обзор и класификация на известните стратегии и алгоритми за подреждане на съединения. Разглеждат се детерминистични алгоритми, недетерминистични – рандомизирани и генетични, както и хибриди алгоритми – такива притежаващи характеристики от повече от един от тези класове. Главата завършва със сравнителен анализ на разгледаните алгоритми: тяхната приложимост, свойства и ограничения. Основните резултати са представени в [4A].

Глава 4 прави кратък исторически преглед на системите за оптимизиране на заявки като проследява развитието им от самото начало в края на 70-те години на миналия век със System R на IBM до днешните разработки, като очертава и посоките на развитие на бъдещите поколения системи за оптимизиране на заявки към БД.

Глава 5 представя оригинален адаптивен генетичен алгоритъм за оптимизиране на заявки. Разглежда се избраното представяне на решенията и реализацията на операторите за мутация, кръстосване и селекция. Особено внимание се отделя на мотивацията и реализацията на предложения алгоритъм за селекция с адаптивен, динамично изменящ се размер на популацията. Главата завършва с доказателство за сходимост на предложения алгоритъм. Основните резултати са представени в [1A] и в [2A].

Глава 6 описва реализацията на генетичен оптимизатор на заявки. Разглежда се архитектурата на системата, конкретизацията на реализацията на отделните модули и входно-изходния интерфейс на системата. Дефинират се ограниченията на системата – поддържани методи за достъп до релация, поддържани методи за съединение, ценови модел и поддържаното подмножество на езика SQL. Специфицира се и форматът на резултата, т.е. синтаксисът на плана за изпълнение на заявката. Основните резултати са представени в [3A].

Глава 7 представя експерименталните резултати от представянето на предложения оригинален алгоритъм спрямо няколко класически недетерминистични оптимизационни алгоритми. Описани са използваните тестови модули, включително инструментите за генериране на случайни системни каталози и заявки над тях, които се използват като входни данни за оптимизатора. Част от резултатите са представени в [2A].

Глава 8 обобщава изводите от предходните глави и разглежда научните, научно-приложните и приложните приноси на настоящия труд. Представен е списък от публикации на основните ни резултати и тяхното представяне на научни форуми, както и известните ни към момента цитирания от други автори. Главата завършва с някои насоки за бъдещо развитие на работата.

Основната част на дисертацията завършва с пълен списък на цитираните литературни източници, след което са поместени няколко приложения, които допълват изложението: детайли по реализацията на библиотеката за изграждане на генетични оптимизатори, пълния изходен код на библиотеката, примерен системен каталог и примерен ценови модел.

КРАТКО СЪДЪРЖАНИЕ НА ДИСЕРТАЦИОННИЯ ТРУД

Глава 1: Увод

В настоящата работа ще се концентрираме върху оптимизирането на един конкретен вид заявки: единични *плоски* SQL заявки, с предикати, в които единствен съюз ще бъде конюнкцията (подобни заявки се наричат още *конюнктивни заявки*, заявки тип *селекция-проекция-съединение*, нерекурсивни клаузи на Хорн) в централизирана релационна СУБД, като предполагаме, че имаме пълна информация за run-time средата още по време на компилация.

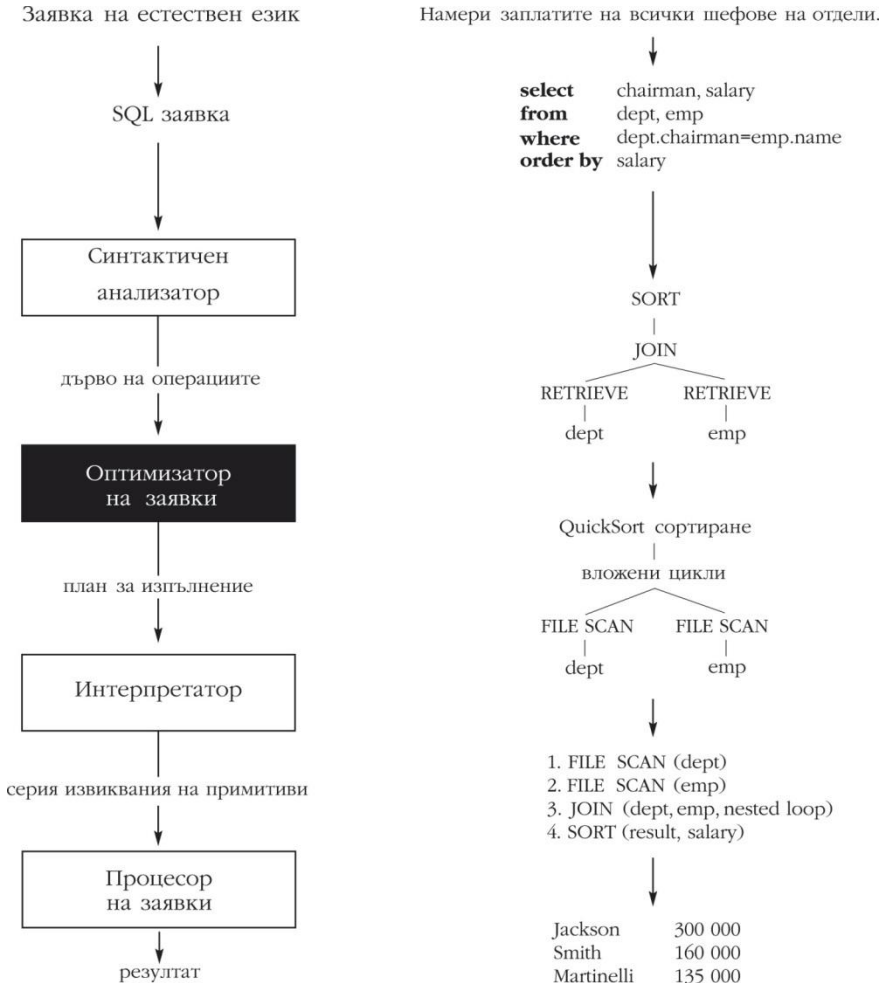
Всяка заявка Q към релационна база от данни дефинирана от някакъв речник на данните $\mathcal{D}$ с множество от релации $\mathcal{R}$ се представя от наредената двойка (R^q, P^q), където $R^q = \{R_i \mid R_i$ участва в $Q\}$, $R^q \subseteq \mathcal{R}$ и $P^q = \{p_i(R^i_j, R^i_k) \mid p_i$ е предикат на съединение в Q, $R^i_j, R^i_k \in R^q\}$. *Планът за изпълнение на заявка* (ПЕЗ) на Q е двоично дърво, вътрешните възли на което представляват реализации на оператора за съединение (*методи за съединение*), например съединение чрез вложени цикли, съединение чрез сливане или съединение чрез хеширане, а листата представляват базови релации. За разлика от самата заявка (която има само декларативна семантика), планът за изпълнение на заявката съдържа процедурната информация как да бъде получен резултатът от заявката. Всеки план за изпълнение има *цена*, която отразява изчислителните ресурси необходими за изпълнението му.

Задачата е, по дадена заявка Q да се намери планът за изпълнение (от множеството на всички еквивалентни) с най-ниска цена, който я оценява (глобалният оптимум). Тъй като комбинаторната експлозия прави пълното обхождане на пространството от решения невъзможно, целта се ограничава до намирането на добър *локален* оптимум.

За заявка Q върху n релации към база данни поддържаща множество $\mathcal{S}$ от различни методи за съединение, съществуват $\frac{1}{n}\binom{2(n-1)}{n-1}$ възможни дървовидни структури на ПЕЗ, за всяко дърво, неговите n листа могат да бъдат подредени по $n!$ различни начина и всеки вътрешен възел може да бъде избран измежду s различни метода за съединение, където $s = |\mathcal{S}|$. Дори да ограничим разглежданията си до множеството на ляворекурсивните решения (съдържащи всички дървета, за които е в сила, че десният наследник на всеки възел е базова релация), все още има $n!s^{n-1}$ различни решения.

Глава 2: Принципи на оптимизирането на SQL заявки

Архитектурата на обработката на заявките се състои от четири основни модула: *синтактичен анализатор*, *оптимизатор на заявките*, *интерпретатор* и *процесор на заявките*.



- *Синтактичният анализатор* проверява синтактичната коректност на заявката и след това я превежда в някакво вътрешно представяне, обикновено израз на релационното смятане или някакъв еквивалентен вид [2].

- *Оптимизаторът на заявки* разглежда всички алгебрични изрази, които са еквивалентни на получения от синтактичния анализатор и избира онзи от тях, чиято цена е най-малка.

- *Интерпретаторът* трансформира плана за изпълнение, генериран от оптимизатора в серия извиквания към процесора на заявки.

- *Процесорът на заявки* изпълнява серията операции и генерира резултата – отговор на заявката.

Принципна архитектура на оптимизатора на заявки

Оптимизаторът е най-сложният от модулите, натоварени с обработката на заявките в СУБД. Той е отговорен за трансформирането на декларативната заявка (в някакво вътрешно представяне) в ефективен процедурен план за нейното изпълнение.

Първо заявката се анализира, за да се открият евентуални синтактични или семантични грешки. През тази фаза оптимизаторът разглежда системния каталог, за да може да осъществява и семантична проверка за коректност. Ако заявката е коректна, тя се превежда във вътрешна, алгебрична форма. Оттук започва същинската оптимизация.

- *Алгебрична оптимизация*. Изпълняват се всички алгебрични трансформации, които винаги водят до опростяване на заявката – например преместване на селекциите и проекциите по дървото колкото се може по-близо до листата. Тази логическа оптимизация не зависи от ценовия модел.

- *Ценова оптимизация*. Тя зависи от наличните физически методи за достъп до данните и реализиране на логическите операции, а също и от ценовия модел. Макар общите принципи за оптимизация да са добре известни, това е нивото, на което всички оптимизатори се различават [28].

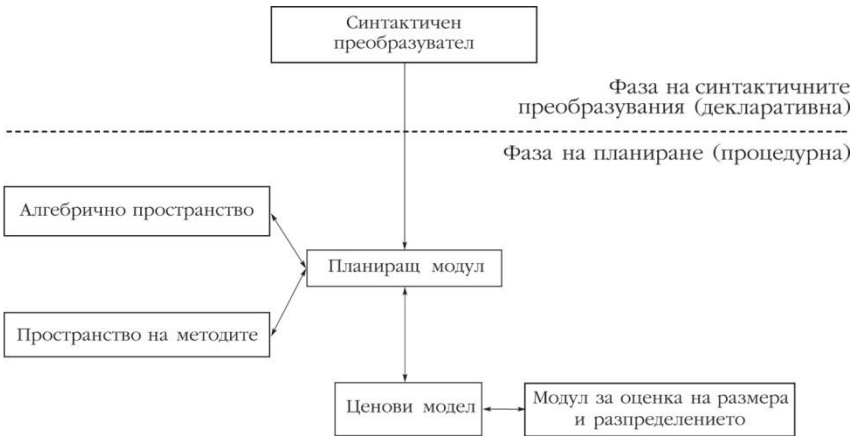
Резултатът от оптимизацията е план за изпълнение на заявката и външни зависимости на плана - това са параметрите, при които планът за изпълнение е приложим и актуален. Тези параметри отразяват характеристики на *run-time* средата и на текущото съдържание на БД, при които е правена оптимизацията.

Всяка СУБД поддържа количествена информация за релациите в БД – т.нар. *профили* на релациите [1], които се съхраняват в системния каталог (речника на данните). Системният каталог съдържа данни и статистики за атрибутите на релацията, мощността на релацията, правата за достъп до релацията и т.н. В системния каталог се съхранява още информация за наличните индекси, тригери, съхранени процедури, пакети и изгледи.

Ценовият модел изисква хипотези за мощностите на междинните резултати от алгебричните оператори, получени по статистически методи.

Абстракция за оптимизационния процес в СУБД

Следната фигура представя абстрактна модулна архитектура на оптимизатор на заявки [10]. Реалните СУБД се основават на същите принципи, макар модулите им не винаги да имат толкова ясно очертани граници.



Синтактичният преобразувател прилага трансформации към дадена заявка и произвежда алтернативни заявки, които са еквивалентни на дадената, но евентуално се характеризират с по-ниска цена за изпълнение. Такива трансформации включват замяна на изгледи (*views*) с техните дефиниции, привеждане на вложените заявки в плосък вид и т.н. Трансформациите, прилагани от синтактичния преобразувател, зависят само от декларативните, т.е. статичните характеристики на заявките и не вземат предвид реалната цена за изпълнението на заявката при специфичната СУБД и БД. Поради характера на трансформациите, прилагани от синтактичния преобразувател, ще казваме, че той работи на *декларативно* ниво [10].

Планиращият модул разглежда всички възможни планове за изпълнение, получени на предната фаза и избира най-добрия от тях, който да се използва за получаване на резултата на оригиналната заявка. Използва *стратегия на търсене*, която изследва пространството от планове за изпълнение по определен начин. Това пространство се дефинира от два други модула на оптимизатора, *алгебричното пространство* и *пространството на методите*. Тези два модула, заедно със стратегията за търсене, определят цената за изпълнение на дадена заявка. Цената се получава от последните два модула на оптимизатора, *ценовия модел* и *модула за оценка на размера и разпределението*.

Алгебричното пространство определя реда за изпълнение на действията, които трябва да се разгледат от планиращия модул. Поради алгоритмичната природа на обектите, генерирани от този модул и препращани на Планиращия модул, за цялата фаза на планиране се каза, че работи на *процедурно ниво*.

Пространството на методите определя избора за реализация от възможните алтернативи, които съществуват за изпълнението на всяка редица от действия, определена от алгебричното пространство. Този избор е свързан с наличността на индекси за базовите релации, достъпните реализации за съединение на релации (напр. вложени цикли, сортирано сливане, хеш-цикли и т.н.), дали спомагателните структури от данни да се генерират динамично, дали и кога да се премахват дубликатите и други подобни характеристики, които са предопределени от реализацията на конкретната СУБД. По дадена алгебрична формула или дървовидно представяне от алгебричното пространство, този модул генерира всички съответни пълни планове за изпълнение, които определят реализацията на всеки оператор и използването на всички индекси.

Ценови модел се нарича математическото приближение на реалното поведение на плана за изпълнение. Този модул определя аритметичните формули, които се използват за определяне на цената на плана за изпълнение. За всеки различен метод за съединение, за всеки различен тип индекс и изобщо за всяка отделна стъпка, която може да фигурира в един план за изпълнение, има формула, която дава цената й. Поради твърде сложното точно пресмятане на сложността на много стъпки, повечето от тези формули са прости приближения на реалната цена и се базират на определени предположения за системата [21]. Най-важните входни данни за една формула са размера на буфера, използван от съответната стъпка, мощностите на участващите релации и различни разпределения на стойностите в тези релации. Първата от тях се получава от СУБД, а останалите две – от модула за оценка на размера и разпределението.

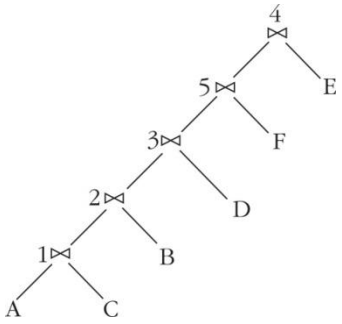
Модулът за оценка на размера и разпределението определя как се оценяват мощностите и честотните разпределения на стойностите на атрибутите на релациите в базата и индексите. Както бе споменато по-горе, тези оценки са нужни на ценовия модел. Конкретният подход за оценка, приет от този модул, определя и това какви статистики трябва да се поддържат в каталозите на всяка БД. Всъщност, за да се оценяват размерите на резултата от заявка, която включва множество атрибути от една и съща релация, са нужни множествените съвместни разпределения на всички тези атрибути. На практика обаче СУБД изследват само честотните разпределения на единични атрибути, защото разглеждането на всички възможни комбинации от атрибути е много скъпо. Този подход се основава на принципа, известен като *предположение за независимостта на стойностите на атрибутите*, който, макар и рядко верен, е възприет от всички съвременни СУБД. Предложени са различни техники за оценка на мощностите на резултатите от заявките и честотните разпределения на атрибутите. Повечето комерсиални СУБД основават своите оценки на *хистограми* [14]. Друг подход е статистическият анализ на случайни извадки от данните по време на изпълнение на заявката [19].

Пространство на решенията

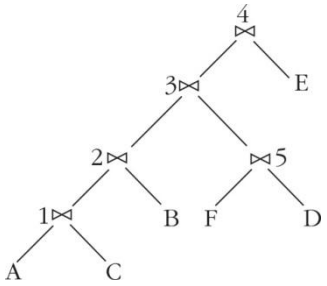
Пространството на решенията се дефинира като множеството от всички решения, т.е. всички дървета на обработка, които изчисляват резултата от съединението и които съдържат всяка базова релация точно веднъж. Листата на дърветата са базови релации, а вътрешните им възли съответстват на резултата от съединението на двата наследника на възела.

Множеството от всички дървета на обработка, чиито вътрешни релации на всяко съединение са базови релации наричаме *ляворекурсивни дървета*. При n базови релации размерът на това пространство е $n!$. Ще означаваме това множество от решения с $\mathcal{L}$.

Множеството от всички дървета на обработка наричаме *храстовидни дървета*. В резултат, пространството на ляворекурсивните решения е (същинско) подмножество на това пространство. Понеже формата на дърветата е произволна, мощността на това множество е много по-висока: при n базови релации броят на различните решения е $\binom{2(n-1)}{n-1}$. Ще означаваме това множество от решения с $\mathcal{A}$.



Ляворекурсивно дърво на обработка



Храстовидно дърво на обработка

Глава 3: Стратегии и алгоритми за подреждане на съединения

Съществуват три основни схеми за пораждане на подредби на съединения:

- *Оптимизация отдолу-нагоре*: Това е синтетичен подход, при който подходящият план се генерира като се започне от базовите релации и постепенно стъпка по стъпка се изграждат все по-пълни частични планове за изпълнение, докато накрая се получи план за изпълнение на цялата заявка.

- *Оптимизация отгоре-надолу*: Това е подход тип "разделяй и владей", при който заявката се разделя на части, всяка част се оптимизира отделно, след което отделните части се обединяват за формирането на цялостния план за изпълнение.

- *Оптимизация чрез трансформации*: Този подход започва от някакъв допустим цялостен план за изпълнение и стъпка по стъпка този план се трансформира в друг допустим цялостен план. На всяка стъпка се запазва планът с най-ниска цена.

Обикновено оптимизаторите на заявки използват елементи от повече от един от горните подходи.

Задачата за намиране на оптималния ред за подреждане на съединение на *n* релации може да се атакува чрез няколко различни типа алгоритми:

Детерминистичните алгоритми извършват някакъв вид детерминистично търсене на пространството на решенията, или чрез пълно обхождане, или чрез прилагането на евристики за ограничаване на пространството на решенията [3].

Рандомизираните алгоритми извършват случайна разходка в пространството на решенията, придвижвайки се от точка в точка. Даден ход между две точки е възможен, ако решението, представено от първата точка може да бъде трансформирано в решението представено от втората точка чрез прилагането на единствено правило за трансформация (измежду множество предварително определени правила за трансформация). Изпълнението на алгоритъма завършва или когато не може да се направи следващ допустим ход от текущата точка в пространството на решенията, или когато е изтекло предварително определено време. Най-доброто решение намерено по време на случайната разходка е резултатът от търсенето.

Генетичните алгоритми наподобяват биологичната еволюция в търсенето си на оптимално решение [29]. Макар да напомня по идея на гореописаните рандомизирани алгоритми, този подход съдържа достатъчно специфични елементи, за да заслужи да бъде разгледан отделно. Основната идея е да се започне от някакво случайно начално поколение решения и да се генерира потомство чрез случайно кръстосване и мутации. Най-добрите елементи на поколението (спрямо ценовата функция) оцеляват и следващото поколение решения се базира върху тях. Алгоритъмът приключва или след пораждането на определен (краен) брой поколения, или когато цялото поколение е достатъчно хомогенно по отношение на ценовата функция [15].

Хибридните алгоритми комбинират елементи от две или повече от гореизброените стратегии. Решенията, получени от някаква детерминистична евристика стават начална точка за рандомизирано търсене или начално поколение за генетичен алгоритъм (подход известен като *посявка*), генетичен алгоритъм се допълва с техники за локално търсене и т.н.

Основните аспекти, които трябва да се отчитат при сравнението на различните алгоритми за задачата за подреждане на съединения са тяхната приложимост и ограничения, влиянието на размера на заявката върху представянето на алгоритмите, свързаността на графа на съединенията и използвания ценови модел.

За малки заявки (такива с до 8-10 съединения), размерът на пространството на решенията позволява прилагането както на детерминистични, така и на недетерминистични алгоритми. Основното преимущество на детерминистичните алгоритми е, че те гарантирано намират глобалния оптимум. Добро изследване на относителното представяне на детерминистичните евристични алгоритми при различни топологии на графа на съединенията и различни ценови модели е представено в [23].

Представянето на евристиката на минималната селективност е сравнително добро при ниски свързаности на графа на съединенията и за ценовия модел на вложените цикли. При

високи свързаности на графа на съединенията и за асиметричния ценови модел на хеш съединението, евристиката се представя лошо.

Изчислението на релационна разлика има много добро цялостно представяне, като постига особено добри резултати за асиметричния ценови модел на хеш съединението дори при високи свързаности на графа на съединенията.

Евристиката „отгоре надолу" се представя много добре при почти всички комбинации на граф на съединенията и ценови модел (с изключение на граф на съединението клика и ценовия модел на вложените цикли) и обикновено успява на намери почти оптимални ляворекурсивни решения.

В обобщение, Изчислението на релационна разлика и Евристиката „отгоре надолу" са най-добрият избор, въпреки че за високо свързани графи на съединенията и двата алгоритъма се представят разочароващо – евристиката „отгоре надолу" при вложените цикли, а Релационната разлика при хеш съединението. Тези резултати предполагат, че по-добро представяне може да се очаква при комбинирането на различни евристики, адаптирайки оптимизацията към формата на графа на съединенията и използвания метод за съединение [23].

A* алгоритъмът, който може да се разглежда като наследник на класическото Динамично програмиране, конструира пълен план за изпълнение на много по-ранна стъпка от Динамичното програмиране и отсича неоптималните поддървета по-агресивно. За малки заявки, A* алгоритъмът има много добро представяне.

IK алгоритъмът се възползва от специалния вид на ценовата формула за вложени цикли и оптимизира дървовидна заявка с N съединения със сложност $O(N^2 \log_2 N)$. Ибараки и Камеда също предлагат алгоритъм, който е приложим дори за циклични заявки и намира добро (но не винаги оптимално) решение със сложност $O(N^3)$.

KBZ алгоритъмът използва в общи линии същите техники, но е по-общ и по-сложен, и е със сложност $O(N^2)$ за дървовидни заявки, при които директно конструира оптималното ляворекурсивно решение [16]. Както и при ИК, приложимостта на KBZ зависи от формата на ценовата функция – тя трябва да е от специален вид. Вложените цикли и хеш-циклите удовлетворяват тези изисквания, но, като цяло, сортираното сливане не е от този тип. KBZ се представя добре при ниско свързани графи, но се справя много лошо при напълно свързания граф клика.

AB алгоритъмът смесва детерминистични и рандомизирани техники и има сложност $O(N^4)$ [26]. Той използва KBZ като подпрограма със сложност $O(N^2)$ и я изпълнява N^2 пъти върху случайно избрани покриващи дървета на графа на заявката. Чрез едно интересно разделяне на ценовата формула на съединението чрез сортирано сливане на част, която влияе на оптимизацията и част, която не влияе, AB алгоритъмът е приложим към всички методи за съединение въпреки ограниченията на KBZ.

При крайно време за работа, рандомизираните алгоритми имат ефективност, която зависи от характеристиките на ценовата функция и свързаността на графа на заявката. Тези резултати са подробно изучени, като са били сравнявани помежду си, а също и с резултатите на динамичното програмиране [24, 11]. Резултатите от сравнението на относителното представяне на Итеративното подобрение и Симулираното изтравване са доста противоречиви – някои автори [25] твърдят, че Итеративното подобрение се представя по-добре при големи нерекурсивни заявки, докато други [11] (по-късно подкрепени от [23]) показват точно

обратното, че Симулираното израстване почти винаги се представя по-добре от Итеративното подобрение.

Сред рандомизираните алгоритми, Итеративното подобрение намира разумно добър план за много кратко време, докато Симулираното израстване изисква повече време, но може да намери по-добро решение. Двуфазовото оптимизиране се възползва от предимствата и на двата подхода и дава най-добри резултати за най-кратко време [7].

За разлика от алгоритмите основани на трансформации като Итеративното подобрение и Симулираното израстване, които обхождат пространството на решенията точка по точка, QuickPick алгоритъмът има по-бърза сходимост и дава по-стабилни резултати. В добавка, алгоритмите основани на трансформации зависят до известна степен от качеството на началното решение, което влияе на стабилността на получените резултати и изисква внимателна настройка на параметрите: ако сходимостта е твърде бърза, алгоритмите могат преждевременно да попаднат в лош локален оптимум [27]. Алгоритмите като Маршрутното симулирано израстване и Двуфазовата оптимизация са разработени за да се справят с този проблем.

Сравнителните експерименти между рандомизираните и генетичните алгоритми са били ограничени основно до емпирични оценки върху конкретни тестови задачи [17]. Някои изследвания показват, че генетичните алгоритми се справят по-добре от Симулираното израстване, което от своя страна се справя по-добре от Итеративното подобрение. Сравнението във времето за изпълнение е също в полза на генетичните (спрямо рандомизираните) алгоритми [23]. Като цяло, резултатите от тези изследвания варират значително, което показва, че единият от двата подхода може да превъзхожда другия в даден клас задачи, а в друг клас ситуацията да е точно обратната.

Теоретичното сравнение между симулираното израстване и генетичните алгоритми [8] показва, че много генетични алгоритми (в частност каноничният генетичен алгоритъм) се характеризират с по-голяма вероятност за намиране на добро решение от симулираното израстване, ако пространството на търсене удовлетворява три ограничения. Тези ограничения обаче не са силни и са в сила при почти всеки избор на генетичните оператори.

Важен аспект в полза на рандомизираните и генетичните алгоритми е, че те се хибридизират лесно, чрез включване на знания специфични за проблемната област. Това ги отвежда от оптимизирането тип „черна кутия” до оптимизирането със знание за проблемната област, което се очаква да се отрази положително на тяхното представяне, както се подсказва от No Free Lunch (NFL) теоремата. Освен това, от гледна точка на дизайн и реализация, и рандомизираните, и генетичните алгоритми могат лесно да се адаптират, за да се възползват от паралелните компютърни архитектури.

Точността на неоптималните решения намирани от рандомизираните и генетичните алгоритми (т.е. тяхната близост до глобалния оптимум) е пропорционална на броя на решенията, които алгоритъмът е разгледал (който от своя страна е пропорционален на времето за изпълнение на алгоритъма) и е силно повлияна от свойствата на пространството на решенията за конкретния екземпляр на задачата. Качеството на решенията зависи също от характеристиките на сходимостта на съответните алгоритми и способността им да се „измъкват” от слаби неоптимални плата в пространството на решенията. Изучаването на корелацията между свойствата на пространството на решенията и трудността на екземпляра на задачата за конкретен клас оптимизационни алгоритми е към днешна дата в ранен стадий на развитие – знанието е все още твърде ограничено и се основава предимно на емпирични

резултати за малки заявки (с до 10 съединени релации). Все още липсват аналитични инструменти или съответна теория, които биха помогнали да се отговори на въпроса какво ще бъде представянето (т.е. отклонението от глобалния оптимум като функция на броя разгледани решения) на конкретен алгоритъм за фиксиран екземпляр на задачата.

Един явен фактор влияещ на ефективността на недетерминистичните алгоритми за търсене е формата на пространството на решенията, разпределението на цените на локалните минимуми и съотношението между броя на „добрите" и „лошите" решения.. Например, пространствата от решения с формата на *кладенец*[1] са по-леки за рандомизираните и генетичните алгоритми и може да се очаква намирането на решения с добро качество. От друга страна, пространствата от решения, които не са гладки и се характеризират с големи разлики в стойностите на ценовата функция представляват значително по-голямо предизвикателство.

Някои емпирични резултати от оценяването на разпределението на цената за пространствата на ляворекурсивните и на храстовидните дървета показва, че за ценовия модел на вложените цикли около 10% от ляворекурсивните решения са не по-лоши от два пъти (2^1) глобалния минимум, а 45% са по-добри или равни на шестнайсет (2^4) пъти глобалния минимум. От друга страна, „качественото съотношение" в пространството на храстовидните решения е много по-ниско – едва около 5% от решенията имат цени по-малки от два пъти глобалния минимум, а 25% са по-добри или равни на 16 пъти глобалния минимум [22].

Едно експериментално изследване на влиянието на свързаността на графа на заявката, селективността на предикатите мощностите на релациите върху формата на пространството на решенията и представянето на евристичните и вероятностните оптимизационни алгоритми е представено в [13]. Заключението е, че формата на пространството на решенията се определя ясно от тези параметри и че представянето на вероятностните оптимизационни алгоритми е пряко свързано с формата на пространството за търсене. От друга страна, евристиките се характеризират с нестабилно представяне, което само отчасти зависи от формата на пространството за търсене.

В заключение може да направим извода, че има твърде малко сравнителни изследвания върху представянето на генетични оптимизационни алгоритми, приложени към задачата за оптимизация на заявки. Не са достатъчно подробно изследвани и изяснени връзките между параметрите на задачата, формата и свойствата на пространството на решенията и представянето на алгоритмите.

Основните резултати от Глава 3 са публикувани в [4А].

Глава 4: Исторически обзор на системите за оптимизиране на SQL заявки

Появата на **System-R** през 1979 г. поставя началото и определя посоката на развитие на оптимизаторите на заявки през следващите десетилетия. Може би най-важният принос на тази система е оптимизацията, основана на цената за изпълнение. Оптимизаторът използва

[1] такива пространства имат две характеристики – разликата в цените на кои да е два локални минимума е малка и съществува път между кои да е два локални минимума, чиито елементи имат цени близки до тези на минимумите, които свързва (т.е. областта на решенията с ниска цена е *гладка*) [12]

статистики за релациите и индексите, които се съхраняват в системния каталог. На базата на тези статистики се изчислява цената на всеки план за изпълнение, по формула, която за пръв път отчита като ресурс и процесорното време. Системата също така използва ефективен метод за определяне на реда на съединенията. Друг важен принос е стратегията на търсене на System R – динамичното програмиране отдолу нагоре.

Въпреки огромното влияние върху изследванията в областта на оптимизацията на заявки, System R страда от един основен недостатък – архитектурата му не позволява лесно разширяване за включване на нови трансформации. Това води до разработването на нови оптимизатори с разширяеми архитектури, които улесняват добавянето на нова функционалност. Пионерите в тази област са проектът Exodus за разширяема СУБД и по-късно генераторът на оптимизатори Volcano, разработен от един от участниците в проекта Exodus – Гьоц Грефе и Уилиам МакКена. Гьоц Грефе по-късно дефинира и инфраструктурата Cascades, поправяйки грешки присъстващи в предишните му две начинания.

Генераторът на оптимизатори **Exodus** [5] е първата разширяема архитектура на оптимизатор, която използва стратегия на търсене отгоре надолу. Целта на проекта Exodus е да се създаде инфраструктура и средство за оптимизация на заявки, като се правят минимални предположения за модела на данните. Входните данни за Exodus представляват файл, описващ модела на данните, в който се дефинират множество методи, трансформационни правила (преобразуващи дървото на заявката) и реализационни правила (определящи съответствието между логическите оператори и методите за тяхната реализация). Генераторът трансформира този моделен файл в програма на C, която се компилира и свързва с набор готови функции. Полученият оптимизатор преобразува първоначалната заявка стъпка по стъпка, като съхранява всички алтернативи, изследвани до момента. Във всеки момент се поддържа множество от възможни следващи трансформации, които се съхраняват в опашка.

Основният принос в Exodus е стратегията за оптимизиране отгоре надолу, разделянето на стратегията на търсене от модела на данните и разделянето на трансформационните правила и логическите оператори от правилата за реализация и физическите оператори. Макар конструирането на оптимизатори с Exodus да не е лесна задача, тази система поставя основите на следващото поколение разширяеми оптимизатори.

Оптимизаторът **Starburst** [18] на IBM разширява оптимизатора на System R с един по-ефективен и по-разширяем подход. Оптимизаторът на Starburst се състои от две подсистеми, основани на правила: синтактичен преобразувател или модел на графа на заявката и планиращ модул. Моделът на графа на заявката е вътрешното, семантично представяне на заявката. Този модул използва множество трансформационни правила, чрез които преобразува евристично един модел на графа в друг, “по-добър”. Целите на тази фаза са опростяване и амелиорация: премахване на излишествата и получаване на изрази, които са по-прости за ценово оптимизиране от гледна точка на планиращия модул. Планиращият модул обработва заявки тип селекция-проекция-съединение. Модулът се състои от номератор на съединенията и генератор на планове. Номераторът на съединенията използва евристики за ограничаване на възможностите за пораждане на различни редици от съединения. Алгоритъмът, който се използва, не е основан на правила, написан е на C и модулната му архитектура позволява да бъде заменен с друг номериращ алгоритъм. Генераторът на планове използва граматика от параметризирани продукционни правила за конструирането на плана за изпълнение. Тези правила определят коя релация да е външна и коя вътрешна в едно съединение, кой метод за реализиране на съединение да се използва и т.н.

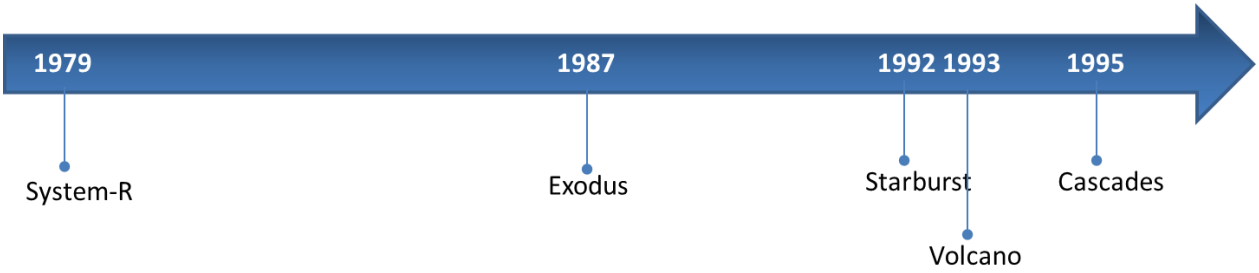
В Starburst оптимизацията е двуфазов процес. През първата фаза, някакво начално вътрешно представяне на заявката като модел на граф на заявката се подава на синтактичния преобразувател и се трансформира в нов, по-добър модел. Този нов модел се подава на планиращия модул, за да се конструират планове за изпълнение и да се избере най-добрият от тях чрез стратегия на търсене отдолу нагоре, подобна на тази в оптимизатора на System R.

Евристиките, които използва оптимизаторът обаче, понякога могат да доведат до вземането на погрешни решения, защото се базират само на логическа информация, т.е. не се основават на ценови модел. Освен това, евристиките са такива, че тяхното разширяване за по-сложни заявки (например съдържащи нерелационни оператори) е много трудно.

Създаден да подобри ефективността на Exodus, генераторът на оптимизатори **Volcano** [6] е проектиран да постигне по-добра ефективност и да има по-големи възможности за разширение. Ефективността се подобрява чрез комбинирането на динамичното програмиране с насочено търсене, основано на физическите свойства, *branch-and-bound* отсичане и евристики. Алгоритъмът за търсене се нарича *насочено динамично програмиране*. Това е стратегия за търсене отгоре надолу. Частите на даден израз се оптимизират само ако участват в обещаващ по-общ план. Всички оптимални междинни планове, а също и всички неуспешни междинни решения се съхраняват до пълното оптимизиране на заявката. Като използва физическите свойства (обобщение на "интересните наредби" в System R), алгоритъмът развива само обещаващите частични планове. Разширяемостта се постига чрез генериране на програма на C от модела на данните и чрез капсулиране на логическите и физическите свойства в абстрактни типове данни.

Чрез Volcano са генерирани два успешни оптимизатора, един за ООБД и един за БД с научно приложение.

Проектът **Cascades** [4] е отворена архитектура за генериране на оптимизатори, която преодолява много от недостатъците на Exodus и Volcano. Постига се съществено подобрение на функционалността, улеснява се използването на системата без да се жертва разширяемостта. Този генератор на оптимизатори става основа за разработването на NonStop SQL на Tandem и на Microsoft SQL Server 7.0 (1999).



Хронология на възникването на системите за оптимизиране на заявки

Глава 5: Генетичен алгоритъм за оптимизиране на заявки

Нека разгледаме следната конкретизация на каноничния генетичен алгоритъм:

Представяне на решенията

Ще се ограничим до разглеждането на пространството на ляворекурсивните решения. Всяка хромозома се кодира с масив от гени, като всеки ген съдържа два елемента – номер на релация и номер на метод за съединение.

Операторът за кодиране Θ преобразува индивида ξ_i във вектор от гени, като всеки ген е наредена двойка от номер на релация и номер на метод за съединение:

$$\Theta(\xi_i) = \Theta(R_{i1} \bowtie_{p1} R_{i2} \bowtie_{p2} \ldots \bowtie_{pn-1} R_{in-1} \bowtie_{pn-1} R_{in}) \rightarrow ((i_1, p_1), (i_2, p_2), \ldots, (i_n, p_n))$$

Този метод на кодиране очевидно допуска създаването на декартови произведения при изпълнение на заявката, но смятаме, че тяхното елиминиране би било по-неефективно от изчислителна гледна точка отколкото тяхното обработване.

Оператор за мутация

Операторът за мутация M преобразува индивида ξ_i в нов индивид ξ'_i като разменя два случайно избрани гена γ_x и γ_y и като променя метода за съединение на друг случайно избран ген γ_m:

$$M(\xi_i) = M(((i_1, p_1), \ldots, (i_x, p_x), \ldots, (i_m, p_m), \ldots, (i_y, p_y), \ldots, (i_n, p_n))) \rightarrow (((i_1, p_1), \ldots, (i_y, p_y), \ldots, (i_m, p'_m), \ldots, (i_x, p_x), \ldots, (i_n, p_n)));\ x, y, m \in \{1, 2, \ldots, n\},\ x \neq y,\ p_m \neq p'_m$$

Мутационната активност μ е вероятността произволен индивид ξ_i да мутира в дадено поколение, т.е. $M(\xi_i) \equiv \xi_i$ с вероятност $(1 - \mu)$.

Операторът не се прилага върху индивида от популацията с максимална жизнеспособност. В случай че в популацията съществуват два или повече индивида с жизнеспособност равна на текущо максималната, "защитен" от мутация е само един от тях. Предпазването на текущо оптималния индивид на популацията от мутация предотвратява загубата на ценен генетичен материал и в крайна сметка се отразява положително на ефективността на оптимизационния алгоритъм.

Оператор за кръстосване

Операторът за кръстосване X комбинира хромозомите на два индивида $\xi^g{}_i$ и $\xi^g{}_j$ от поколение g за да получи два нови индивида $\xi^{g+1}{}_i$ и $\xi^{g+1}{}_j$ от поколение $(g+1)$. Избира се случаен локус x, двата родителски хромозома се разцепват в този локус, всяко от двете деца наследява цял фрагмент от единия родител (първото дете – левия, а второто – десния спрямо локуса), а останалата част от хромозома се попълва с липсващите гени в реда, в който те се срещат у втория родител:

$$X(\xi^g{}_i, \xi^g{}_j) = X(((i_1, p_1), \ldots, (i_x, p_x), (i_{x+1}, p_{x+1}), \ldots, (i_n, p_n)), ((j_1, q_1), \ldots, (j_x, q_x), (j_{x+1}, q_{x+1}), \ldots, (j_n, q_n))) \rightarrow \{((i_1, p_1), \ldots, (i_x, p_x), (j'_{x+1}, p'_{x+1}), \ldots, (j'_n, q'_n)), ((i'_1, p'_1), \ldots, (i'_x, p'_x), (j_{x+1}, q_{x+1}), \ldots, (j_n, q_n))\},\ x \in \{1, 2, \ldots, n\}$$

Това гарантира структурна и функционална прилика на децата и с двата родителя. Ако с n означим размера на хромозома в брой гени, сложността на този оператор е $O(n^2 + n)$.

Всеки индивид ξ_i от популацията си избира партньор за кръстосване от своята *околност*, която се определя като неговите k съседа по индекс във вектора на популацията, $k = const$. Вероятността даден индивид ξ_j от околността да бъде избран за партньор за кръстосване е пропорционална на неговата жизнеспособност $\varphi(\xi_j)$.

Оператор за селекция

Операторът за селекция Σ преобразува една популация ξ в друга популация $\xi' \subseteq \xi$:

$$\Sigma(\xi) = \Sigma(\{\xi_1, \xi_2, \ldots, \xi_k\}) = \{\xi_{i1}, \xi_{i2}, \ldots, \xi_{im}\},\ m \leq k.$$

Всички класически алгоритми за селекция налагат едно съществено ограничение – размерът на популацията да е инвариантен, т.е., разглеждани като оператори, трансформиращи една популация в друга, да запазват мощността на аргумента си. Предимствата на подобно ограничение са, че то гарантира използването на константна по размер памет за оптимизацията, а също и значително опростява самите алгоритми. Подобно ограничение обаче е изкуствено и не следва никаква аналогия с биологичната еволюция, при която броят на индивидите в популацията не само, че не е фиксиран, но и непрекъснато се изменя във времето, като нараства при наличие на индивиди с висока жизнеспособност в популацията и изобилие на природни ресурси и намалява при тяхното отсъствие. Този биологичен факт ни навежда на идеята, че вероятно един метод за селекция, който допуска разширяване и свиване на популацията би бил по-добър. Отсичането на популацията след сортиране не само няма никаква теоретична обосновка, но може и силно да ощети оптимизационния процес. Да си представим следната ситуация, която възниква сравнително често – след сортиране на популацията в намаляващ ред на жизнеспособност на индивидите, след позиция N в списъка съществува група индивиди, чиято жизнеспособност е или равна на жизнеспособността на N-тия индивид, или е много близка до нея. Пренебрегването на индивидите след "чертата" ще доведе до отхвърляне на индивиди, които са не по-лоши от най-лошите, които селекцията ще запази. Това очевидно ще доведе до загубата на ценен генетичен материал. Ако решим да не правим подобно грубо отсичане, възниква проблемът къде точно да спрем с приемането на индивиди, които да оцелеят. Ако не искаме да приемем никаква загуба на индивиди, първо ще обезсмислим прилагането на оператора за селекция (той трябва все пак да *подбира*!), и второ, ще се сблъскаме с едно лавинообразно нарастване на популацията (след всяка селекция размерът й ще нараства три пъти). Интуитивно желанието ни е да отсечем там, където в монотонно намаляващото качество на индивидите се забележи някакъв рязък спад. Също така бихме желали размерът на популацията, макар и не фиксиран, да се движи в определени граници – от една страна да не нарасне неограничено, а от друга да не намалее до размери, при които разнообразието е сведено до минимум и е вече невъзможно да се реализира конкуренция между индивидите.

Предлаганият оригинален алгоритъм за селекция постига тези изисквания. При него всички индивиди в популацията са равноправни в смисъл, че шансът за оцеляване на всеки е право пропорционален на жизнеспособността му (както е при класическата селекция тип "рулетка на Съдбата", но без да се налага фиксиране на размера на популацията). Това

гарантира, че при наличие в популацията на групи от индивиди с равна или сходна жизнеспособност, тези групи или като цяло ще оцелеят, или като цяло ще отпаднат и се изключва вероятността от несправедливо отсичане. Намерено е и решение на проблема с контрола върху размера на популацията. Интуицията подсказва, че е логично популацията да се разраства в началните поколения на своето развитие, когато генетичният материал е много разнообразен и трябва да се даде възможност за "експериментиране" на различни характеристики на индивидите, а трябва да намалява с нарастване на конвергенцията на индивидите, когато тяхното уеднаквяване в смисъл на структура и на качество вече не оправдава поддържането на голяма популация и свързаните с това увеличени разходи на изчислителни ресурси като памет и най-вече процесорно време за осъществяване на кръстосване, мутация и селекция.

Предлаганият алгоритъм постига този желан контрол над размера на популацията като дефинира *колективна* вероятност за оцеляване на популацията (т.е. да се разшири или да се свие), наред с *личната* вероятност на всеки индивид да оцелее.

Личната вероятност за оцеляване на всеки индивид $\xi_i \in \xi$ се определя като $p_i = \varphi(\xi_i) / \varphi^*$, където $\varphi(\xi_i)$ е жизнеспособността на ξ_i, а φ^* е максималната жизнеспособност в популацията. С други думи, $\xi_i \in \xi'$ с вероятност p_i за всеки $\xi_i \in \xi$.

Колективната вероятност за оцеляване на популацията се определя чрез отношението на *предполагаемия размер на популацията след селекцията* и *желания размер на популацията след селекцията*. Ще апроксимираме очаквания размер[2] на популацията s^E със сумата от вероятностите за оцеляване на всеки неин индивид[3].

Размерът N на популацията след прилагане на селекция може да намалее (в крайния случай – до един индивид – най-добрият), да се запази без промяна или да нарасне (в крайния случай – до 3.N, при положение, че оцелеят всички стари индивиди и всички новополучени чрез кръстосване). Крайният вариант за намаляване е неприемлив – свеждане на популацията до един индивид практически означава край на оптимизационния процес – бихме искали в най-лошия случай размерът да намалее до първоначалния размер на популацията (какъвто е бил броят на индивидите от нулевото поколение). Разрастването на популацията бихме желали да е обратно пропорционално на степента на конвергенция на индивидите. За мярка за степента на конвергенция на жизнеспособността можем да приемем отношението на средната жизнеспособност в популацията към максималната. Тогава ще представим желания размер[4] на популацията s^D като линейна комбинация на двете крайни алтернативи с коефициенти, зависещи от степента на конвергенция:

$$s^D = c.s^0 + 3(1 - c)N,$$
$$c = a / b,$$

където s^D е желаният размер,
s^0 е началният размер,
N е текущият размер,
c е степента на конвергенция,

[2] очакваният размер може да не е цяло число

[3] точното пресмятане на математическото очакване на сума от N случайни величини е доста по-сложно, а голяма точност не ни е нужна

[4] Желаният размер може да не е цяло число

a е средната жизнеспособност и
b е максималната жизнеспособност.

Колективната вероятност за оцеляване p^* се определя като нормализираното отношение на желания към очаквания размер на популацията ($p^* = s^D / (s^D + s^E)$).

Индивидуалната вероятност за оцеляване p_i след това се мащабира нагоре или надолу в зависимост от колективната вероятност за оцеляване p^*.

Важно е са се отбележи, че при така описания алгоритъм, индивидът с максимална жизнеспособност в популацията оцелява с вероятност 1.0, т.е. предлаганият оператор за селекция винаги запазва най-добрия индивид.

Поради вероятностния характер на определянето на това дали даден индивид ще оцелее или не, размерът на популацията след селекция е случайна величина и фиксирането на s^0 като минимален размер във формулата за определяне на желания размер съвсем не е достатъчно да ни гарантира, че след селекция броят на индивидите в популацията ще е поне s^0. Понеже падането на размера на популацията под s^0 е крайно нежелателно, необходимо е в случай, че тази минимална бройка не е попълнена при селекцията, след това популацията да се допълни с липсващия брой индивиди като за целта на една допълнителна стъпка на алгоритъма се генерират нови случайни индивиди. Решението да се предпочетат нови случайни индивиди вместо да се "реабилитират" част от вече отхвърлените алтернативи при селекцията е продиктувано от желанието да се внесе допълнително разнообразие в генотипа, такова, каквото не може да се постигне само с прилагането на оператора за мутация. Така постигането на основната цел – гарантиране на някакъв минимален размер на популацията във всеки един момент - води и до един положителен страничен ефект – увеличава се генетичното разнообразие в много по-висока степен, отколкото единствено чрез използване на оператора за мутация, което намалява значително риска от "зацикляне" на оптимизацията в локален оптимум с лошо качество.

Сложността на предлагания алгоритъм е $O(N)$ срещу $O(N^2)$ за Елитарната селекция, където N е размерът на популацията.

Критерий за спиране

Алгоритъмът приключва или при изчерпване на предварително зададен краен брой поколения, или при генериране на поколение с коефициент на сходимост под предварително зададен праг:

,

където е максималният брой поколения, а е коефициентът на сходимост. И двете константи са конфигурируеми параметри на алгоритъма.

Сходимост на алгоритъма

Предложеният генетичен алгоритъм е глобално сходящ, т.е. жизнеспособността на най-добрия индивид в популацията клони към глобалния оптимум в пространството на решенията когато броят на поколенията клони към безкрайност. Това е следствие от свойствата на трите генетични оператора: операторът за селекция запазва най-добрия индивид в популацията с вероятност 1.0, а операторите за мутация и кръстосване гарантират, че всяка точка в пространството на решенията е достижима тръгвайки от произволно избрано начално множество от точки (т.е. индивиди в началното поколение). Формално доказателство за сходимост на генетичния алгоритъм за оптимизация при горните изисквания към свойствата на генетичните оператори може да бъде намерено в [20].

Основните резултати от Глава 5 са публикувани в [1A] и [2A].

Глава 6: Реализация на генетичен оптимизатор на заявки

Архитектура на системата

Архитектурата на оптимизатора се описва от следната диаграма:

Синтактичният анализатор проверява синтактичната коректност на заявката и я превежда във вътрешно представяне, което се използва от останалите модули.

Семантичният анализатор проверява семантичната коректност на заявката (във вътрешното и представяне) срещу системния каталог (т.е. речникът на данните на схемата на

базата данни). Той валидира консистентността на релациите, атрибутите и типовете данни, а също така и разрешава референциите към атрибутите.

Системният каталог поддържа данни за релациите в схемата – т.нар. *профили на релации*. В нашия прототип системният каталог е помощен модул дефиниран от XML дескриптор. За всяка релация се поддържат параметрите име, кардиналност, атрибути, а за всеки атрибут – име, тип, разпределение на стойностите, налични индекси. Поддържа се един-единствен тип данни за атрибутите – целочислен. Системният каталог разпознава два вида индекси – хеш индекс и B-дървовиден индекс.

Модулът за оценка на размера и разпределението осигурява данни за кардиналностите на релациите и честотното разпределение на стойностите на атрибутите в дефиниционните им области. Тези данни са необходими за определянето на селективността на предикатите в заявката, което е фундаментално изискване на всеки ценови оптимизатор.

Статистиката за честотното разпределение на стойностите на атрибутите в дефиниционната им област се поддържа от *хистограми с равна ширина*. Хистограмите на всички атрибути на всички релации, които са описани в системния каталог съдържат еднакъв брой сегменти, който може да се конфигурира.

Генетичният оптимизатор използва оптимизационния алгоритъм описан в предходната глава.

Ценовият модел е помощен модул дефиниран от XML дескриптор и представлява списък от описания на ценови параметри. Поддържат се пет конфигурируеми ценови параметъра, всеки от тип число с плаваща запетая: цената на изчитането на релация от диска и зареждането и в оперативната памет, цената на обработката (обхождането) на един елемент на релация (т.е. *n*-торка), цената за достъпване на елемент на релация през хеш индекс и B-дървовиден индекс и цената на сортирането на елементите на релация по стойностите на атрибут.

Каталогът на методите определя наличните методи (т.е. физически реализации) за всеки оператор от релационната алгебра. Нашата реализация поддържа три реализации на съединение (съединение с вложени цикли, съединение чрез сливане и хеш съединение) и две реализации на селекция (обхождане по индекс и пълно обхождане, придружено с опционално филтриране по предикат).

Системен интерфейс

Входните данни за оптимизатора са три вида:

- данни за релациите, атрибутите, наличните индекси, разпределенията на стойностите на атрибутите и т.н. Тази информация се поддържа и осигурява от системния каталог;
- данни за наличните методи за съединение, тяхната цена, цената на всички останали операции, които могат да фигурират в плана за изпълнение (сортиране, търсене в хеш индекс,

търсене в B-дървовиден индекс, двоично търсене, обработка на единична *n*-торка и т.н.). Тази информация се поддържа и осигурява от ценовия модел;

- текст на конкретна SQL заявка.

В настоящата реализация разглеждаме едно подмножество на езика SQL, което ще дефинираме тук. Общият вид на заявките, които системата приема като вход е:

```
SELECT attr1, attr2, ..., attrN
FROM rel1, rel2, ..., relM
[WHERE predicate(attri1, ..., attrik)]
[ORDER BY attrl];,
```

където $attr_i$ са имена на атрибути, а rel_j са имена на релации. Поддържат се само *вътрешни съединения* (*inner joins*), и те се изразяват с неявна нотация. Не се поддържат циклични заявки.

Формат на резултата

Резултатът от оптимизацията на заявката е процедурен план за нейното изпълнение. Планът представлява списък от базови операции, т.е. обръщения към примитивни функции от ниско ниво на СУБД, които трябва да се изпълнят в указания ред за получаване на резултата. С цел опростяване на плана за изпълнение, който се генерира, част от операциите, които изграждат плана за изпълнение са от по-високо, абстрактно ниво, но отново се свеждат до извикване на няколко примитивни функции на СУБД, т.е. част от операциите на най-ниско ниво са направени прозрачни за удобство, тъй като детайлите по реализацията им не касаят пряко оптимизационния процес[5].

Планът за изпълнение се изгражда от следните базови операции:

- **`FILE SCAN(relation)`** Това е операция от високо ниво, която зарежда дадена релация в оперативната памет.

- **`FILTER(relation, predicate)`** Операцията филтрира елементите на дадена релация по даден предикат.

- **`JOIN(relation, relation, method)`** Операцията осъществява съединение на две релации по зададен метод за съединение и с евентуалното използване на индекс по дясната релация, зададен с директивата `USE INDEX`.

[5] например примитивната операция FILE SCAN, която зарежда релация в оперативната памет очевидно не винаги би могла да зареди *цялата* релация наведнъж – обикновено изчитането на релации с висока мощност се извършва като последователни техни части се зареждат в буфер в оперативната памет, чийто размер е константен, но може да се конфигурира

- **`USE INDEX(relation, attribute)`** Тази директива се използва в комбинация с операцията `JOIN` и указва на СУБД да използва определен индекс за достъп до релацията, вместо да я зарежда цялата в оперативната памет.

- **`SORT(relation, attribute)`** Сортира дадена релация по стойностите на даден атрибут. Използва се в два контекста – за сортиране на крайния резултат при наличие на `ORDER BY` клауза в заявката и за сортиране на двата аргумента на операцията `JOIN` за реализиране на съединение чрез сортиране и сливане.

- **`PROJECT(relation, attribute1, ..., attributeN)`** Осъществява проекция на дадена релация по даден списък от атрибути.

Основните резултати от Глава 6 са публикувани в [ЗА].

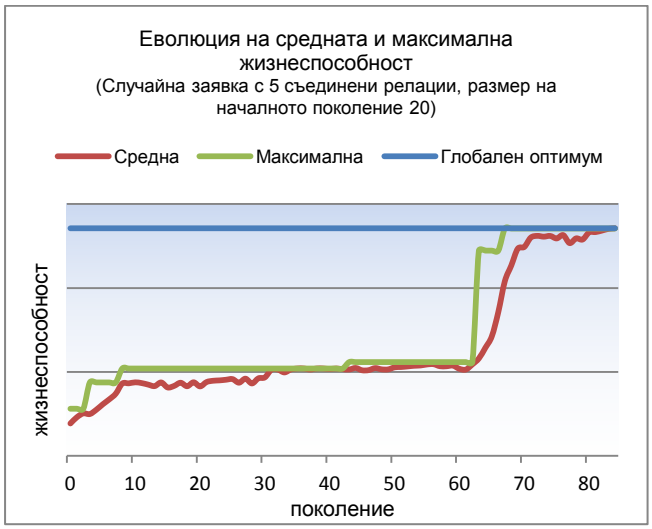
Глава 7: Експериментални резултати и сравнителен анализ

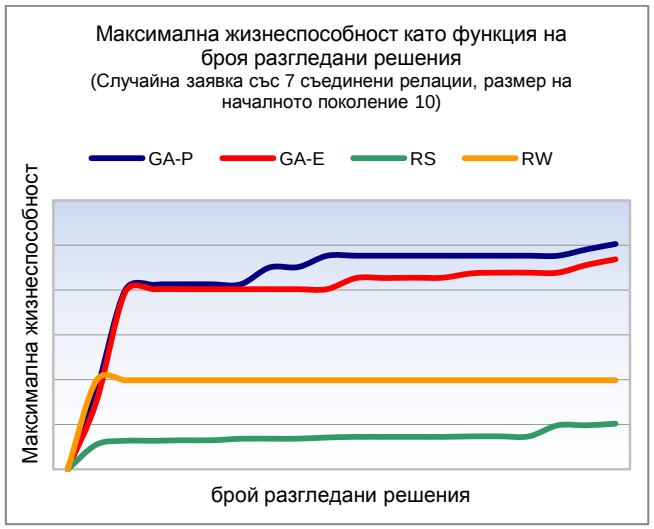
Резултати

Ще сравним предлагания генетичен алгоритъм (ще го означаваме с GA-P) с най-популярния класически генетичен алгоритъм, използващ елитарна селекция (ще го означаваме с GA-E) и с двата най-прости рандомизирани оптимизационни алгоритми, *Случайното Търсене* (RS) и *Случайната Разходка* (RW). При GA-E размерът на популацията е фиксиран по време на целия оптимизационен процес. Ако N е фиксираният размер на популацията, елитарната селекция просто сортира индивидите в популацията в намаляващ ред на тяхната жизнеспособност и запазва първите N от тях. Алгоритъмът RS поражда крайна редица от случайни решения и запазва намереното решение с най-добра цена. Алгоритъмът RW започва от случайно избрана точка в пространството на решенията и на всяка итерация прави „ход" от текущата точка към някоя „съседна" точка (т.е. такава, която може да бъде достигната чрез прилагането на някаква трансформация към текущата точка), ако този ход подобрява цената на текущата точката.

Всички експерименти са проведени върху случайно генериран речник на данните и случайно генерирани заявки. Във всички експерименти е използван един и същ опростен ценови модел. Избраната мутационната активност е 0.1, а стойността на параметъра k определящ размера на областта на съседство е 6. Всеки тест на всеки алгоритъм се състои от 10 изпълнения, като резултатите се осредняват.

Първо ще разгледаме представянето на GA-P при оптимизирането на заявка с 5 съединени релации. Максималната жизнеспособност монотонно нараства и достига глобалния оптимум около 70-тото поколение.



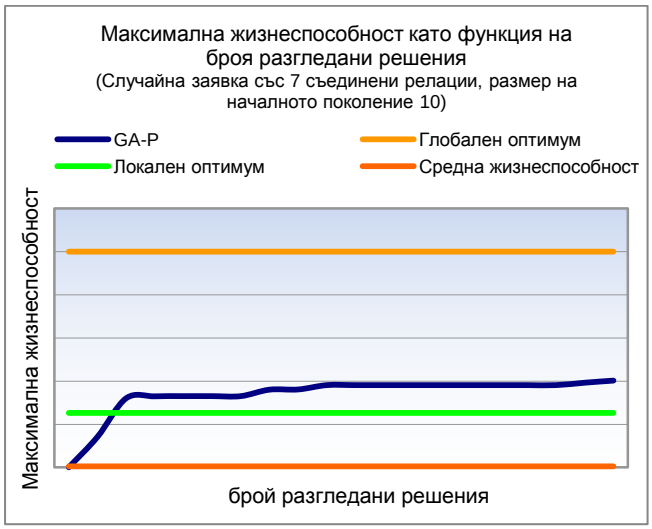


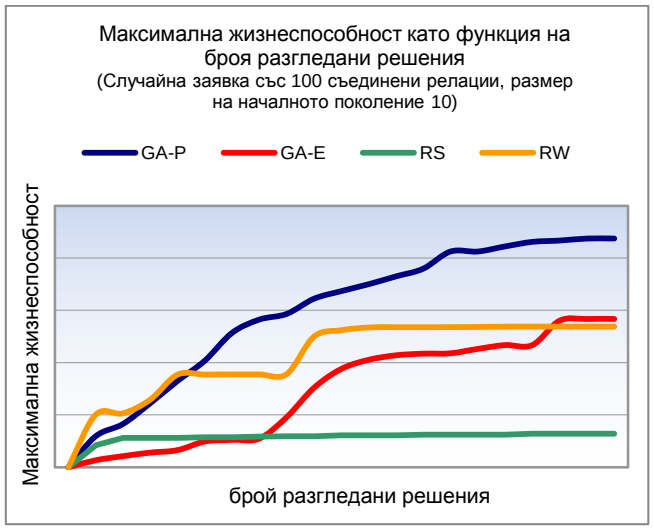
Сега нека разгледаме представянето на алгоритмите при оптимизирането на заявка със 7 съединени релации. Изборът на този конкретен размер на заявката не е случаен – това се счита за горната граница на размера за "класически" (или "малки") заявки и грубо очертава границите на приложимост на детерминистичните оптимизационни алгоритми.

Сравнителното представяне на четирите алгоритъма е аналогично и при случайна заявка с 10 съединени релации и размер на началното поколение 100.

Двата генетични алгоритъма очевидно превъзхождат двата рандомизирани. Рандомизираните стратегии за търсене лесно попадаха в лоши локални оптимуми и не успяха да стигнат до решения, които да конкурират най-добрите намерени от генетичните алгоритми. Двата генетични алгоритъма показват като цяло сходно представяне, но GA-P изглежда малко по-добре, когато броят разгледани решения нараства. Едно изразено предимство на GA-P е по-бавната сходимост в сравнение с GA-E, което позволи на GA-P да избегне някои неоптимални плата, в които попадна GA-E. Вероятната причина е че популациите при GA-E се характеризират с по-слабо разнообразие (много близки стойности на минималната, средната и максималната жизнеспособност в популацията).

GA-P се представя много добре що се отнася до качеството на намерените решения. Фигурата в ляво показва еволюцията на максималната жизнеспособност при GA-P в границите на средната жизнеспособност в пространството на решенията и глобалния оптимум. Показан е също и един от най-добрите локални оптимуми (в който попаднаха много от изпълненията).

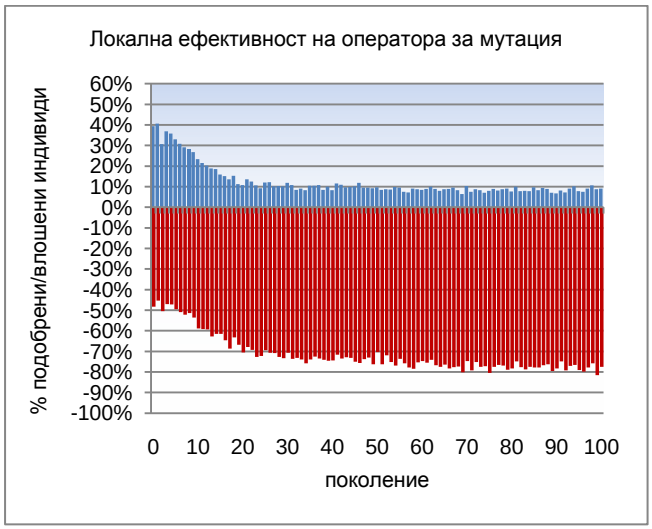




В следващия тест сравняваме представянето на четирите алгоритъма върху много по-трудна задача – заявка със сто съединени релации. Тук превъзходството на двата генетични алгоритма над RS е още по-изразително. RW обаче се представя изненадващо добре, с резултати често сравними с тези на генетичните алгоритми. Сравнителното представяне на GA-P и GA-E варира в значителна степен в зависимост от началния размер на популацията. Не можахме да стигнем до правило, свързващо относителното представяне на двата генетични алгоритъма с началния размер на популацията.

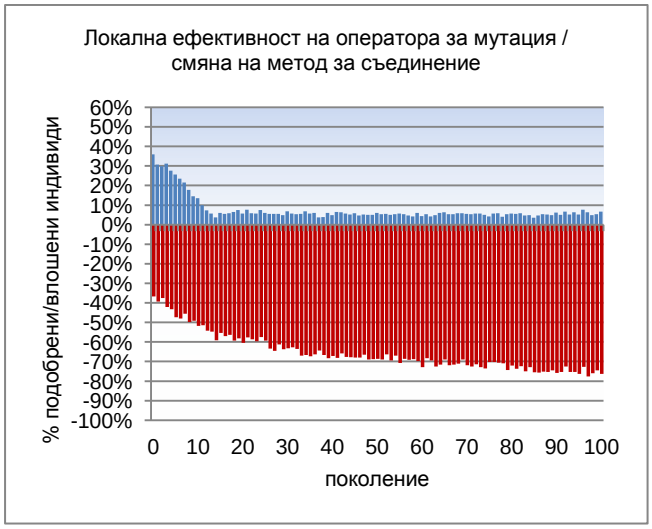
Нека сега разгледаме *локалната* ефективност на операторите за мутация и кръстосване, т.е. способността им да подобряват жизнеспособността на индивидите при еднократното им прилагане. Тъй като локалната ефективност зависи от качеството на входните решения (което нараства от поколение в поколение при подходящ избор на оператора за селекция), интересуваме се също от еволюцията на локалната ефективност през поколенията. Всички експерименти за локална ефективност са проведени в пространство на решенията дефинирано от случайна заявка с 50 съединени релации, размер на началното поколение 100. Използва се вероятностна селекция с динамична популация, освен където е указано друго.

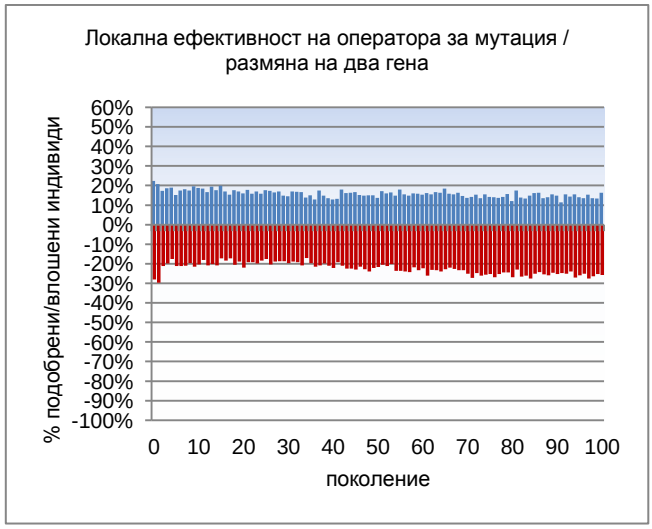
Ще започнем с изследване на ефективността на оператора за мутация като средния процент подобрени и съответно влошени жизнеспособности на индивидите, към които е приложен.



Ефективността на оператора за мутация започва около 40% и постепенно намалява до около 10% след 20 поколения, след което се запазва сравнително постоянна. Трябва да отбележим, че сумата на процентите подобрени и влошени решения не е винаги 100%, като разликата е процентът решения, чиято жизнеспособност остава без промяна след прилагането на оператора.

Тъй като използваният оператор за мутация прилага две независими трансформации – смяна на метода на съединения на случайна релация и размяна на две случайни релации, интересно е да анализираме отделния принос на всяка от тези трансформации.

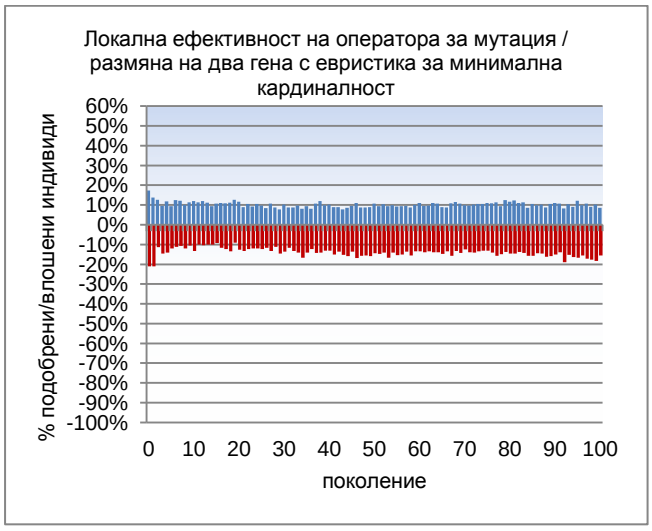


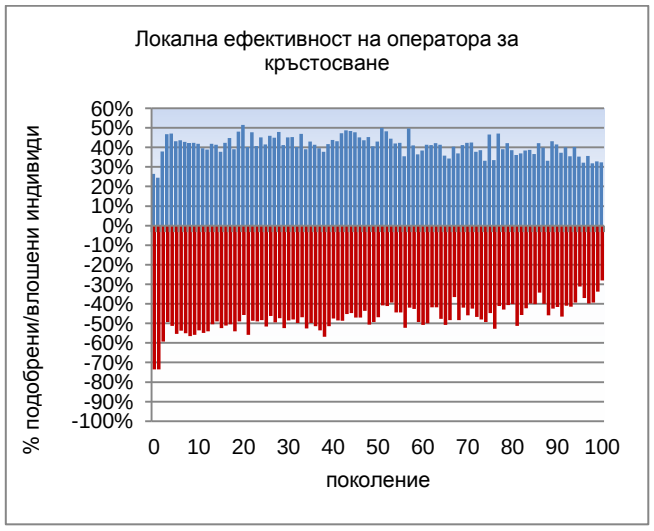


Можем да направим извод, че смяната на метода за съединение е ефективна само в първите поколения, вероятно докато се оптимизира използването на наличните индекси. За разлика от тази трансформация, размяната на два гена има стабилна ефективност.

Трансформацията с размяна на два гена ни дава възможност да изследваме ефективността на една от най-популярните евристики при определяне на оптималната подредба на съединенията – евристиката за минимална кардиналност. Следвайки тази

евристика, релациите с ниска кардиналност трябва да бъдат съединени преди тези с висока. Затова модифицирахме алгоритъма на оператора за мутация, така че размяната на две релации да се извършва с вероятност пропорционална на отношението на кардиналностите на двете релации.





Хибридизирането на генетичния алгоритъм с евристиката за минимална кардиналност намалява процента влошени решения, но (противно на очакването ни) намалява и процента на подобрените решения.

Като критерий за локалната ефективност на оператора за кръстосване ще ползваме процента случаи, в които прилагането на оператора води до получаването на поне един наследник с жизнеспособност по-добра от средната жизнеспособност на двамата си родители. Експериментът е проведен при стандартната околност на кръстосване 6, т.е. всеки индивид „избира“ партньор за кръстосване от околните си 6 индивида в популацията. Локалната ефективност на оператора се влияе от стойността на този параметър - при околност с размер 2, локалната ефективност е сравнима с тази при размер 6, но видимо спада след 60-тото поколение. При околност за кръстоване 12, локалната ефективност е по-висока с няколко процента и бележи значим спад едва около 80-тото поколение.

Резултатите показват, че 6 е един добър избор за околност за кръстосване – увеличаването на размера на околността логично подобрява локалната ефективност (всеки индивид има по-голям „избор“ и съответно се кръстосва с индивид със статистически по-висока жизнеспособност), но разликата е незначителна с локалната ефективност на оператора с околност 12. Трябва да се има предвид, че по-голямата околност е свързана с известно забавяне на алгоритъма за кръстосване, а и може да се очаква, че колкото по-голяма е околността за кръстване, толкова по-бързо сходящ ще е алгоритъмът (с произтичащата от това опасност от попадане в „лоши“ локални оптимуми).

Изводи

Може да заключим, че сравнението между GA-E и GA-P е в полза на втория алгоритъм. Макар при определени ситуации GA-E да завършва за по-кратко време, в общия случай GA-P е по-бърз поради линейната си зависимост от размера на популацията. Що се отнася до качество на намираните решения (нещо, за което критикуват рандомизираните и генетичните подходи като цяло), GA-P превъзхожда GA-E с по-плавната си сходимост и възможност за излизане от „слаби" локални оптимуми.

Като цяло резултатите категорично доказват приложимостта и ефективността на генетичния подход към задачата за оптимизиране на заявки към БД.

Част от резултатите в Глава 7 са представени в [2A].

Глава 8: Заключение

В този труд направихме обзор на принципите, системите и алгоритмите за оптимизиране на заявки към релационни БД. Специално внимание отделихме на проблема за оптимизиране на заявки с голям брой съединения като задача, възникваща все по-често в съвременните приложения на БД. Достигайки до извода, че класическите детерминистични алгоритми са неприложими към такива заявки, разгледахме основните недетерминистични подходи приложими към $\mathcal{NP}$-пълни оптимизационни задачи. На базата на предишни изследвания върху прилагането еволюционни оптимизационни алгоритми към задачата за оптимизиране на заявки, се спряхме на генетичните алгоритми като добра алтернатива на детерминистичните за заявки с голям брой съединения. Анализирайки недостатъците на генетичните алгоритми, приложени към този момент към задачата, предложихме оригинален адаптивен генетичен алгоритъм и показахме, че той превъзхожда класическите. Реализирахме оптимизатор на заявки използващ този алгоритъм, а като междинни цели създадохме библиотека за разработване на оптимизатори, както и средства за автоматизиране на експериментите и сравняване на представянето на генетични и рандомизирани алгоритми.

ПЕРСПЕКТИВИ

Работата, започната в този дисертационен труд може да продължи в следните насоки:

Генетични оператори. Настоящата разработка използва оригинален алгоритъм за селекция, но реализира класически варианти на операторите за мутация и кръстосване. Едно интересно продължение на работата би включвало разработването и експериментирането с нови алгоритми за тези оператори. Адаптивността в генетичните алгоритми е област на активни изследвания в последните няколко години и освен оператора за селекция, кръстосването и мутацията също биха могли да се параметризират и да се реализират с адаптивен алгоритъм. Предизвикателна насока за бъдещо развитие е представянето на трите генетични оператора като функции в релационната алгебра и изследването на техните математически свойства, в т.ч. локална и глобална ефективност.

Пространство на решенията. Могат да се направят експерименти и за разглеждане на пространството на храстовидните решения $\mathcal{A}$ вместо това на ляворекурсивните $\mathcal{L}$.

Хибридизиране. Интересно би било да се създаде хибриден оптимизатор на базата на генетичния, който да използва определени евристики или да бъде комбиниран с рандомизирани техники за локално търсене. Предвид голямото бързодействие на някои рандомизирани методи за локални подобрения като симулираното изтравване и двуфазовата оптимизация, интерес би представлявало интегрирането на тези алгоритми в генетичния оптимизатор – например след генерирането на нулево поколение от случайни индивиди, всеки индивид може да се замени с локално оптималния в своята околност, т.е. може да се извърши търсене на локален оптимум в околност на всяка случайно избрана точка от пространството на решенията. Това би подобрило качеството на началната популация и вероятно би довело до по-бърза сходимост към достатъчно добро решение. Освен при генерирането на началната популация, рандомизираните техники за локално търсене могат да се прилагат и на по-късни етапи от оптимизацията – например след всяка селекция или след някои селекции по определен критерий.

Експерименти. Може би най-много възможности за бъдеща работа има по отношение на експериментите. Поради обективните трудности при тестването и сравняването на алгоритми за оптимизация, качествени статистически достоверни експерименти са проведени малко не само в този труд, но и като цяло в изследователската област. Добре би било да се проведат сравнителни експерименти с класически оптимизационни алгоритми върху реална БД с различни съдържания. Голям интерес представлява изследването на връзката между формата на заявката и формата на ценовата функция със свойствата на породеното пространство от решения, както и зависимостите между стойностите на параметрите на алгоритъма и представянето му в даденото пространство.

АВТОРСКА СПРАВКА ЗА ПРИНОСИТЕ В ДИСЕРТАЦИОННИЯ ТРУД

Научни приноси

- Направен е обзор на проблемната област - принципи и методи за обработка и оптимизирането на заявки към БД, както и исторически обзор на съществуващите системи за оптимизиране на заявки
- Направен е обзор и класификация на съществуващите стратегии и алгоритми за подреждане на съединения и са анализирани техните свойства, ограничения и приложимост. Доколкото ни е известно, това е най-пълната класификация на този клас алгоритми публикувана до момента
- Предложен е адаптивен генетичен алгоритъм, позволяващ оптимизирането на заявки с много голям брой съединения за полиномиално време, без налагане на ограничения върху формата на ценовата функция. Алгоритъмът използва оригинален оператор за селекция - вероятностна селекция с динамична популация - показал резултати, превъзхождащи по скорост и качество на намираните решения алтернативните алгоритми предложени в

литературата. Операторът е с линейна (по размера на популацията) сложност, срещу квадратична сложност на класическата Елитарна селекция. Предложеният генетичен алгоритъм превъзхожда по резултати и два класически рандомизирани алгоритми. Доколкото ни е известно, това е първият *адаптивен* генетичен алгоритъм, предложен за задачата за намиране на оптимална подредба на съединения.

- Получени и анализирани са експериментални резултати от сравнението на класически и оригинален алгоритъм за селекция, както и експерименти за скорост и за качество на намираните решения с различни параметри. Конкретизирани са стойности на параметрите на алгоритъма, които максимизират качеството на намираните решения за типични заявки

Научно-приложни приноси

- Конкретизирани са кодирането на решенията и генетичните оператори за мутация и кръстосване. При този избор на операторите и използвайки оригиналния оператор за селекция, генетичният алгоритъм е доказано сходящ
- Проектирана е библиотека за разработване на генетични оптимизатори. Модулната архитектура, спазването на обектноориентираната парадигма и платформената независимост позволяват бъдещо разширение на библиотеката и лесно интегриране в по-цялостни системи.
- Проектирана е разширяема система за оптимизиране на заявки, използваща предложения оригинален алгоритъм

Приложни приноси

- Реализирана е библиотека на ANSI C++ за разработване на генетични оптимизатори, с която могат да се реализират оптимизационни модули на СУБД с различна архитектура. Постигната е независимост на оптимизиращите модули от конкретната дефиниция и представяне на оптимизационната задача. Осигурена е платформена независимост - библиотеката е компилирана и тествана на различни компютърни архитектури и под различни операционни системи
- Реализиран е оптимизатор на заявки, способен да обработва заявки с много голям брой съединения в реално време
- Създадена е система за автоматизиране на експерименти с генетични и рандомизирани оптимизационни алгоритми. Тя включва инструменти за генериране на случайни речници на данни, случайни релации с много голям брой съединения и случайни заявки над тези релации

Във връзка с провеждането на процедура за придобиване на образователната и научна степен *Доктор* в Софийски университет "Св. Климент Охридски" и защита на представения от мен дисертационен труд, декларирам, че резултатите и приносите на проведеното дисертационно изследване, представени в дисертационния труд на тема "*Оптимизация на SQL заявки чрез генетичен алгоритъм*", са оригинални и не са заимствани от изследвания и публикации, в които нямам участие.

ПУБЛИКАЦИИ

Основните резултати от настоящия труд са публикувани в рецензирани издания и са докладвани на следните научни форуми:

[1A] Vellev St. *Adaptive genetic optimization algorithm for the join ordering problem*. In Proc. of the 34th Spring Conference of the Union of Bulgarian Mathematicians, pp. 213-216, 2005.

Работата е докладвана на 34-тата Пролетна конференция на Съюза на математиците в България, Боровец, България, 2005. Статията представя част от резултатите от Глава 5.

[2A] Vellev St. *An adaptive genetic algorithm with dynamic population size for optimizing join queries*. In IBS Information Science and Computing, vol. 2 (Advanced Research in Artificial Intelligence), ISSN 1313-0455, pp. 82-88, 2008.

Работата е докладвана на международната конференция „Intelligent Information and Engineering Systems" INFOS 2008, в рамките на ITA 2008 XI-th Joint International Scientific Events on Informatics, Варна, България, 2008. Статията представя основните резултати от Глава 5 и част от резултатите в Глава 7.

[3A] Vellev St. *Architecture of a cost-based relational database query optimizer*. In Proc. of the UNITECH'08 International Scientific Conference, vol. 1, pp. 318-323, 2008.

Работата е докладвана на международната научна конференция UNITECH'08, Габрово, България, 2008. Статията представя основните резултати от Глава 6.

[4A] Vellev St. Review of algorithms for the join ordering problem in database query optimization. In Information Technologies and Control, Year VII, vol. 1, ISSN 1312-2622, 2009.

Статията представя основните резултати от Глава 3.

ЦИТИРАНИЯ

Към настоящия момент ни е известно следното цитиране (на публикацията [2A]):

Figueira C., Soares R., Lobato F., Steffen Jr V. *A Comparative Study of Gibbs Free Energy Minimization in a Real System Using Heuristic Methods*. In Proc. of the 10th International Symposium on Process Systems Engineering, pp. 1059-1064, Brazil, 2009

Figueira et al. прилагат предложения от нас алгоритъм за адаптивно изменение на размера на популацията към известна оптимизационна задача от областта на термодинамиката/физикохимията. Анализирайки представянето на алгоритъма спрямо

класически с фиксиран размер на популацията, авторите стигат до същия извод като в нашата работа: динамичният размер на популацията влияе положително върху сходимостта на оптимизационния алгоритъм и върху качеството на намираните решения.

БИБЛИОГРАФИЯ

[1] Atzeni P., Ceri S., Paraboschi S., Torlone R. *Database Systems. Concepts, languages & architectures*. The McGraw-Hill Companies, London, 1999.

[2] Chaudhuri, S. *An overview of query optimization in relational systems*. In Proc. of ACM PODS, 1998.

[3] Fergas L. *A new heuristic for optimizing large join queries*. The University of Texas at Arlington, 1998.

[4] Graefe G. *The Cascades Framework for Query Optimization.* In IEEE Data Engineering Bulletin, pp 19-29, 1995.

[5] Graefe G., DeWitt D. *The EXODUS optimizer generator.* In Proc. of ACM SIGMOD Conference on the Management of Data, 1987.

[6] Graefe G., McKenna W. *The Volcano Optimizer Generator: Extensibility and Efficient Search.* In Proc. of the Ninth International Conference on Data Engineering, 1993.

[7] Hart W. *A theoretical comparison of evolutionary algorithms and simulated annealing*. Sandia National Laboratories. In Proc. of the 5th Annual Conference of Evolutionary Programming, 1996.

[8] Haynes T. *A comparison of random search versus genetic programming as engines for collective adaptation*. 1997.

[9] Ibaraki T., Kameda, T. *On the optimal nesting order for computing N-relational joins*. In Proc. of ACM, 1984.

[10] Ioannidis Y. *Query optimization*. In The computer science and engineering handbook, pp 1038-1057, 1997.

[11] Ioannidis Y., Kang Y. *Randomized algorithms for optimizing large join queries*. In Proc. of ACM, 1990.

[12] Ioannidis Y., Kang Y. *Left-deep vs. Bushy trees: an analysis of strategy spaces and its implications for query optimization*. In Proc. of ACM SIGMOD Conference on the Management of Data, 1991.

[13] König-Ries, B., Helmer S., Moerkotte G. *An experimental study on the complexity of left-deep join ordering problems for cyclic queries*. In Technical Report 95-4 of the RWTH Aachen, Germany, 1995.

[14] Kooi R. *The optimization of queries in relational databases*. PhD thesis, Case Western Reserve University, 1980.

[15] Koza J. *Genetic evolution and co-evolution of computer programs*. In Proc. of 2nd Conference on Artificial Life, 1990.

[16] Krishnamurthy R., Boral H., Zaniolo C. *Optimization of nonrecursive queries*. In Proc. of the 12th International VLDB conference, Kyoto, Japan, 1986.

[17] Park K., Carter B. *On the effectiveness of genetic search in combinatorial optimization. In Selected Areas in Cryptography*, 1995.

[18] Pirahesh H., Hellerstein J., Hasan W. Extensible/rule based query rewrite optimization in Starburst. In Proc. of ACM SIGMOD Conference on the Management of Data, 1992.
[19] Riondato M., Akdere M., Çetintemel U., Zdonik S., Upfal E. *The VC-Dimension of SQL Queries and Selectivity Estimation Through Sampling*. In Proc. of the ECML/PKDD, 2011.
[20] Rudolph G. *Convergence Analysis of Canonical Genetic Algorithms*. In: IEEE Transactions on Neural Networks, 1994.
[21] Scheufele W., Moerkotte G. *On the complexity of generating optimal plans with cross products*. In Proc. of ACM PODS, 1997.
[22] Steinbrunn M., Moerkotte G., Kemper A. *Optimizing join orders*. Technical report MIP-9307, Universität Passau,1993.
[23] Steinbrunn M., Moerkotte G., Kemper A. *Heuristic and randomized optimization for the join ordering problem*. In Proc. of VLDB, Springer-Verlag, 1997.
[24] Swami A. *Optimization of large join queries: combining heuristics and combinatorial techniques*. In Proc. of ACM, 1989.
[25] Swami A., Gupta A. *Optimization of large join queries*. In Proc. of ACM SIGMOD, 1988.
[26] Swami A., Iyer B. *A polynomial time algorithm for optimizing join queries*. In Proc. of IEEE International Conference on Data Engineering, Vienna, Austria, 1993.
[27] Waas F., Pellenkoft A. *Probabilistic bottom-up join ordering selection – breaking the curse of NP-completeness*. In Report INS-R9906, ISSN 1386-3681, p.15-32, Centrum voor Wiskunde en Informatica, 1999. [62] Xu Y. *Efficiency in the Columbia database query optimizer*. MSc Thesis, Portland State University, 1998.
[28] Xu Y. *Efficiency in the Columbia database query optimizer*. MSc Thesis, Portland State University, 1998.
[29] Yao X. Global Optimization by evolutionary algorithms. The University of New South Wales. 1997.

БЛАГОДАРНОСТИ

Много хора пряко или косвено ми помогнаха да извървя пътя дотук. На всеки един от тях дължа благодарност.

Бих започнал с моите преподаватели, от които имах шанса да се уча и да работим заедно вече почти петнадесет години.

Благодаря на Магдалина Тодорова за това, че повярва в мен и ми даде шанса още като третокурсник да водя упражнения на студенти по нейни курсове по функционално и логическо програмиране. Магда е човекът, който ме запали по декларативните езици, на нея дължа и запознаването си с генетичното програмиране. За мен тя винаги е била пример – и като преподавател, и като човек.

Вторият човек, който ми подаде ръка в университета, беше колоритният Димитър Шишков (днес покойник), който ме включи в екипа си работещ по сборник по структури от данни, а по-късно ми предложи и работа със студенти в курса му по изкуствен интелект. Благодарен съм му за доверието, за енциклопедичните му знания, от които можех да черпя, за десетките интересни идеи, които обсъждахме понякога до полунощ в университета и дома му, а също и за перфекционизма му, от който често страдах, но от който и се учех. На тези двама мои преподаватели дължа сбъдването на мечтата си да работя със студенти, те бяха и тези, които настоятелно, почти насила, ме убедиха да започна докторантура след дипломирането ми.

Благодаря на Калоян Калоянов за това, че ме насочи към изключително интересния проблем за оптимизиране на заявки към бази данни и за това че стана мой дипломен ръководител. Той подкрепи идеята ми да приложа генетичен оптимизационен алгоритъм към задачата, а в многобройните ни срещи ме насочваше в работата ми, споделяше идеи и ми даваше полезни съвети. Дипломната ми работа всъщност е основата, от която се разви настоящата дисертация.

Благодаря на колегата ми от университета и мой верен приятел, Симо Симов, за оригиналните му идеи и критичните му коментари към моите идеи. С него обсъждахме алгоритмичните проблеми, с които се сблъсквах по време на работата си, а се е случвало и да ме „спасява“ с изключителните си знания и практически опит по C++. На друг мой добър приятел, Николай Георгиев, благодаря за помощта по графичното оформяне на работата и създаването на част от диаграмите.

Специална благодарност дължа на моя научен ръководител, Владимир Димитров. В основата на всичко, което знам за базите от данни, стоят нещата, които съм научил от него и от Калоян Калоянов. Работейки в известна изолация като докторант на самостоятелна подготовка, на научния си ръководител дължа основната част от връзките си с академичната общност – участието ми в научни конференции и успешните ми публикации дължа в голяма степен на неговите насоки. Благодаря му за свободата, която ми даде, за изключителното му търпение и за всички случаи, в които ми помагаше да се върна „в правия път“, както по съществото на дисертацията, така и по методологията и организацията на работата ми. Благодаря му, че през всички тези години не загуби вярата си в мен и работата ми, дори когато аз я губех.

На всички вас – сърдечно ви благодаря за подкрепата!

КРАТКА БИОГРАФИЯ

Стоян Велев е роден през 1976 г. в гр. София. През 1995 г. завършва със златен медал Първа английска езикова гимназия – София. Същата година продължава образованието си във Факултета по Математика и Информатика (ФМИ) на Софийския университет (СУ) „Св. Климент Охридски", специалност Информатика. Дипломира се като магистър през 2001 г. по специалност Информационни системи, с магистърска теза в областта на обработката на заявки към бази данни. С дипломирането си е зачислен в свободна форма на докторантура към катедра Компютърна информатика.

От 1998 г. като студент и по-късно докторант Стоян Велев работи като хоноруван преподавател във ФМИ и води упражнения към курсовете по Функционално и логическо програмиране, Увод в програмирането и Изкуствен интелект, а през 2003 г. води лекции по Структури от данни. През този период е преподавал в летни курсове за следдипломна квалификация на учители по информатика, водил е упражнения по обектоориентирано програмиране на студенти в Стопанския факултет на СУ, курсове по информатика за ученици към Института по Математика и Информатика на Българската Академия на Науките и др.

Паралелно с академичните си занимания, Стоян Велев работи като софтуерен инженер, а по-късно и като ръководител на екипи и проекти в няколко интернационални софтуерни компании. Стоян Велев е съавтор на четири софтуерни патента в областта на системите за асинхронна комуникация чрез размяна на съобщения (*message-oriented middleware*).

Научните интереси на Стоян Велев са в областта на системите за управление на бази от данни и изкуствения интелект – обработка и оптимизация на заявки, декларативни езици, компютърна лингвистика, извличане на информация, когнитивна наука, еволюционни алгоритми и др.