Optimizing Large Query by Simulated Annealing Algorithm Based On Graph-Based Approach

Yongheng Chen
College of Computer Science and Technology, Jilin University, Changchun, China
Email:cyh771@163.com

Wanli Zuo
Key Laboratory of Symbol Computation and Knowledge Engineering of the Ministry of Education
College of Computer Science and Technology, Jilin University, Changchun, China
Email:wanli@jlu.edu.cn

Fenglin He and Kerui Chen
College of Computer Science and Technology, Jilin University, Changchun, China

***Abstract*—In the relational database setting today, large queries containing many joins are becoming increasingly common. In general the ordering of join-operations is quite sensitive and has a devastatingly negative effect on the efficiency of the DBMS. Scheufele and Moerkotte proved that join-ordering is NP-complete in the general case [1]. The dynamic programming algorithm has a worst case running time, thus for queries with more than 10 joins, it becomes infeasible. To resolve this problem, random strategies are used. Simulated annealing is an intelligent algorithm of developing very fast. In this paper, we introduce the traditional simulated annealing algorithm through discussing its theory and process, analyzes its shortcoming in detail, simple describe influence of key parameters to simulated annealing algorithm and provided feasible improvement. Especially, in order to avoid the deficiency resulted by the neighbors of state, and make the query optimization support complex non-inner join, the improved algorithm gives a semantics expression of query and a method of constructing the connected join pairs. Experimental results show that the improved algorithm outperforms the original algorithm in terms of both output quality and running time.**

***Index Terms*—simulated annealing, Query optimization, Join-Order, Randomized Algorithms**

I. INTRODUCTION

The key to the success of a Database Management System, especially of one based on the relational model, is the effectiveness of the query optimization module of the system. The input to this module is some internal representation of an ad-hoc query. Its purpose is to select the most efficient algorithm (access plan) in order to access the relevant data and answer the query. Query optimization is an expensive process, primarily because the number of alternative access plans for a query grows at least exponentially with the number of relations participating in the query. The application of several useful heuristics eliminates some alternatives that are likely to be suboptimal, but it does not change the combinatorial nature of the problem. Future database systems will need to optimize queries of much higher complexity than current ones. This increase in complexity may be caused by an increase in the number of relations in a query, by an increase in the number of relations in a query, by an increase in the number of queries that are optimized collectively [2].

The heuristically pruning, almost exhaustive search algorithms used by current optimizers are inadequate for queries of the expected complexity [3, 4]. To resolve this problem, the need to develop random strategies optimization algorithms becomes apparent. The transformational approach characterizes this kind of strategies. Several rules of transformation were proposed [5] where the validity depends on the nature of the considered search space [6].The random strategies start generally with an initial execution plan which is iteratively improved by the application of a set of transformation rules. The start plan(s) can be obtained through an enumerative strategy like Augmented Heuristics. Two techniques were abundantly already studied and compared: the Iterative Improvement and the Simulated Annealing [7].

The performance evaluation of these strategies is very hard because of strong influence, at the same time, of random parameters and factors. The main difficulty lies in the choice of these parameters. Indeed, the quality of execution and the optimization cost depend on the quality of choice [9]. After the tuning of the parameters, the comparison of the algorithms will allow to determine the most efficient random algorithm for the optimization problem of complex queries.

Randomized algorithms have been successfully applied to various combinatorial optimization problems. In the literature there are many alternative approaches to the join ordering problem, Steinbrunn et al [8] present a good overview. Appro- aches such as Iterative Improvement,

Corresponding author: WanLi Zuo, Wanli@jlu.edu.cn

doi:10.4304/jsw.6.9.1655-1663

Simulated Annealing, Genetic Algorithms, Two phase optimization etc all provide efficient alternatives although producing sub-optimal solutions to the join-ordering problem for large queries. Ioanrudis and Wong [10] applied Simulated Annealing to the optimization of some recursive queries. Swami and Gupta applied both Simulated Annealing and Iterative Improvement on optimiza- tion of select-project-join queries.

In this paper, we present a probabilistic algorithm for query optimization which is suitable for large access plan spaces based on the Simulated Annealing algorithm. In order to avoid the deficiency resulted by the transformation of state, and make the query optimization support complex non-inner join, the improved algorithm gives a semantics expression of query and a method of constructing the connected join pairs. Also designed an adaptive temperature update function and set up dual-threshold to reduce amount of calculation. Finally the algorithm is increased memory function to remember current best state so as to avoid missing current optimal solution.

In next Section we introduce some technology from Combinatorial Optimization and describe different algorithms for tackling such problems. In Section 3 the specific parameters of the simulated annealing will be given. In Section 4 the limitation and improving frame of Simulated Annealing is given. Section 5 introduce related techniques the about the improved simulated annealing algorithm. Then gives the overall improved simulated annealing algorithm. Section 6 presents the results of performance evaluation. Section 7 concludes the paper.

II. COMBINATIORIAL OPTIMIZATION TECHNIQUES

Each solution to a combinatorial optimization problem can be thought of as a state in a space that includes all such solutions. Each state has a cost associated with it, which is given by some problem-specific cost function [11]. The goal of an optimization algorithm is to find a state with the globally minimum cost. Randomized algorithms usually perform random walks in the state space via a series of moves. A move is a perturbation applied to a solution to get another solution, one moves from the state represented by the former solution to the state represented by the latter solution. Two states are said to be adjacent states if one move suffices to go from one state to the other. A move is called uphill (downhill) if the cost of the source state is lower (higher) than the cost of the destination. A local minimum in the state space is a state such that its cost is lower than that of all neighboring states. A global minimum is a state which has the lowest cost among all the local minima. There can be many such local minima, but be only one global minimum. Using the above technology we describe three randomized optimization, i.e. Simulated Annealing and Iterative Improvement to query optimization and introduce the Two Phase Optimization algorithm [15].

A. *Iterative Improvement(II)*

The Iterative Improvement [12] is a variant of local search where the selection of which neighbor state to visit is performed at random. Neighbors are defined such that they can be reached by one move from the current state. In Iterative Improvement, two different type of move were defined and are applied with probability α and (1 − α) respectively. Given that the move results in a valid state, the moves are:

• Swap: Choose two relations at random and perform a swap.

• 3Cycle: Choose three relations at random <a, b, c>, rotate them to the right <c, a, b>.

Let S and minS be states and cost(S) be the cost of state S. Let neighbors(S) is the set of states reachable from S: the Iterative Improvement algorithm is described in Table I.

TABLE I.
ITERATIVE IMPROVEMENT ALGORITHM

Iterative Improvement Algorithm
1: function II
2: while not stopping-condition do
3: S ← random state.
4: while not local-optimum do
5: S′← random state in neighbors(S)
6: if cost(S′) < cost(S) then S ←S′
7: end while
8: if cost(S) < cost(minS) then minS ← S
9: end while
10: return minS. end function.

Since it would be inefficient to check all the neighbors, a Local optimum is declared if in a large sample of the neighborhood-set, no neighbor yields a better solution. Itera- tive Improvement could equally well be termed Iterated Local Search as it fits very well in the Local Search group of heuristics. Other random based heuristics mentioned in [12] such as Perturbation Walk (PW) and Quasi random Sampling (QS) also fit very well into the definition of Local Search. QS selects new states completely at random which is the same as considering all states as neighbors.

B. *Simulated Annealing(SA)*

Simulated Annealing is based on the annealing process of crystals where, simply stated, first heating and then slowly cooling a liquid will result in crystals. It differs from the methods described so far in that it allows, with a certain probability, a non-improving move to be made. This proba- bility is proportional to the Temperature of the system, which decreases over time. Ioannidis and Wong [10] implement and evaluate a Simulated Annealing system for optimization of recursive queries with satisfactory results. The genetic algorithm is shown in Table II.

TABLE II.
SIMULATED ANNEALING ALGORITHM

Simulated Annealing Algorithm
1:function SA() (
2: s = so ; T=To; minS=S
5: while not (frozen) do (
7: while not (equilibrium) do (
8: S′ = random state in neighbors(S)
9: △C = cost(S′) - cost(S)
10: if (△C < 0) then S = S′
11: if (△C > 0) then S = S′ with probability $e^{-\triangle c/T}$.
12: if cost(S) < cost(minS) then minS = S)
13:T = reduce(T)). return(minS)).

C. Two Phase Optimization(2PO)

The Two Phase Optimization [14] heuristic was developed for select-project-join query optimization. It combines Iterated Improvement with Simulated Annealing. As name suggests, 2PO consists of two phases, the first one is to run Iterative Improvement for a short period of time and using the resulting solution as the starting state for the Simulated Annealing. In the SA pass, a low initial temperature is used, limiting the search somewhat. The motivation for using the low temperature is that the cost function resembles a cup with uneven bottom. According to Ioannidis and Kang [14], the Two Phase Optimization outperforms the two original algorithms in both running time and quality.

III. THE PARAMETERS OF SIMULATED ANNEALING

When SA randomized optimization algorithms are applied to a particular problem, there are several parameters that need to be specified based on the specific characteristics of the problem. For SA, the definition of the cost function, the state space and neighbors function is necessary. This section gives several parameters' definition and analyses the limitation of SA.

A. Cost Function

The cost function that we used in this paper only accounts for the number of tuples required by each strategy.

Definition 1. For a join S=R1 join R2, the calculation of Cost and V (a, S) is following.

$$Cost(S)=\frac{|R_1||R2|}{\prod_{ci\in C}\max(V(ci,R_1),V(ci,R2))}$$

$$V(a,S)=\begin{cases} V(a, R_1),\ a\in R_1-R_2 \\ V(a, S),\ a\in R_2-R_1 \\ \min(V(a, R_1), V(a, R_2)) \\ a\in R_1 \text{ and } a\in R_2 \end{cases}$$

|R| denotes the number of tuples of R. C is the collection of public property between R1 and R2. V (a, S) denotes the different value' number of a in S. V (a, S) is similar with V (a, S).

Definition 2. The Cost of a query plan is the total cost of Sj belong the join query tree' internal nodes

$$COST=\sum_{j=1}^{n-1} Sj$$

n is the number of join-operations.

B. Neighbors Function

The neighbors of a state, which is a join processing, are determined by a set of transformation rules. Each rule is applied to one or two internal nodes of the states, replaces them by one or two new nodes, and usually leaves the rest of the nodes of the state unchanged. With A, B, and C being arbitrary join processing formulas, the set of transformation rules that we used in our study is given below

(1)Join method choice $A \bowtie_{methodi} B \rightarrow A \bowtie_{methodj} B$

(2)Join commutativity $A \bowtie B \rightarrow B \bowtie A$

(3)Join associativity $(A \bowtie B) \bowtie C \rightarrow A \bowtie (B \bowtie C)$

(4)Left join exchange $(A \bowtie B) \bowtie C \rightarrow (A \bowtie C) \bowtie B$

(5)Right join exchange $(A \bowtie B) \bowtie C \rightarrow B \bowtie (A \bowtie C)$

C. State Space

Each state in query optimization corresponds to an access plan of the query to be optimized. We reduce the goal of the query optimizer to finding the best join order, together with the best join method for each join. In this case, each strategy can be represented as a join processing tree, i.e., a tree whose leaves are base relations, internal nodes are join operators, and edges indicate the flow of data [15]. If all internal nodes of such a tree have at least one leaf as a child, then the tree is called linear. Otherwise, it is called bushy. In our study, the strategy space includes all possible join processing trees, i.e. , both linear and bushy ones

IV. THE LIMITATION AND IMPROVING FRAME OF SIMULATED ANNEALING

Simulated annealing technique during the iterative process- ing not only accepts the value making the objective function change "good" point, but also "bad" point by a certain probability. The acceptance probability gradually decreases as the temperature dropping. Because the randomness obtaining the value in the entire solution, the SA algorithm can avoid the local optimal solution and improve the reliability of the global optimal solution obtained

Although the simulated annealing algorithm accepts the existence of a limited-inferior solution and can avoid the local optimal solution and is suitable for solving the global optimiz- ation problems, there are a few points in the traditional SA algorithm where is room for improvement.

A. *The Llimitation of The Traditional Simulated Annealing*

The limitation of the traditional simulated annealing is divided to two types. One of the limitations is the parameters applied to the traditional SA. Another is the realization of the neighbors function.

The neighbors function is inferior.

(1)The traditional transformation rules are not supporting the non-join operator. The transformation rules used by SA only consider the "inner-join" which can be freely reordered. The non-join, such as outerjoin and antijoin, is not supported by SA which has not freely associative property.

(2) The neighbors function is deficiency. Given a state s, |NA(s)| is also called neighbors function. The cost change (△C in algorithm 2) when called the neighbors function is usually small relatively to the total cost. For example, chang- ing the order of two joins cannot significantly affect the total cost of the whole computation. Hence, even at very high temperatures the system does not undergo drastic changes in terms of its cost.

The limitation of the parameters is following:

(1) Time finding the solution is too long. In the situation of the complex of the objective function and many variables, in order to find a suboptimal solution, the controlling parameter T must start from the larger value, and must implement repeatedly the algorithm of metropolis. So the operation of iteration is slow.

(2)The initial value and the decreasing step size of tempera- ture are difficultly defined. If the initial value of T is larger and the decreasing step of it is too smaller, although the better solution can be found, the rate of algorithmic convergence is too slow; otherwise, the global optimal solution may not be found.

(3) The search process misses the optimal solution of the current encounter because receiving the executive probability.

B. *Improved Programs*

On basis of the optimization quality satisfying that certain requirements, improving the search efficiency of SA algorithm is main. The improved programs are following:

(1) Setting up dual-threshold to reduce amount of calcula- tion.

(2) Designing an adaptive temperature update function. If the state is received many times at one temperature, the decreasing step size of temperature should larger. Otherwise, it should smaller. Thus the temperature can remain some adaptive in the update.

(3) Adding memory function. In order to avoid miss the current optimal solution, the improved algorithm is increased memory function to remember the current best state so that it becomes an intelligent algorithm

(4) Realizing effective and practical neighbors function. At first, we propose a comprehensive and practical framework for transformation rules and make it support the complex non-inner join. By this way, we improve the practicality of the algorithm. Secondly, we produce the join pairs based on graph approach and construct the query plans. At last, we use the random query plan as the basic unit of perturbation, not the transformation rule. By this way we improve the efficiency of neighbors function.

V. PRACTICAL AND EFFECTIVE SA

In order to make the neighbors function support non-join and improve the perturbation efficient. This section the practical and effective improved SA called as $SA_{improved}$ will be introduced.

A. *The Semantics Expression of Query Supporting non-join*

"inner-join" is the most common type of join operation, which can be freely reordered. The freely reorder property endows the optimizer a great convenience and lead to a great deal of possible join orderings. Then the optimizer chooses the cheapest execution query. Liking inner-join, improve the performance of query execution can be completed by reorder the outerjoins and antijoins [13]. Because the associative with each other of these types of joins are not always valid, reorder involving outerjoins and/or antijoins are complicated, i.e. not all order will give the same answer as the original query, which cause the generation of equivalent plan a much more difficult task, unless special consideration is taken.

It is convenient to show connections by deriving a query graph consisting of base relations as nodes, plus edges for predicates. However the difficulty when a query involving outerjoins and/or antijoins is that, unlike joins, a query graph without information on evaluation order is ambiguous.

The semantics expression shows how to capture the seman- tics of a query through query graph analysis to detect conflict- ing when a query involving outerjoins and/or antijoins.

At the beginning of explanation of the algorithm, we briefly introduce the rules of conflict. All of the conflicts are summarized in Figure 1.

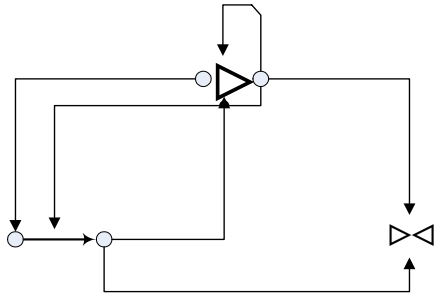

Figure 1. The conflicts of join predicate

The conflicting graph of Figure 1 provides the following information. An inner-join has no conflicts with outerjoin and antijoin. An antijoin has conflicts with outerjoin "pointing" inward in the preserving side and all kinds of joins in the null-producing side. An outerjoin conflicts with antijoin and inner join in the null-producing side.

Let us now formalize this approach. As usual, A NS includes relations referenced by the join predicate, which is used to associate each join predicate. An ES includes additional relations needed by a predicate to preserve the semantics of the original query. We denote by ref(p) the set of tables referenced by the join predicate p. The ref(preserving(p)) and ref(nullproducting(p)) denote the set of table reference by the join predicate p in the preserving side and null-producing side. An set_outerjoin contains all the relations associated by inner join or antijoin predicates. An set_antijoin includes the relations that are linked to each relation through outerjoins pointing to each relation.

After initializing ES with NS for every join predicate p and an set_outerjoin and an set_antijoin for each table in the join to include only the table itself. Figure 2 describes how to sets the ES of outerjoin, antijoin, inner join predicates.

Example 1. Figure 2 is illustrated using Example 1. In Example 1, the ES for predicate Prs is {R, S, T}, which requires table S be joined with table T before R. The ES for predicate Puv includes{R, T, U, and V} and thus Puv

Example 1.

$$\left(\left(R \xrightarrow{Prs}\left(S \overset{Pst}{\triangleright\!\triangleleft} t\right)\right) \xrightarrow{Ptu} U\right) \overset{Puv}{\triangleright\!\triangleleft} V$$

	R	S	T	U	V
set_outerjoin	{R}	{S,T}	{S,T}	{U}	{V}
set_antijoin	{R}	{R,S}	{R,T}	{R,T,U}	{V}

	NS	ES
Psr	{R,S}	{S,T,R}
Pst	{S,T}	{S,T}
Ptu	{T,U}	{S,T,U}
Puv		{U,V} {R,T,U,V}

can only be applied at the end. The order of the two outer joins can be swiched. So ((R,((S,T,),U)),V) is valid order equivalent with original query.

By the semantics expression of query graph, we can detect whether the join reorder is rational. So this can cause the generation of equivalent plan.

B. The Realization of Efficient Perturbation

The neighbors function is deficiency. The main reason is SA uses the transformation rule as the basic unit of perturbation, which is usually small relatively to the total cost. Hence, even at very high temperatures the system does not undergo drastic changes in terms of its cost. The realization of efficient perturbation is by using the random query plan as the basic unit of perturbation.

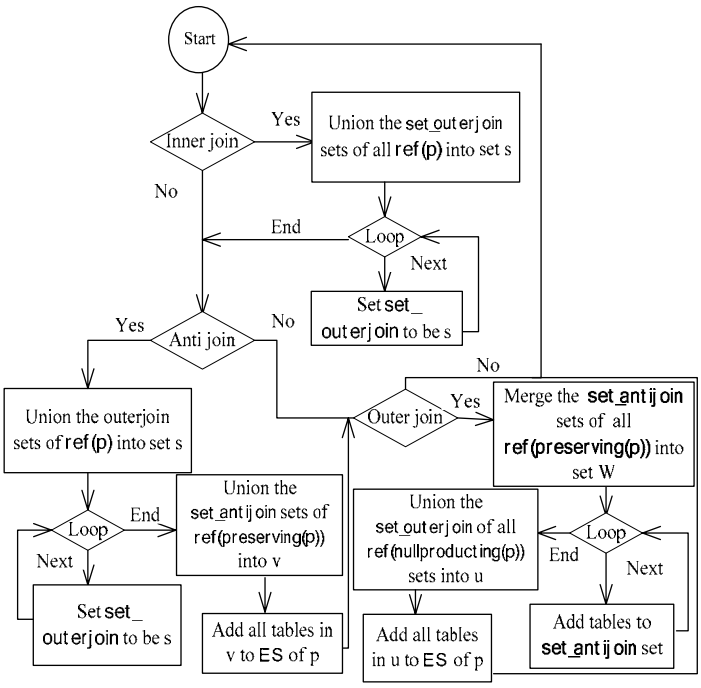

Figure 2. semantics expression of query graph

So in this subsection, we introduce the method how to produce non-empty connected compare pairs base on graph-based and use the non-empty connected join pairs, not transformational rule, to construct query plans base on join pairs used by perturbation.

1)Construction of Connected Join Pairs

Let us start the exposition by fixing some notations. For a node $v \in V$ define the neighborhood N(v) of v as $N(v):=\{v' | (v, v') \in E\}$. For a connected subset $S \subseteq V$ of V we define the neighborhood of S as $N(S):=\cup v \in s\ N(v) \backslash S$. The neighborhood of a set of nodes thus consists of all nodes reachable by a single edge. The complement of S is a connected subgraph comprised by V\S.

The following statement gives a hint on how to construct the connected subsets. Let S be a connected subset of an undirected graph G and S' be any subset of N(S). Then $S \cup S'$ is connected. As a consequence, a connected subset can be enlarged by adding any subset of its neighborhood using a breadth-first.

Figure 3 provides a skeleton framework generating all connected subsets and the complement subsets.

For every descending element of all nodes union (line 1), we firstly construct s1 to be {v}. And then, an iteration function is called and mainly used to construct s1 (line 4) by expanding the node adding the neighborhood N (v) (line 3).

Consider every v of S1 set (line 5), we construct S2 set. At first, for each descending neighborhood of v (line 3), we construct s2 to be the descending neighborhood, and then, recursively create those s2 that more than a single node (line 4). At last, we return the non-empty connected compare pairs {(s1, s2), (s2, s1) |$s1 \in$ S1, $s2 \in$ S2}.

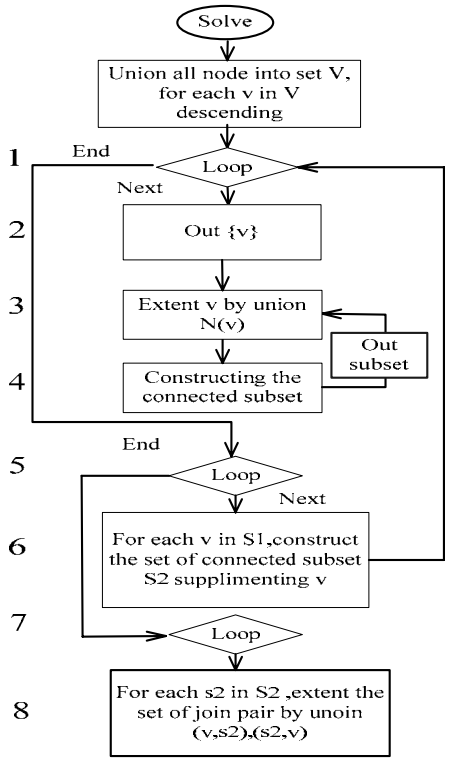

Figure 3. semantics expression of query graph

2)The Production Of Query Qlans Supporting Non-join

After construct non-empty connected compare pairs, we will use the compare pairs product query plans supporting non-join (called QPnon-join). In order to illustrate this process, we use Figure 4 as the query graph.

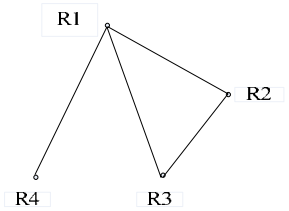

Figure 4. query graph

To product query plans using the connected compare pairs, we convert the total order over compare pairs into a partial over unordered groups of pairs. We reorder an incoming sequence of pairs of quantifier sets (called join pair sequence) by the size of the resulting quantifier set. This is illustrated using the following Figure 5, which contains all reordered join pairs and codes of Figure 4.

We will sequentially use the reordered non-empty connected join pair to acquire the query plans. In this process, there is an important function called Partition, the following definition formally defines Partition.

Definition 3. For a join S = G_l and G_r are connected sub-graph and include respectively n and m relations. Through the following process we decompose S:

(1) Index reorder join pairs group including n relations, if the LVA$\cup$RVA= G_l, we acquire QS. We utilize the QS to express G_l. If QS is more than one, the "or" operator is used to connect QS.

Group[1]

QS	VA
R1	1
R2	2
R3	3
R4	4

Group[2]

QS	LVA	RVA
R1,R4	1	4
R1,R3	2	3
R1,R2	2	4
R2,R3	3	4

Group[3]

QS	LVA	RVA
R1R2,R3	1,2	3
R1R2,R4	1,2	4
R2R3,R1	2,3	1
R1R4,R2	1,4	2
R1R4,R3	1,4	3
R1R3,R2	1,3	2
R1R3,R4	1,3	4

Group[4]

QS	LVA	RVA
R1R2R3,R4	1,2,3	4
R1R2R4,R3	1,2,4	3
R1R3R4,R2	1,3,4	2
R0R1R3,R4	1,4	2,3

Figure 5. the index structure of the reorder join pairs group

(2) The process of G_r is similar with G_l.

(3) S is expressed by the decomposed G_l and G_r.

Example 2. Definition 3 is illustrated using the following example. The R0R1R2 in Group [4] can be logical expressed via traveling every row in Group [3] and estimates whether the LVA$\cup$RVA equals the {0, 1, 2}. The collection satisfying the condition is considered as the logical expressing of R0R1R2. This method makes the Top-Down dynamic programming not relying on transformation rule of traditional. It is optimal with respect to the join graph and avoids the Cartesian products which can extremely decreasing the search space.

We now define the important process of constructing query plan in Definition 4.

Definition 4. Start from root layer (including all relations, but no join operator), we use the Group[n] (n is n is the number of relations) to express root layer. Then calling Partition iteratively decomposes the join pairs of each layer. The production of query plan is also the top-down traveling process of decomposed query tree.

This process is illustrated using the following example. Figure 6 achieves query graph's Partition of figure 4 from top to down. Then it travels the decomposed query tree from top-down (labeled by $\rightarrow$) to product the query plan. At last, it uses the ES to select query plan (called as QPnon-join), whose join order is valid (i.e., supporting non-join operator).

Definition 5. The total number of QPnon-join is defined as following:

$$\text{QPnon-join}=SUMestrue\left(NUM\,\text{sec}+\sum_{i=1}^{h}\left(\sum_{j=1}^{p}(OREDGE-1)\right)\right)$$

*NUM*sec is the number of join pairs in second layer. h height is the of decomposed tree. p is the number of "or" operator of some layer. *OREDGE* is "or" operator's the number of edges. *SUMestrue* is using *ES* to calculate the query plan's number, witch join order is valid.

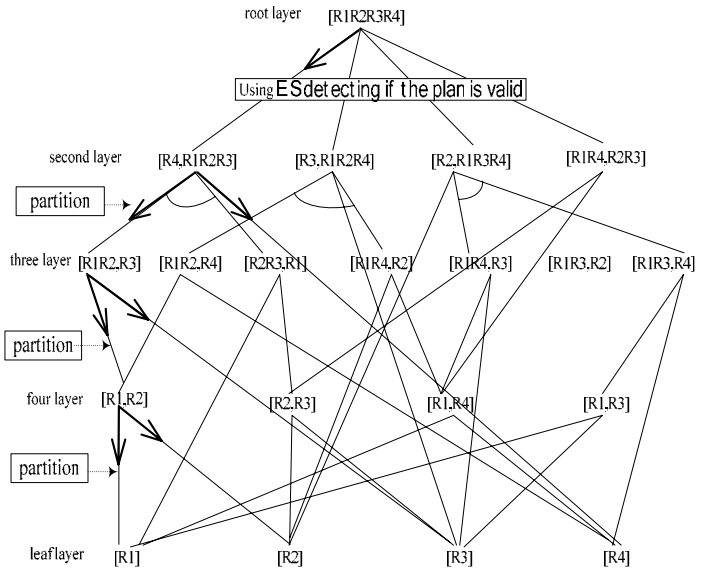

Figure 6. the process of construct QPnon-join

C. The Algorithm of SAimproved

The whole $SA_{improved}$ includes two stages, i.e. improved process of Annealing and perturbation.

Improved process of perturbation including:

(1) $SA_{improved}$ firstly run for a small period of time, i.e. a few local optimizations are performed. The output of this phase, which is the best local minimum, is the initial state of the next phase.

(2) Use the QPnonjoin as the basic unit of perturbation resolves the problem of cost changed too small.

(3) Design an adaptive temperature update function and set up dual-threshold to reduce amount of calculation. If the current state remains immovability with continuous maxstep2-step, we consider the Metropolis sample is stability; during the processing of temperature decreasing, if the optimal solution remains immovability with continuous maxstep1-step, we consider the algorithm is convergence.

Improved process of Annealing including:

In order to avoid miss the current optimal solution, the improved algorithm is increased memory function to remember the current best state.

VI. PERFORMANCE ANALYSIS

In order to make II, 2PO and SA support non-join, we apply ES method to them. The II, 2PO and using ES is respectively called $II_{nonjoin}$, $2PO_{nonjoin}$ and $SA_{nonjoin}$. In this section, we report on an experimental evaluation of the performance and behavior of $II_{nonjoin}$, $2PO_{nonjoin}$, $SA_{nonjoin}$ and $SA_{improved}$ on query optimization. Due to known difficulties in tree query optimization, specific attention was given to star queries. We describe the tested that we used for our experiments by Table III.

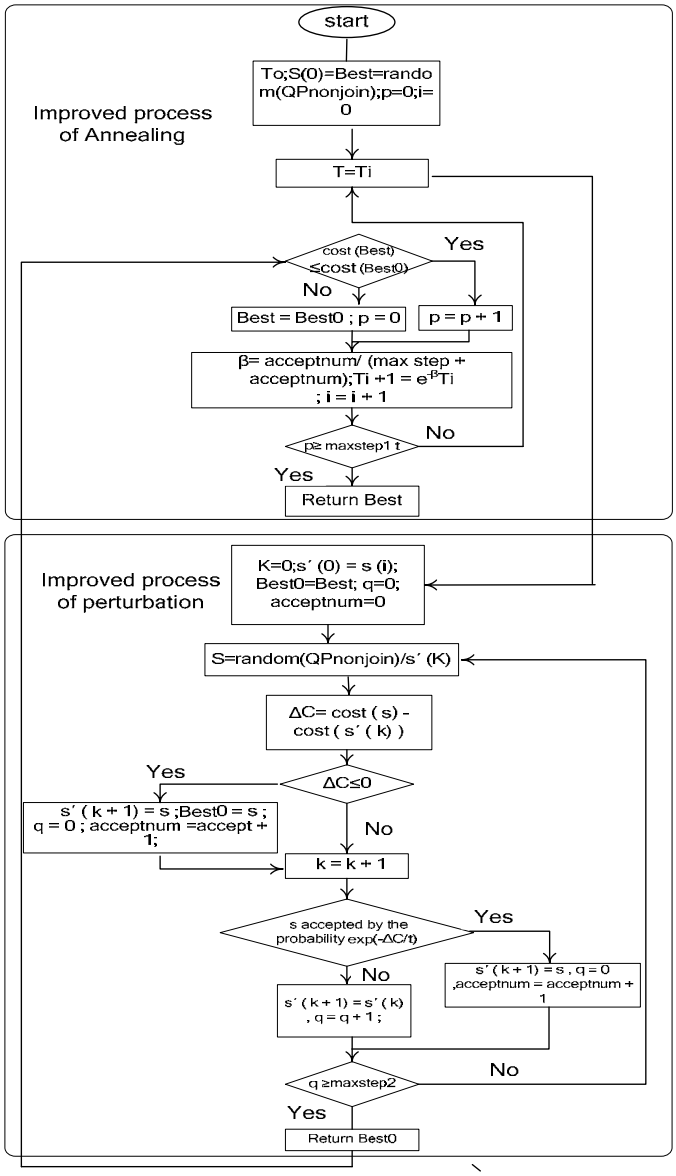

Figure 7. improved SA algorithm

TABLE III.
THE CATALOG OF TEST

Algorithm involved	Join involved	#of quantifiers	topology
$II_{nonjoin}$ $2PO_{nonjoin}$, $SA_{nonjoin}$, $SA_{improved}$	*join,outerjoin, antijion*	*[20,40]*	*star*

A. Performance as a Function of Time

As part of the experiments, we recorded how the minimum cost found changed over time during the course of the algorithms' execution. The typical behavior is shown in Figure 8. The particular example is for a 40 inner-join query. The y-axis represents the ratio cost over the minimum strategy cost found for the query among all runs of all algorithms.

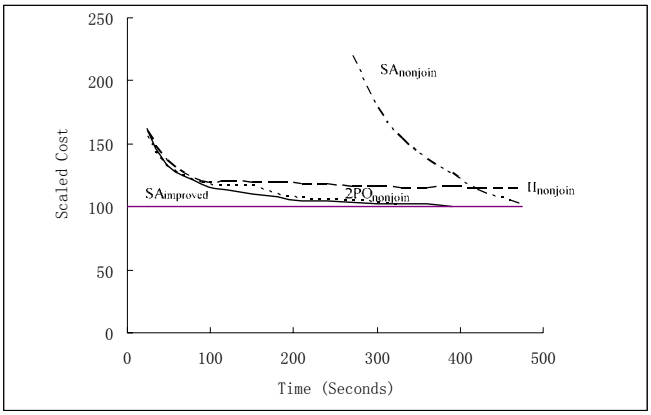

Figure 8. performance of a function of time

Clearly there are significant differences between $SA_{nonjoin}$ and $II_{nonjoin}$. On the one hand, after a few local optimizations, II reaches a state of cost that is close to the minimum cost found by a complete run of $SA_{nonjoin}$. After that $II_{nonjoin}$ makes only small improvements. On the other hand, in the early stages, $SA_{nonjoin}$ wanders around states of very high cost. During the later stages, however, it reaches states of costs similar to those found by $II_{nonjoin}$ after a few local optimizations, and eventually finds a better state. This fact is what motivated the introduction of $SA_{improved}$. The first phase of $SA_{improved}$ produces a low cost state from which the second phase can start with low temperature. Indeed, we observe that performance and behavior of $SA_{improved}$ initially is similar as $2PO_{nonjoin}$, but soon it surpasses it, and eventually converges to its final solution much more rapidly than $2PO_{nonjoin}$.

B. Output Quality

The cost of the output strategies produced by the algorithms for the various relation catalogs as a function of query size is shown in Figure 9 for star query. Again the y-axis represents scaled cost, i.e. the ratio of the output strategy cost over the minimum cost found for the query among all runs of all algorithms.

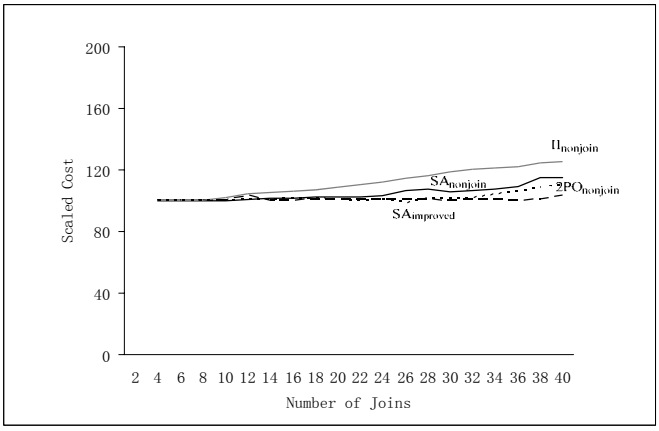

Figure 9. output quality

Clearly, $2PO_{nonjoin}$ and $SA_{improved}$ not only outperforms the other algorithms, but also becomes very stable. So $SA_{improved}$ and $2PO_{nonjoin}$ is the algorithm of choice for large queries, particularly if it is run a small number of times for stability. We should also observe, however, that the performance of $SA_{nonjoin}$ becomes very stable as well. To the contrary, $II_{nonjoin}$ is still not very stable, rarely outperforming $SA_{nonjoin}$.

VII. SUMMARY

Query optimization for relational database systems is a combinatorial optimization problem, which makes exhaustive search unacceptable as the query size grows. In this paper, we analyze and discuss the advantages and disadvantages of the well-known randomized optimization of traditional Simulated Annealing and propose the $SA_{improved}$ algorithm, whose performance is superior to that of the other algorithms with respect to both quality and running time.

The work reported in this paper is only the beginning in understanding how randomized optimization algorithms of SA perform on complex queries. There are several issues that are interesting and on which we plan to work in the future. First, we want to experiment other topologies of relational queries. Second, we would like to consider the better perturbation function than the random method of query solution. Finally, we want to compare $SA_{improved}$ and the other randomized optimization algorithms with the traditional ones, e.g. those of System-R. This should lead into an understanding of the relative advantages between generation-based and transformation-based query optimization.

ACKNOWLEDGMENT

The research is under the support of the Specialized Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 200801830021; Nature Science Foundation of China under Grant No.60973040, 60903098; The Science and Technology Development Program of Jilin Province of China under Grant No.20070533; The basic scientific research foundation for the interdisciplinary research and innovation project of Jilin University under Grant No.200810025

REFERENCES

[1] Wolfgang Scheufele and Guido Moerkotte, “On the complexity of generating optimal plans with cross products,” In ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pages 238–248, 1997.

[2] Swami, A.N, “Optimization of Large Join Queries: Combining Heuristic and Combinatorial Techniques,” In Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp. 367–376 ,1989.

[3] Lanzelotte, R.S.G., Zaït, M., Gelder, A.V, “Measuring the effectiveness of optimization. Search Strategies,” In BDA, pp.162–181, 1992.

[4] Tan, K.L., Lu, H, “A Note on the Strategy Space of Multiway Join Query Optimization Problem in Parallel Systems,” in SIGMOD, Vol.20, pp.81–82, 1991.

[5] Ioannidis, Y.E., Christodoulakis, S, “ On the Propagation of Errors in the Size of Join Results,” In Proc. of the ACM

SIGMOD Intl. Conf on Management of Data, pp. 268–277, 1991.

[6] Lanzelotte, R.S.G., Valduriez, P, Zaït, M, "On the Effectiveness of Optimization Search Strategies for Parallel Execution Spaces," In Proceedings of the 19th International Conference on VLDB, pp. 493–504, 1993.

[7] Swami, A.N., Gupta, A, "Optimization of Large Join Queries," In Proceedings. of the ACM SIGMOD International Conference on Management of Data, vol. 17, pp. 8–17, 1988.

[8] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper, "Heuristic and randomized optimization for the join ordering problem," In the VLDB Journal, vol. 6, pp. 191–208, 1997.

[9] Avnur, R., Hellerstein, J.-M.: Eddies, "Continuously Adaptive Query Processing," In Proceedings of the ACM SIGMOD International Conference on Management of Data, vol. 29, pp. 261–272, 2000.

[10] Yannis E. Ioannidis and Eugene Wong, "Query optimization by simulated annealing," In Proceedings of the 1987 ACM SIGMOD international conference on Management of data, vol. 16, pp. 9–22, 1987.

[11] C. Galindo-Legaria and A. Rosenthal, "Outerjoin simplification and reordering for query optimization," In TODS, vol. 22, pp. 43–73,1997.

[12] Arun N. Swami and Anoop Gupta, "Optimization of large join queries," In SIGMOD Conference, vol. 17, pp. 8–17, 1992.

[13] J. Rao, B. Lindsay, G. Lohman, H. Pirahesh, and D. Simmen, "Using EELs: A practical approach to outerjoin and antijoin reordering," In Proceedings of the 17th ICDE, pp.585, 2001.

[14] Y. E. Ioannidis and Younkyung Kang, "Randomized algorithms for opti-mizing large join queries," In Proceedings of the 1990 ACM SIGMOD international conference on Management of data, pp. 312–321, 1990.

[15] Ioannidis Y E, Kang Y C, "Randomized algorithms for optimizing large join queries," In ACM SIGMOD international conference on Management of Data, vol. 19, pp.312--321, 1990