EXECUTION PLAN OPTIMIZATION TECHNIQUES

Július Štroffek
Database Sustaining Engineer
Sun Microsystems, Inc.

Tomáš Kovařík
Faculty of Mathematics and Physics,
Charles University, Prague

PostgreSQL Conference, 2007

Agenda

- Motivation
  - Questions
  - Goals
- Definition of The Problem
  - Join Graph
  - Optimizer and Execution Plan
  - Rule Based Optimization
  - Cost Based Optimization

Agenda II.

Deterministic Algorithms for Searching the Space

- Dynamic Programming
- Dijkstra's algorithm
- A* Search Algorithm

Non-Deterministic Algorithms for Searching the Space

- Random Walk
- Nearest Neighbor
- Simulated Annealing
- Hill-Climbing

Agenda III.

Demo

Algorithms Already Implemented for Query Optimization

- Genetic Algorithm
- Simulated Annealing
- Iterative Improvement
- Two Phase Optimization
- Toured Simulated Annealing

Experimental Performance Comparison

Conclusions

Motivation

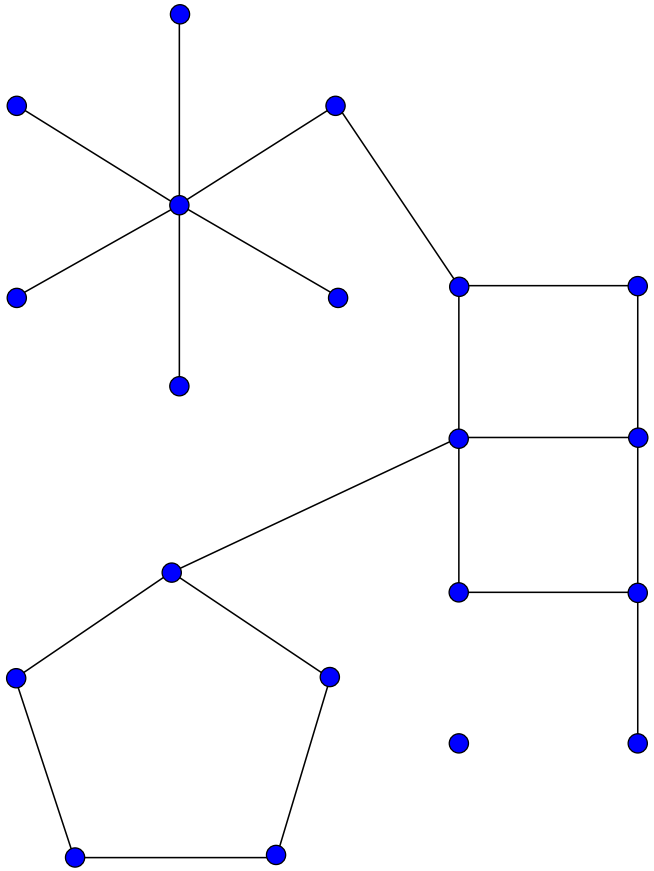
Questions which come to mind looking at PostgreSQL optimizer implementation:

- Why to use genetic algorithm?
- Does this implementation behave well?
- What about other ”Soft Computing” methods?
- Are they worse, the same or better?
- What is the optimizer’s bottle-neck?

Goals

- Present ideas how PostgreSQL optimizer might be improved.
- Receive feedback.
- Discuss the ideas within the community.
- Implement some of those ideas.

Join Graph



Definition

- A node represents a relation
- An edge indicates that a condition was used to restrict records of the connected relations
- Arrows might be used to reflect outer joins

Optimizer and Execution Plan

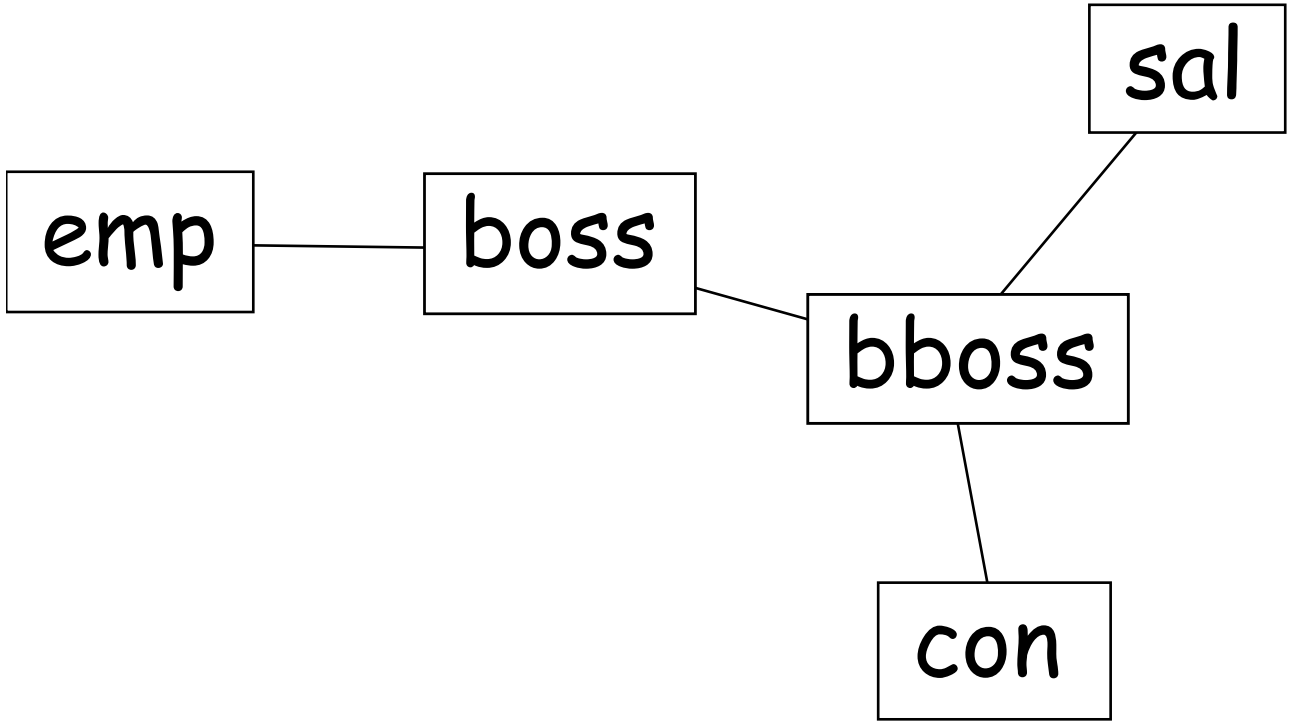
Lets have a query like this

```
select
  bboss.EmpId, sal.Salary,
  sal.Currency, con.Value
from
  Employees as emp,
  Employees as boss,
  Employees as bboss,
  Salaries as sal,
  Contacts as con
where
  emp.LastName='Stroffek'
  and emp.ReportsTo = boss.EmpId
  and boss.ReportsTo = bboss.EmpId
  and bboss.EmpId = sal.EmpId
  and con.EmpId = bboss.EmpId
  and con.Type = 'CELL'
```

How to process the query?

- There are 5 relations to be joined.
- If we do not allow product joins we have 10 options.
- With product joins allowed we have 60 options.

Join Graph for the Example Query



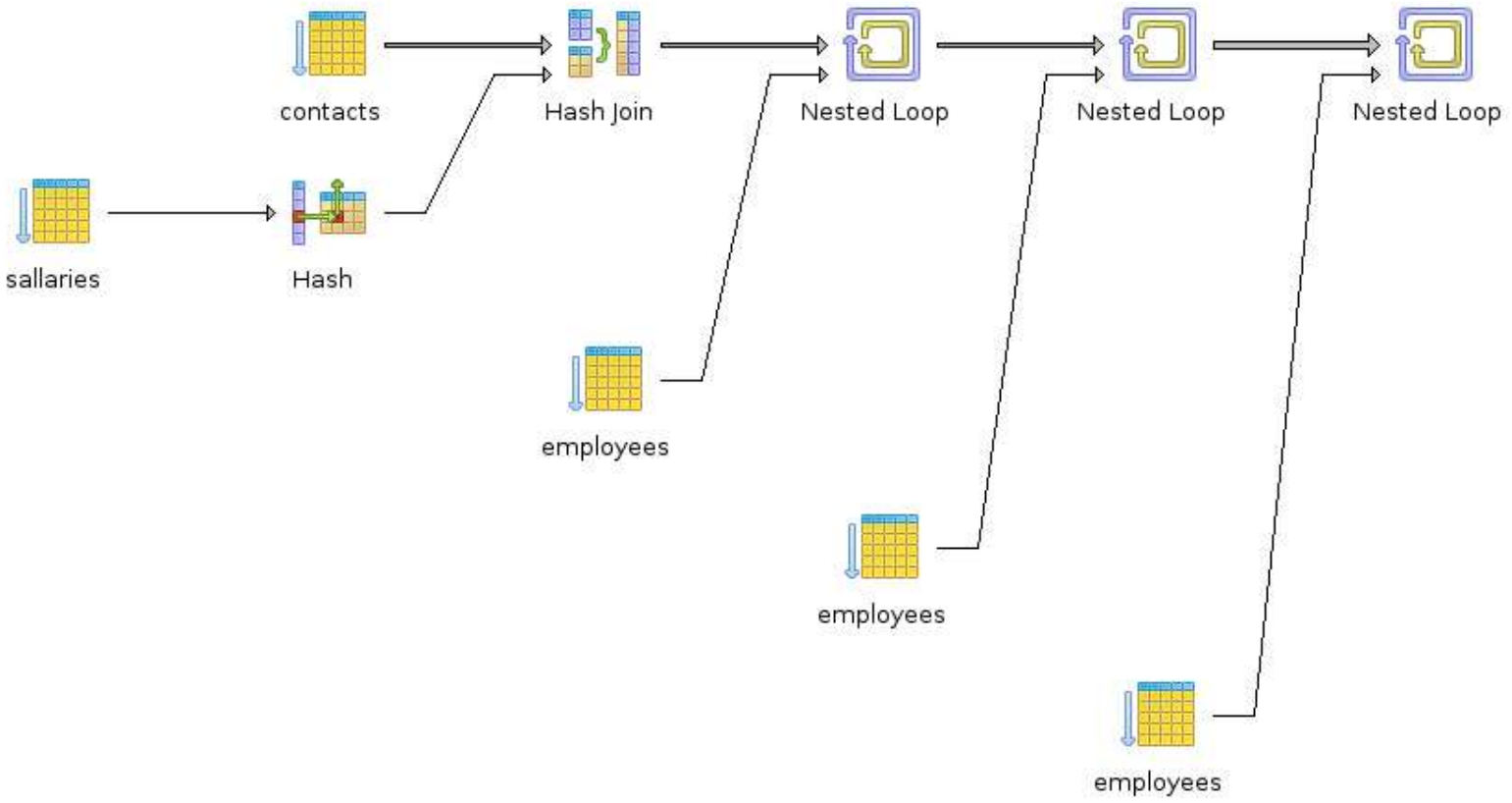
Execution Plan

Left-Deep Tree Example

```
Hash Join  (cost=13.84..17.81 rows=1 width=151)
  Hash Cond: (”outer”.empid = ”inner”.reportsto)
  → Seq Scan on employees bboss  (cost=0.00..3.64 rows=64 width=4)
  → Hash  (cost=13.84..13.84 rows=1 width=159)
      → Hash Join  (cost=10.71..13.84 rows=1 width=159)
          Hash Cond: (”outer”.empid = ”inner”.reportsto)
          → Seq Scan on contacts con  (cost=0.00..2.80 rows=64 width=136)
              Filter: (”type” = ’CELL’::bpchar)
          → Hash  (cost=10.71..10.71 rows=1 width=23)
              → Hash Join  (cost=7.78..10.71 rows=1 width=23)
                  Hash Cond: (”outer”.empid = ”inner”.reportsto)
                  → Seq Scan on sallaries sal  (cost=0.00..2.28 rows=128 width=19)
                  → Hash  (cost=7.77..7.77 rows=1 width=4)
                      → Hash Join  (cost=3.80..7.77 rows=1 width=4)
                          Hash Cond: (”outer”.empid = ”inner”.reportsto)
                          → Seq Scan on employees boss  (cost=0.00..3.64 rows=64 width=8)
                          → Hash  (cost=3.80..3.80 rows=1 width=4)
                              → Seq Scan on employees emp  (cost=0.00..3.80 rows=1 width=4)
                                  Filter: (lastname = ’Stroffek’::bpchar)
```

Execution Plan

Left-Deep Tree Example - pgAdmin



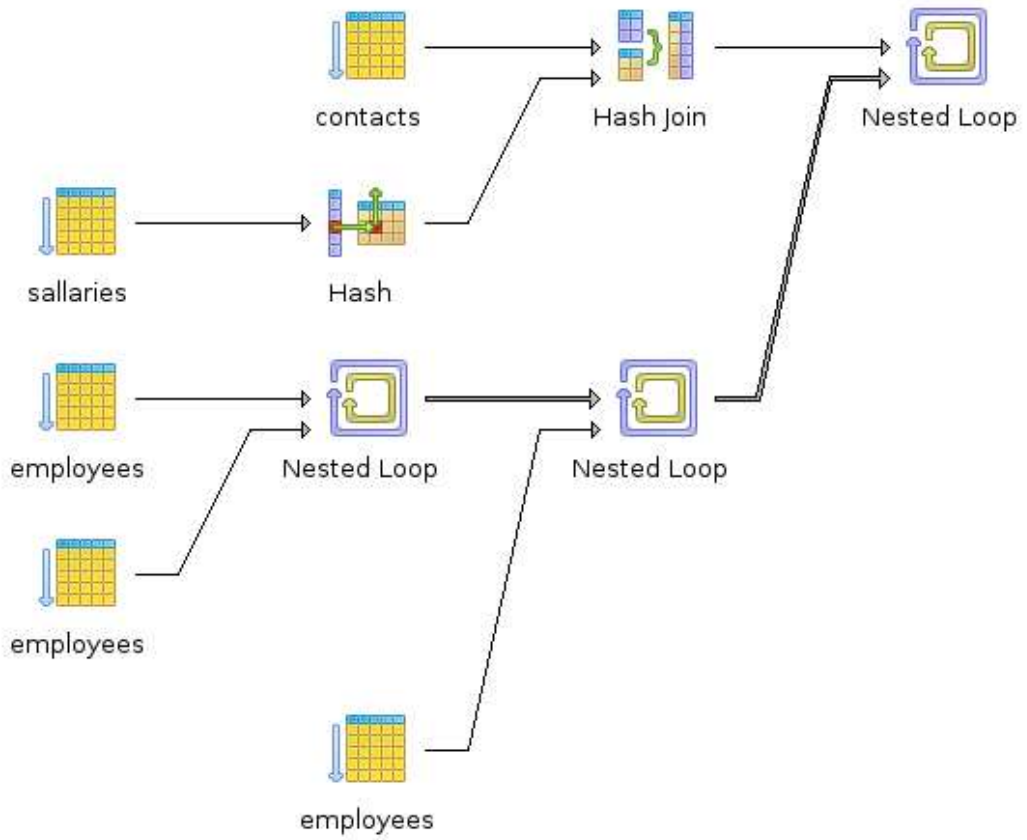
Execution Plan

Bushy Tree Example

```
Nested Loop  (cost=1.00..13.17 rows=1 width=151)
  Join Filter: (”inner”.empid = ”outer”.empid)
  →  Hash Join  (cost=1.00..4.13 rows=1 width=155)
       Hash Cond: (”outer”.empid = ”inner”.empid)
       →  Seq Scan on contacts con  (cost=0.00..2.80 rows=64 width=136)
            Filter: (”type” = ’CELL’::bpchar)
       →  Hash  (cost=1.00..1.00 rows=1 width=19)
            →  Seq Scan on sallaries sal  (cost=0.00..1.00 rows=1 width=19)
  →  Nested Loop  (cost=0.00..9.02 rows=1 width=8)
       Join Filter: (”outer”.reportsto = ”inner”.empid)
       →  Nested Loop  (cost=0.00..6.01 rows=1 width=4)
            Join Filter: (”outer”.reportsto = ”inner”.empid)
            →  Seq Scan on employees emp  (cost=0.00..3.00 rows=1 width=4)
                 Filter: (lastname = ’Stroffek’::bpchar)
            →  Seq Scan on employees boss  (cost=0.00..3.00 rows=1 width=8)
       →  Seq Scan on employees bboss  (cost=0.00..3.00 rows=1 width=4)
```

Execution Plan

Bushy Tree Example - pgAdmin



Rule Based Optimization

- Deterministic - produce the same result at every call for the same query.
- Mostly produces a plan based on a join graph.
- Does not search through the large space.
- Thus it is relatively ”fast”.
- Does not find the best plan in most cases.
- Many approaches how to find the plan based on a use case.

Cost Based Optimization

- Evaluate every execution plan by it's cost.
- Using relation statistics to calculate estimates and cost.
- Exponential search space size.
- Have to use a proper method for searching the space.
- Finds the best plan for a small number of relations.
- Not possible to find the best plan for too many relations.

Deterministic Algorithms

Dynamic Programming

- Checks all possible orders of relations.
- Applicable only for small number of relations.
- Pioneered by IBM's System R database project.
- Iteratively creates and computes costs for every combination of relations. Start with an access to an every single relation. Later, compute the cost of joins of every two relations, afterwards do the same for more relations.

Dynamic Programming

Example

Four relations to be joined – A, B, C and D.

{A} {B} {C} {D}

{AB} {AC} {AD} {BC} {BD} {CD}

{ABC} {ABD} {ACD} {BCD}

{ABCD}

Dijkstra's algorithm

Description

A graph algorithm which finds the shortest path from a source node to any destination node in a weighted oriented graph.

- A graph has to have a non-negative weight function on edges.
- The weight of the path is the sum of weights on edges on the path.

Question

How can we manage that the path will go through all the nodes/relations in a join graph?

Dijkstra's algorithm

Description II

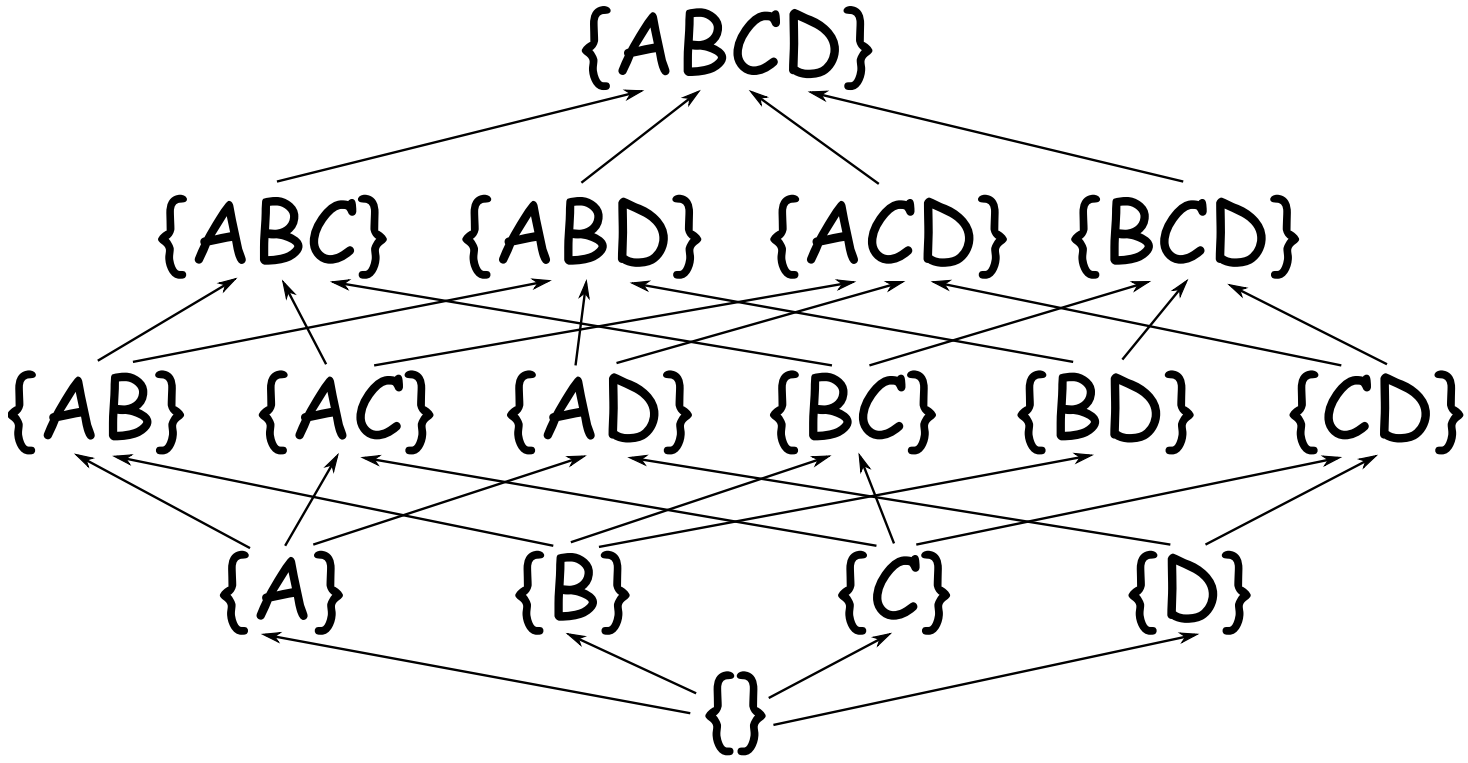
We will search for the path in a join graph directly but we will search for a shortest path in a different graph.

- Nodes – all possible subsets of relations to be joined.
- Edges – directed; connect every node to all other nodes where it is possible to "go" by performing exactly one join.
- Being in a node on the path means that the corresponding relations are joined.
- Going through the edge means that we are performing a join.

Dijkstra's algorithm

Graph Example

We are joining four relations - A, B, C and D.



Dijkstra's algorithm

Description III

For each node V we will maintain

- The lowest cost $c(V)$ of already found path to achieve the node.
- The state of the node – open/closed.

We will also maintain a set of nodes $\mathcal{M}$ which will hold open nodes.

Dijkstra's algorithm

Steps

1. Put a starting node '$\{\}$' to the set $\mathcal{M}$ with a cost '0'.
2. Remove a node V with minimal cost $c(V)$ from the set $\mathcal{M}$ and mark it as closed.
3. For all outgoing edges of the node V calculate the cost of a path going to every destination node through the node V.
4. Put all the open nodes on outgoing edges to the set $\mathcal{M}$ if are not already there.
5. If the set $\mathcal{M}$ is not empty and we have not picked up the destination node yet go back to step 2.

A* Search Algorithm

- Modification of Dijkstra's algorithm.
- Introduces a heuristic function $h(V)$ which calculates the lower bound of the path cost from the node V to the destination node.
- We will change the 2nd step and we will remove the node $c(V)$ with minimal $c(V) + h(V)$ and mark it as closed.
- If we have "better" heuristic function we do not have to go through the whole space.

A* Search Algorithm

Heuristic Function

If we still would like to mark nodes as closed the heuristic function has to satisfy the following condition

$$h(X) - h(Y) < c(Y) - c(X) \tag{1}$$

If the condition is broken we can not mark nodes as closed and we have to process some of the nodes multiple times.

It is possible to prove that if the heuristic function is the lower bound of the real cost the algorithm will find the best solution in both cases.

A* Search Algorithm

Possible Heuristic Functions

- The number of tuples in the current node.
  - Reduces paths where too many tuples are produced.
  - Applicable only in materialization nodes.
- The number of tuples that need to be fetched from the remaining relations.
  - A realistic lower bound satisfying the condition allowing the marking of nodes as closed.
  - Might be extended to get a lower bound of the cost of remaining joins.
- Any other approximation?

Non-Deterministic Algorithms

Random Walk

Randomly goes through the search space

1. Set the MAX_VALUE as a cost for the best known path.
2. Choose a random path.
3. If the cost of the chosen path is better than the currently best known remember the path as the best.
4. Go back to step 2.

Very naive, not very usable in practise but can be used for comparison with other algorithms.

Nearest Neighbor

1. Lets start in the first node.
2. Chose an edge with the lowest cost going to an unreached node.
3. Repeat the previous step until there is any unvisited node.
4. If the path found has lower cost than the best path already known remember the path.
5. Choose the next node as a starting one and continue with 2nd step.

It is possible to check all the paths of the constant length instead of only edges.

Nearest Neighbor for Bushy Trees

1. Build a set of nodes with an access to every relation.
2. Choose the cheapest join from all the combinations of possible joins of any two relations from the set.
3. Add the result node to the set of relations to be joined.
4. Remove the originating relations from the set.
5. Go back to 2nd step if we still have more than one relation.

It is also possible to look forward for more joins at once.

Simulated Annealing

It is an analogy with physics – a cooling process of a crystal.

A crystal has lots of energy and small random changes are happening in its structure. These small changes are more stable if the crystal emits the energy afterwards.

An execution plan has a high cost at the begining. A small random change is generated in every iteration of the process. Depending upon the change of the cost the change the probability whether the change is preserved or discarded is computed. The change is kept with the computed probability.

Hill-Climbing

1. Start with a random path.
2. Check all the neighbor paths and find the one with the lowest cost.
3. If the cost of the best neighboring path is lower than the actual one change the actual one to thatneighborg.
4. Go back to 2nd step if we found a better path in the previous step.

Demo

Travelling Salesman Problem

A travelling salesman has *N* cities which he needs to visit. He would like to find the shortest path going through all the cities.

- Similar to the execution plan optimization problem.
- The evaluation function has different properties.
  - The evaluation of an edge between two cities does not depend on the cities already visited.
  - Different behavior of heuristic algorithms.

Implemented Algorithms

Implemented Algorithms

- Number of algorithms for optimization of large join queries was described and studied, e.g. in:
  - M. Steinbrunn, G. Moerkotte, A. Kemper. Heuristic and Randomized Optimization for the Join Ordering Problem. In VLDB Journal, 6(3), 1997
  - Ioannidis, Y. E. and Kang, Y. 1990. Randomized algorithms for optimizing large join queries. In Proceedings of the 1990 ACM SIGMOD international Conference on Management of Data
- Performance evaluation and comparison of these algorithms is available
- Implementation is often experimental and only for purposes of the given research

Studied algorithms

We will look at the following algorithms in more detail

- Genetic Algorithm
- Simulated Annealing
- Iterative Improvement
- Two Phase Optimization
- Toured Simulated Annealing

Genetic Algorithm

- Well known to the PostgreSQL community - GEQO - GEnetic Query Optimizer
- Fundamentally different approach designed to simulate biological evolution
- Works always on a set of solutions called population
- Solutions are represented as character strings by appropriate encoding.
  - In our case query processing trees
- Quality, or “fitness” of the solution is measured by an objective function
  - Evaluation cost of processing tree, has to be minimized

Genetic Algorithm

Steps

- “’Zero” population of random character strings is generated
- Each next generation is obtained by a combination of
  - **Selection** - Selected fittest members of the population survive to the next generation
  - **Crossover** - Selected fittest members of the population are combined producing an offspring
  - **Mutation** - Certain fraction of the population is randomly altered (mutated)
- This is repeated until
  - the population has reached desired quality
  - predetermined number of populations has been generated
  - no improvement has been detected for several generations

Simulated Annealing

- Doesn't have to improve on every move
- Doesn't get "trapped" in local minimum so easily
- Exact behavior determined by parameters:
  - starting temperature (e.g. function of node cost)
  - temperature reduction (e.g. 90% of the old value)
  - stopping condition (e.g. no improvement in certain number of temperature reductions)
- Performance of the algorithm is dependent on the values of these parameters

Simulated Annealing

Determining neighbor states

- Join method choice
- Left-deep processing trees solution space represented by an ordered list of relations to be joined
  - Swap – exchange position of two relations in the list
  - 3Cycle – Cyclic rotation of three relations in the list
- Complete solution space including bushy trees
  - Join commutativity $A \bowtie B \rightarrow B \bowtie A$
  - Join associativity $(A \bowtie B) \bowtie C \leftrightarrow A \bowtie (B \bowtie C)$
  - Left join exchange $(A \bowtie B) \bowtie C \leftrightarrow (A \bowtie C) \bowtie B$
  - Right join exchange $A \bowtie (B \bowtie C) \leftrightarrow B \bowtie (A \bowtie C)$

Iterative Improvement

- Strategy similar to Hill climbing algorithm
- Overcomes the problem of reaching local minimum
- Steps of the algorithm
  - Select random starting point
  - Choose random neighbor, if the cost is lower than the current node, carry out the move
  - Repeat step 2 until local minimum is reached
  - Repeat all steps until stopping condition is met
  - Return the local minimum with the lowest cost
- Stopping condition can be processing fixed number of starting points or reaching the time limit

Two Phase Optimization

- Combination of SA and II, uses advantages of both
  - Rapid local minimum discovery
  - Search of the neighborhood even with uphill moves
- Steps of the algorithm
  - Predefined number of starting points selected randomly
  - Local minima sought using Iterative improvement
  - Lowest of these minima is used as a starting point of simulated annealing
- Initial temperature of the SA is lower, since only small proximity of the minimum needs to be covered

Toured Simulated Annealing

- Simulated annealing algorithm is run several times with different starting points
- Starting points are generated using some deterministic algorithm with reasonable heuristic
- The initial temperature is much lower than with the generic simulated annealing algorithm
- Benefits over plain simulated annealing
  - Covering different parts of the search space
  - Reduced running time

Experimental Performance Comparison

What to consider

- Evaluation plan quality vs. running time
- Different join graph types
  - Chain
  - Star
  - Cycle
  - Grid
- Left-deep tree optimization vs. Bushy tree optimization
- Algorithms involved
  - Genetic algorithm (Bushy genetic algorithm)
  - Simulated annealing, Iterative improvement, Two phase optimization

Experimental Performance Comparison

Results

- Bushy tree optimization yields better results especially for chain and cycle join graphs.
- If solution quality is preferred, 2PO achieves better results (although slightly sacrificing the running time).
- Iterative improvement and Genetic algorithm are suitable in case the running time is more important.
- Pure Simulated annealing requires higher running time without providing better solutions.

Conclusions

Conclusions

- We have presented an overview of algorithms usable for query optimization.
- Choosing the best algorithm is difficult.
- PostgreSQL has configuration parameters for GEQO
  - Threshold
  - Population size
  - Number of populations
  - Effort based approach
- More configurable optimizer with different algorithms
  - OLAP - would prefer better plan optimization
  - OLTP - would prefer faster plan optimization

???

Július Štroffek
Julius.Stroffek@Sun.COM
julo@stroffek.net

Tomáš Kovařík
tkovarik@gmail.com