Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/090339

Degradation of Complexity for Join Enumeration via Weight Measure on CMP

Yongheng Chen[1,*], *Kerui Chen*[2] *and YaoJin Lin*[1]

[1] College of Computer Science,MinNan Normal University, Zhangzhou 363000, China
[2] College of Computer and Information Engineering, Hennan University of Economics and Law, Zhengzhou, China

Received: 9 Aug. 2014, Revised: 10 Nov. 2014, Accepted: 11 Nov. 2014
Published online: 1 May 2015

Abstract: Most contemporary database systems query optimizers exploit System-R's Bottom-Up dynamic programming method (DP) to find the optimal query execution plan (QEP). 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. In this paper we propose a parallel top-down join enumeration algorithm that is optimal with respect to the partial order graph based on Chip Multi-Processor (CMP). This paper firstly transforms the undirected query graph to Weighted Edge Join Graph (WEJG) according to the edge weight and constructs all partial order join and partial order graph within WEJG. Then the global optimal query plan is achieved according to the parallelize top-down enumeration. Our theoretical results and empirical evaluation show that our algorithm could gracefully degrade the complexity degree for top-down join enumeration with large number of tables and gains impressive in the performance in terms of both output quality and running time.

Keywords: chip multi-processor, parallel query processing, DP query optimization

1 Introduction

On the hardware front, the development trend of processor is transforming from high-speed single-core to Chip Multi-Processor, and from instructions level parallel to thread level parallel. Tomorrows computer will have more cores rather than exponentially faster clock speeds, and software designs must be restructured to fully exploit the new architectures [1]. The question for database researchers is this: how best can we use this increasing multithreading capability to improve database performance in a manner that scales well with machine size [2,3,4]?

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 [5]. 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 [6,7].

Based on this trend, it has become temping to revisit the concepts of database parallelism in the light of those emerging hardware architectures. Most contemporary database systems perform cost-based join enumeration using some variant of System-R's bottom-up dynamic programming method [8]. For example, recently, by exploiting the new wave of multi-core processor architecture, Han et al. first propose a novel algorithm PDPsva to parallelize query optimization process to exploit multi-core processor architectures whose main memory is shared among all cores [9]. This join enumeration algorithm separates join pair generation and plan generation. To avoid synchronization conflict,

* Corresponding author e-mail: cyh771@163.com

PDPsva convert the total order over join pair into a partial order by grouping the same size of the resulting quantifier set, and then, perform plan generation using multiple threads. PDPsva generated QEPs for all smaller quantifiers sets (i.e. size-driven). On contrary, the bottom-up dynamic programming optimizers such as DPcpp [10] and DPhyp [11], which directly traverse a query graph to generate join pairs, i.e., only considers pairs of connected sub-queries. Thus, plan generation mainly use of join pair without cross products, reduce execution time. Further, DPhyp is capable to handle complex join predicates efficiently.

The algorithms discussed above which all constructed based on bottom-up join enumeration method. The notable exceptions are systems based on the top-down transformational search of Volcano/Cascades [12]. David DeHaan describes a top-down join enumeration algorithm that is optimal with respect to the join graph [13]. This algorithm enumerates the search space top-down, it does not rely on transformations and thus retains much of the architecture of traditional dynamic programming. As such, this work provides a migration path for existing bottom-up optimizers to exploit top-down search without drastically changing to the transformational paradigm.

But these algorithms discussed above which all constructed based on Dynamic Programming (DP) methods. DP methods, regardless of Top-down or not, face a difficult for complex queries because of its inherent exponential nature owing to the explosive search space.

The heuristically pruning, almost exhaustive search algorithms used by current optimizers are inadequate for queries of the expected complexity [11,2]. 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 where the validity depends on the nature of the considered search space [6,14].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.

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 [15]. 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, V.V. Meduri et al present a good overview [16]. Approaches such as Iterative Improvement, 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. John W. Raymond and Peter Willett give a thorough survey of the various approaches towards the detection of subgraph isomorphism [18]. Bertrand et al. aim at finding the largest common induced subgraph of two graphs [19]. Qiang Zhu et al. introduced a technique to perform query optimization via exploiting similarity of substructures in a given complex query [15]. Meduri applied Simulated Annealing to the optimization of some recursive queries and constructed the plans by re-using the query plans among the identified similar sub-queries and avoided multiple plan construction for each join candidate in order to make memory efficient [17]. However these algorithms are proposed for single-core CPU. Impressive gains in the performance of running time by these algorithms have been built largely on contempt for the quality of the optimization plan. On the other hand, it does not take advantage of the special characteristics of the query[20].

In order to improve quality of the output plan and consider the characteristics of the query, the parallelize top-down enumeration based on Weighted Edge Join Graph is proposed in this paper. The method firstly parameterizes and quantifies the impact of a join to the cost of its succeeding join and constructs WEJG. Secondly, the partial order join and partial order graph are archived take advantage of the weight in WEJG. Finally, the global solution is constructed based on parallelize top-down enumeration.sults of performance evaluation and conclude this paper.

2 Query Graph Definition

2.1 Preliminary Concepts

Graph is widely used to represent query structure. A connected Graph is denoted by G(V, E, T, P, α, β), where V is the finite set of its vertices, E $\in$ V $\times$ V the set of edges, α a function assigning labels to the vertices and β a function assigning labels to the edges. T = {R1, R2...Rn} is the set of tables referred in G and P = {p1, p2...pm} the set of all predicates referred in G. $\alpha : \mapsto$ R is a one-to-one function, where x $\in$ V and R $\in$ T. β : e $\mapsto$ c is a function, where e $\in$ E and c $\in$ P.

NumRel(V) denotes the number of vertices in V. NumE(E) denotes the number of edges in E. The edge e $\doteq$ (u, v) $\in$ E is said to be incident with vertices u and v, where u and v are the end of e. These two vertices are called adjacent. vertices (e) denotes the set of vertices connected by edge e in a query graph. N(v) obtains the adjacent nodes of node v.

Definition 2.1: A connected graph S (V, E, T, P, α, β) is a sub-query of G (V, E, T, P, α, β) if

V= {v| v$\subseteq$ N(E) $\in$V },

E= {e| e$\subseteq$E and vertices (e) $\in$V},

T= { r| r$\in$T and α(r) $\in$V},

P= {p| p$\in$P and p $\in$h and h is the set of predicates labeled on e $\in$E},

α: x $\mapsto$ r, where x∈V, r∈T and α(x) $\doteq$r,

β: e$\mapsto$c, where e∈E, c∈Pand β(e)$\doteq$c.

For a subquery S (V, E, T, P α,β) of G (V, E, T, P, α, β), the neighborhood of S is denoted as ad(S) = {v∈ (V-V)| (u,v)∈E and u∈V}.

Definition 2.2: A connected sub-query S (V, E, T, P, α, β) of G (V, E, T, P, α, β) is a partial order join or POJ for short, if the connected graph S only includes two different edges, called $E_1 and E_2$, and the predicates {β (E_1), β (E_2)} are a partial order, i.e., one of $\{\beta(E_1), \beta(E_2)\}$ must be executed before the other.

Definition 2.3: A connected sub-query S (V, E, T, P,α, β) of G (V, E, T, P, α, β) is a partial order graph or POG for short, if the connected graph S has more than two edges, and the predicates are partially ordered set.

2.2 Weighted Edge Join Graph

The definition of a G can be further extended to include a weight and a direction among edges to represent precisely the influence of a join to the cost of the next join.

For convenience, we will use $\bowtie_i$ to represent the ith join predicate operation in the join sequence. We use an example to illustrate the idea. A connected graph with predicate operation $\{\bowtie_1, \bowtie_2, \bowtie_3, \bowtie_4\}$ is considered the following join sequence:

$(((\bowtie_1) \bowtie_2) \bowtie_3) \bowtie_4$

In this sequence, $\bowtie_1$ is carried out first. As the cost of the next join operation $\bowtie_2$ is dependent on its operand sizes, the result produced by $\bowtie_1$ directly influences the cost of $\bowtie_2$. This influence is termed an impact in this paper. For instance, assume that one of the inputs of $\bowtie_2$ is relation S and relation S is the common base relation of $\bowtie_1$ and $\bowtie_2$. As $\bowtie_1$ is evaluated before $\bowtie_2$, the intermediate result of $\bowtie_1$ will replace the common base relation S as the input to $\bowtie_2$. If $\bowtie_1$ produces a large result (assume much larger than S), then it creates s stronger impact to $\bowtie_2$ because S, one of the original operand relation of $\bowtie_2$, is replaced by a larger relation. Otherwise, $\bowtie_1$ gives a small impact to $\bowtie_2$. In the following, we formally parameterize and quantify the impact of a join to the cost of its succeeding join and embed it in a Weighted Edge Join Graph.

Definition 2.4: A Weighted Edge Join Graph (WEJG) (V", E", P", α", β") of G (V, E, T, P, α, β) is defined as following:

V"= $\{e \mid e \in E\}$

E"= $\{e_1e_2,\ e_2e_1 \mid e_1, e_2 \in$ E and vertices$(e_1) \bigcap$ vertices$(e_2) \neq \emptyset\}$

P"= {p | p ∈ P}

α": v$\mapsto$t, where v ∈V", t∈ $\{\alpha(v_n) | v_n \in$ vertices(v)$\}$ ∈ T.

β": e$\mapsto$c, where e∈V", c =β(e)∈P", and β"(e) =c.

WEJG is a weighted complete (p, q) digraph, where each vertex is connected to another vertex via two edges in opposite direction, and the vertex in WEJG is exactly the edge in the connected Graph G based on the definition. From the definition, WEJG is deducible from G. Let us consider an example. WEJG of connected Graph G in Fig. 1(a) is shown in Fig. 1(b). The predicate between node v1 and v2 is indicated as $\bowtie_i$, v2 and v3 as $\bowtie_j$, and v3 and v4 is indicated as $\bowtie_k$ in Fig. 1(a).

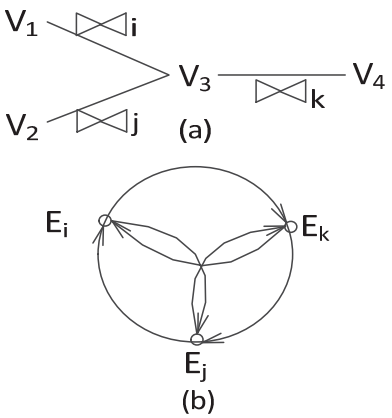


Fig. 1: Weighted Directed Join Graph

If there is common node between v_iv_j and v_lv_k, two edges in opposite direction exist between the two. By this rule, Fig. 1(b) can be achieved. Every edge in graph G will be regarded as a node by Corresponding WEJG. In Fig.1(b) E_i, E_j and E_k are used to denote $\bowtie_i$,$\bowtie_j$ and $\bowtie_k$ respectively. The direction of an edge $E_i \mapsto E_j$ indicates an execution partial order of $\bowtie_j$ immediately after $\bowtie_i$. Associated with each edge $E_i \mapsto E_j$ is a weight W_{ij}, which is calculated as follows:

$$W_{ij} = \langle W^1_{ij}, W^2_{ij} \rangle$$

$$= \left\{ \left\langle \frac{\text{The cardinality of the result of } \bowtie_i}{\text{The cardinality of } \bowtie_i \cap \bowtie_j} \right\rangle, \left\langle \frac{\text{The tuple size of the result of } \bowtie_i}{\text{The tuple size of } \bowtie_i \cap \bowtie_j} \right\rangle \right\}$$

In order to simplified representation, we use E_{ij} to denote Ei$\mapsto$Ej. Edges(E_{ij}) function is used to achieve the partial ordered set E_i, E_j. SN(E_{ij}) and EN(E_{ij}) are used obtain the first and last element in the partial ordered set. Accordingly, we define:

SN(E_{ij})=Edges(E_{ij})[0]

EN(E_{ij})=Edges(E_{ij})[Edges(E_{ij}).length-1]

Such aswe can list the following equation according to above equation

$$SN(E_{123}) = Edges(E_{123})[0] = \{E_1, E_2, E_3\}[0] = E_1$$

$$EN(E_{123}) = Edges(E_{123})[2] = \{E_1, E_2, E_3\}[2] = E3$$

2.3 Weight Parameters

As the join selectivity factor is normally assumed known from collecting statistical information, the elements of a weight can be defined based on such known parameters. Let us formally define those parameters that are used in the weight representation.

Join Selectivity Factor (JSF)

Join selectivity factor is a factor to represent the ratio of the cardinality of a join result to the cross product of the cardinalities of the two join relations.

$$JSF_i = \frac{|R_i \bowtie S|}{|R| \times |S|} \quad |X| \text{ means the cardinality of relation X}$$

Join Concatenation Factor (JCF)

Join Concatenation Factor is the ratio of the concatenated tuple size of the join to the sum of the sizes of two join relations.

$$JCF_i = \frac{||R \bowtie S||}{||R|| + ||S||} \quad ||X|| \text{ means the width of each tuple of X}$$

Assume that $\bowtie_j$ is over R and S and $\bowtie_i$ is over S and T. Then, the weight of $\bowtie_j \mapsto \bowtie_i$ can be reformulated as follows:

$$W_{ji} = \langle W^1_{ji}, W^2_{ji} \rangle = \left\langle \frac{|R| \times |S| \times JSF_j}{|S|}, \frac{(||R|| + ||S||) \times JCF_j}{||S||} \right\rangle$$

Let us define an operator $\odot$ as:

$s \odot t = \langle s_1 \times st_1, s_2 \times t_2 \rangle$

$s = \langle s_1, s_2 \rangle$ $t = \langle t_1, t_2 \rangle$

The weight can then be further simpled as following.

$$W_{ji} = \langle W^1_{ji}, W^2_{ji} \rangle = \left\langle |R| \times |S|, ||R|| + ||S|| \right\rangle \odot \left\langle \frac{JSF_j}{|S|}, \frac{JCF_j}{||S||} \right\rangle$$

Therefore, the weight of an edge between $\bowtie_j = R \bowtie S$ and $\bowtie_i = S \bowtie R$ is expressed as follows.

Consider two edges $\bowtie_1$ and $\bowtie_2$where $\bowtie_1 = R \bowtie S$ and $\bowtie_2 = S \bowtie T$. An weight $W_{12} = \langle 2, 1 \rangle$ implies that the result of $\bowtie_1$ has twice as many tuples(i.e.,the cardinality is doubled) as the number of tuples of S, an input to $\bowtie$. Because the cost of a join depends on its input size, this $\langle 2, 1 \rangle$ weight implies that the degree of influence (impact) of $\bowtie_1$ to $\bowtie_2$ is doubled in cardinality(since $w^1{}_{12}=2$) and remains the same in tuple size(since $w^2{}_{12}=1$). Consequently, the WEJG and the edge weights in it can be used as a tool for finding a good (low cost) execution plan for query optimization.

$$W_{ji} = \langle W^1_{ji}, W^2_{ji} \rangle = \begin{cases} \langle |R| \times |S|, ||R|| + ||S|| \rangle \odot \left\langle \frac{JSF_j}{|S|}, \frac{JCF_j}{||S||} \right\rangle & \text{if } \bowtie_i \cap \bowtie_j \neq \emptyset \\ \langle 1, 1 \rangle & \text{if } \bowtie_i \cap \bowtie_j = \emptyset \end{cases}$$

3 Generating cover set of Partial Orders

3.1 Construction of POJ List

Generating the partial order join is an important first step. We firstly need consider the relationship between edges of G.We could generate all partial order join according to two enumeration steps.

For every node $v_i \in V$ we perform the first enumeration step: Firstly, we use $N(v_i)$ to obtain the adjacent nodes. Then, for every adjacent nodes achieved, the edge is represented by E_{NumN}. NumN is an incrementing variable starting with one. In order to avoid producing duplicate edge, the idea is to use the numbering to define an enumeration order: the algorithm enumerates adjacent nodes for every node v_i, but not considers node n_j with j<i. Using the definition $B_i = v_j | j \leq i$.

For every node $v_i \in V$ we perform the second enumeration step: firstly, we calculates the neighborhood of v_i. If the number of neighborhood is greater than 0, the weight W^1 is calculated among the edges of v_i and adjacent nodes. And add partial order join to set of POJ set. The execution is demonstrated in pseudo code format by CreatePOJ Algorithm.

Let us consider an example for CreatePOJ algorithm.Figure 2 contains a query graph whose nodes

CreatePOJ Algorithm
Input: a connected query graph G = (V, E)
Precondition: nodes in V are numbered according to a breadth-first search
Out: emits all POJs of G
initialize POJ=ϕ NumN=1
for all i∈[n − 1, . . . , 0] descending {
 N=N(n_i)/B_i;
 for node n∈ N {
  using E_{NumN} denote edge (n_i, n)
  NumN++; }
}
for all i∈[n − 1, . . . , 0] descending {
 N=N(n_i);
 if (N>0)
  for node N_R∈N{
   for node N_L∈[N-N_R] {
    if ($W^1_{N_RN_L}$<1)
     append $E_{N_RN_L}$ to POJ }
  }
 }

are numbered. The result of first Enumerations for all nodes is indicated by Figure 3(a). According the existing edges in Figure 2, obviously, eight edges are produced. For example, the edge of n_1 to n_2 is represented as E_1, n_1 to n_4 is E_2 and so on. It is an upper triangular matrix by using B_i function. The nodes of Figure 3(a) are the result of first enumeration by CreatePOJ algorithm. The directed edge with W^1 less than one in Figure 3(b) is the result of second enumeration by CreatePOJ algorithm. The resulting set of POJ is {E_{12}, E_{31}, E_{32}, E_{41}, E_{45}, E_{67}, E_{78}, E_{74}}.

3.2 Enumeration POG by Growth of POJ

In the previous subsection, we have constructed POJ. In this subsection, we present CreatePOG algorithm to generate POG of query graph. The POG can be producted by combining the POJ and smaller constructed POG. So POG is the extension of POJ. The pseudocode of CreatePOG algorithm looks as follows:

In this algorithm, the variable POG_0 is initialized with POJ. Two phases are included in the algorithm.

The first phase is to choose an edge that can reduce the cost of constructed POG as much as possible. For every element E_i in the set of POG_0, we consider the element E_j in the set of constructed POG_i initialized with POG_0.

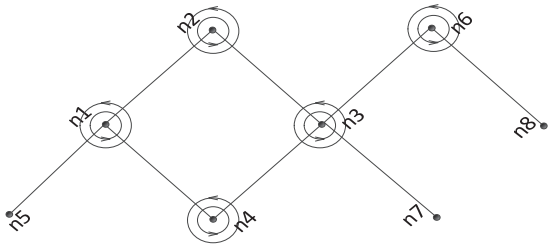
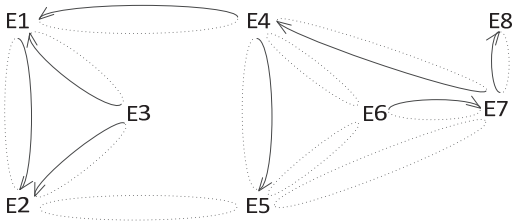


Fig. 2: Weighted Directed Join Graph

	n1	n2	n3	n4	n5	n6	n7	n8
n1	0	E1	0	E2	E3	0	0	0
n2	0	0	E4	0	0	0	0	0
n3	0	0	0	E5	0	E6	E7	0
n4	0	0	0	0	0	0	0	0
n5	0	0	0	0	0	0	0	0
n6	0	0	0	0	0	0	0	E8
n7	0	0	0	0	0	0	0	0
n8	0	0	0	0	0	0	0	0

(a) The result of first Enumeration



(b) The result of second Enumeration

Figure 3: Enumeration by CreatePOJ algorithm

Fig. 3: Weighted Directed Join Graph

If E_i and E_j meet the conditions with EN (E_i) equal to SN (E_j) or SN (E_i) equal to EN (E_j), and not including common node in addition to connecting join node, this shows edge E_i causes a large cost benefit to E_j or the opposite E_j to E_i. We can connect E_i and E_j to product bigger POG. And E_i and E_j are respectively appended to POJUsed and POGUsed. For example, consider E_{41} and E_{12} in Figure 4. EN (E_{41}) and SN (E_{12}) all equal to one. And the remaining node is E_4 except common node E_1 for E_{41}. The remaining node is E_2 except common node E_1 for E_{12}. So E_{41} and E_{12} meet the connection condition, we can construct bigger POG E_{412} by combining E_{41} and E_{12}.

```
CreatePOG Algorithm
Input: all constructed POJ of G = (V, E) by
CreatePoj algorithm
Precondition: create a new data structure POG for
storage Partial Order Graph of G
Out: emits all POGs of G
Initialize POG0=POJ;POJUsed= Φ ;POGUsed= Φ
          HPOJ= Φ ; i=0;
While POG0≠ Φ
 //constructing  partial order graph of POGi+1
 for EI∈POG0
  for EJ∈POGi
   if EN(EI)=SN(EJ) and (Edges(EI)-EN(EI))
     ∩(Edges(EJ)-SN(EJ))= Φ
      append partial order graph <(Edges(EI)-
      EN(EI))∪Edges(EJ)> to POGi+1
      append EI to POJUsed
      append EJ to POGUsed
   else if SN(EI)=EN(EJ) and (Edges(EI)-SN(EI))
     ∩(Edges(EJ)-EN(EJ))= Φ
      append partial order graph <(Edges(EJ)-
      EN(EJ))∪Edges(EI)> to POGi+1
      append EI to POJUsed
      append EJ to POGUsed
 //adjust the collection of POG0
 if(i=0)
  append POG0-POJUsed to HPOJ
  delete POG0-POJUsed from POG0
 else
   delete POG0-POJUsed from POG0
   delete POGUsed from POGi
 i++;
Put HPOJ to POG0
```

The second phase is adjustment phase for the set POG_0 and current set POG. When POG_i is the set of POG_0, we put the set of POG_0 minus POJUsed to HPOJ and delete these elements from POG_0. This is because these seed elements can not be used to product bigger POG. Otherwise, the unused seed elements are deleted from POG_0. And the used elements of POGUsed are deleted from POG_i, because the POGUsed elements in POG_i can been replace by the bigger POG in POG_{i+1}.

The two phases continue until no more beneficial edges can be found.

Let us consider an example for CreatePOG algorithm. Figure 4 is the execution result of CreatePOG algorithm based on Figure 3. CreatePOG algorithm accepts input parameter of the constructed set POJ$\{E_{12}\ E_{32}\ E_{31}\ E_{41}\ E_{45}\ E_{67}\ E_{74}\ E_{78}\}$, and uses this set to initialize parameter of POG_0. Based on POG_0, POG_1 is constructed, that is $\{E_{312}\ E_{412}\ E_{741}\ E_{745}\ E_{674}\ E_{678}\}$. Because E_{32} is not used

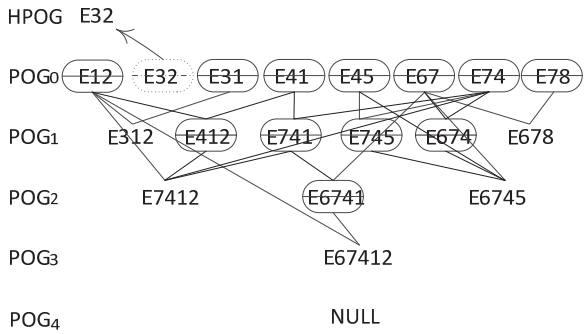


Fig. 4: Weighted Directed Join Graph

during this processor, E_{32} is put to the set HPOG, and deleted from POG_0. Based on the adjusted POG_0 and POG_1, we can achieve $POG_2\{E_{7412}\ E_{6741}\ E_{6745}\}$ in the first phase. In the second phase, we respectively adjust the set POG_0 and set POG_1 to $\{E_{12}\ E_{67}\}$ and $\{E_{312}\ E_{412}\ E_{674}\ E_{678}\}$. POG_3 can be achieved by set POG_0 $\{E_{12}\ E_{67}\}$ and POG_2 $\{E_{7412}\ E_{6741}\ E_{6745}\}$. When the adjustment is made, the set POG_0 is transformed into $\{E_{12}\}$ and $POG_2\{E_{7412}\ E_{6745}\}$. The set POG_0 and POG_3 deduce the null set POG_4. And then the set POG_0 is cleared. It is the end of a while-loop, and HPOJ is put to the set POG_0.

4 Parallel Top-down Enumeration using Partial Orders

In this section, we will use the constructed HPOG and POG to build the best query plans according to the parallelize top-down enumeration. In order to realize the parallel algorithm for the top-down enumeration, we firstly need allocate the set of {HPOG, POG} to different threads. We use the number of cores, num(cores), to denote the number of threads. Every local thread, in turn, obtains the supplementary node set of item allocated it and builds the best query plan of the supplementary node set.

CreatePlans algorithm shows the process with pseudocode. CreatePlans algorithm firstly appends every POJ included the set {HPOG, POG} to set POJ_s, and reallocate the POJ_s to array SSM according to num (cores). Secondly, num (cores) threads parallel implement the corresponding set of SSM. The local thread firstly constructs the complementary set for every allocated POJ, and builds the best query plan using the function of BestPlan based on top-down enumeration algorithm. Lastly, best query plan for POJ and the corresponding best query plan of complementatry set can be achieved by using BestPlan function. CreatePlan Algorithm finally uses MergeAndPrune Plans function to achieve the global optimal solution.

```
CreatePlans Algorithm
Input: all constructed HPOG and POGs of G by
CreatePOG algorithm
Out: emits the optimal query plan
Put every POJ in HPOG an POG to the set POJs
SSM←ReallocateSearch(POJs,m)
for i←1 to m   //m thread parallel implement POJs
  pool.SubmitJob
    for every POJ CurrPlan in SSM[i]
     construct CurrPlan's complementary node set
     PN
     ComPlan=BestPlan(PN)
     PL=BestPLan(ComPlan, CurrPlan )
     PR=BestPlan(CurrPlan , ComPlan)
     if cost(PL) < cost(PR)
       Put PL to LocalTreadBestPi
     else
       Put PR to LocalTreadBestPi
  pool.Sync()
MergeAndPrunePlans(MEMO,{LocalTreadBestP1
...LocalTreadBestPm });
Return MEMO
```

5 Performance Analysis

Type	Enumeration Style	
Top-down DP	the top-down transformational search of Volcano/Cascades	TD_VC
	top-down join enumeration algorithm optimal with respect to the join graph	TD_OJP
	optimized top-down enumeration based constructed POG	TD_POJ
Bottom-up DP	Parallel Size-Driven	PDPsva
Iterative DP	Randomized query	IDP

Table 1: Table I Experimental Parameters

In the following, we will present the experimental findings of our algorithm. All the experiments were performed on a Windows Vista PC with two Intel Xeon Quad Core E540 1.6GHz CPUs (=8 cores) and 8GB of physical memory. Each CPU has two 4Mbyte L2 caches, each of which is shared by four cores. The experimental parameters and their values are illustrated by Table I.

We ran several experiments to evaluate the different algorithms under different settings. The goals of out experiments are to show that our algorithm is an efficient polynomial time method, which is suitable for optimizing complex queries. Due to lack of space, we selected the two typical experiments.

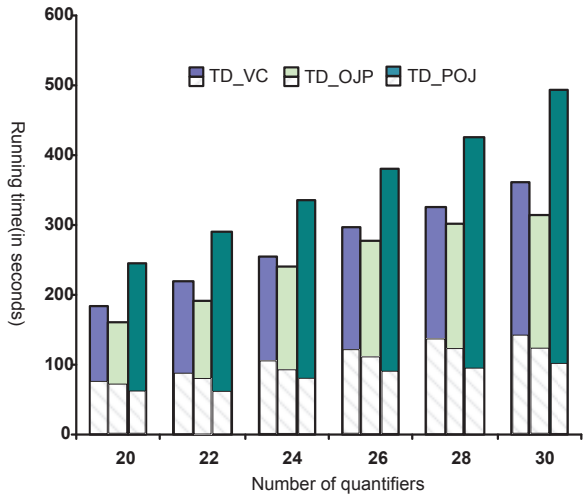
In the first experiment, we compares the running time of TD_VC, TD_OJP and TD_POJ algorithms by changing the number of quantifiers for varying query graphs in Figure 4. The algorithms listed above which all constructed based on top-down join enumeration method. The running time consists of two parts, optimization time used to construct query plan and execution time for query plan. Execution time reflects the quality of constructed query plan. We want to answer that besides clique queries the algorithm optimized top-down enumeration TD_POJ based constructed POJ significantly outperforms the conventional TD_VC and TD_OJP algorithms.

Figure 4 (a) compares the running time for clique queries. As illustrated in Figure 4 (a), the total running time increases as the number of relations is increased. TD_VC and TD_OJP have the same execution time because they are exhaustive search DP algorithms and can construct the best query plan. TD_OJP is optimal with respect to the join graph in the process of building query plans, and it can avoid constructing logical join pairs which are not connected. So the optimization time for TD_OJP is longer than TD_VC. TD_POJ algorithm has the shortest optimization time due to using the constructed partial order graph. However, it cannot guarantee an optimal query plan. So the execution time is the longest. Figure 4 (b) compares the running time for star queries. Figure 4 (c) compares the running time for cycle queries. Figure 4 (b) and Figure 4 (c) show similar experiments with Figure 4 (a).

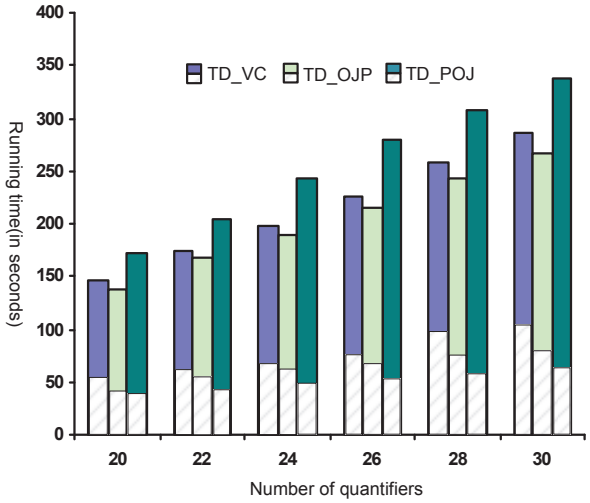
Figure 4 shows that TD_POJ algorithm is better than TD_VC and TD_OJP. This shows the optimization based partial order graph is effective in query plans construction for clique, star and cycle queries.

In the second experiment, we compared TD_POJ, PDP_{sva} and IDP algorithms in Figure 5. By Figure 5 we want to answer that TD_POJ algorithm based on the partial order graph POG significantly outperforms the PDP_{sva} and IDP algorithms.

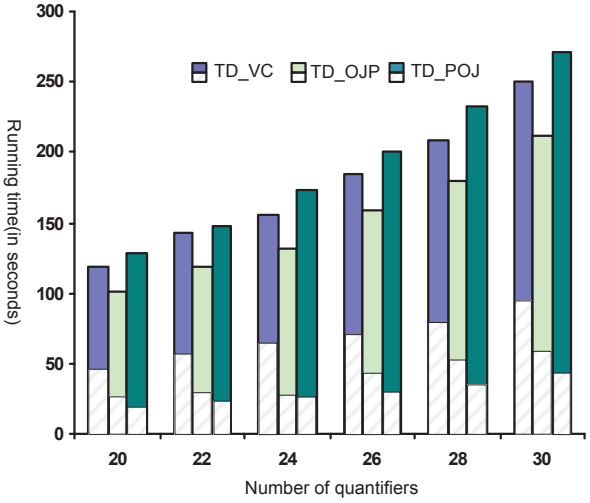
Figure 5 (a) compares the running time for clique queries. As illustrated in Figure 5 (a), the optimization time for IDP is the shortest. However it does not take advantage of the special characteristics of the query, the quality of constructed query plan is not high. So the execution time of output query plan is the longest. PDP_{sva} algorithm in Figure 5 (a) has the shortest execution time because it is exhaustive search DP algorithms and can construct the best query plan. However it constructs join pairs by exhaustive method in the process of building query plan, the optimization time of this algorithm is the longest. The optimization time of TD_POJ is longer the IDP, and shorter than PDP_{sva} in Figure 5 (a). The execution time of TD_POJ is longer the PDP_{sva}, and



(a) Total time for clique queries

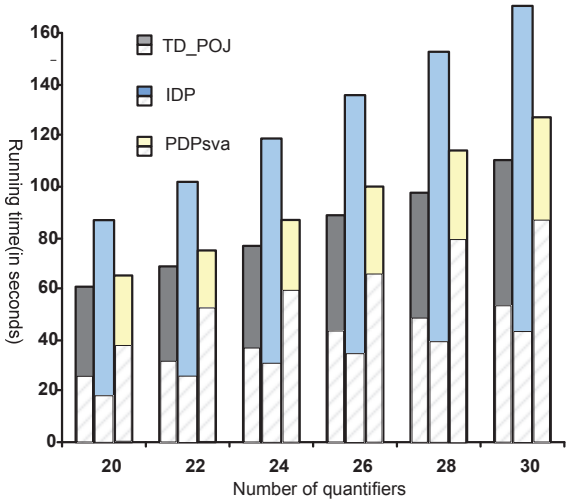


(b) Total time for star queries

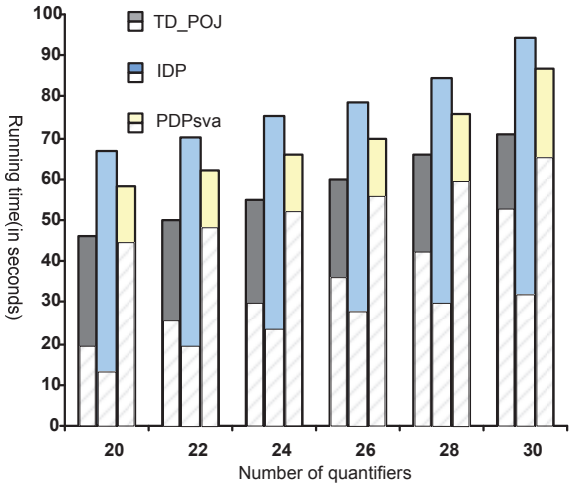


(c) Total time for cycle queries

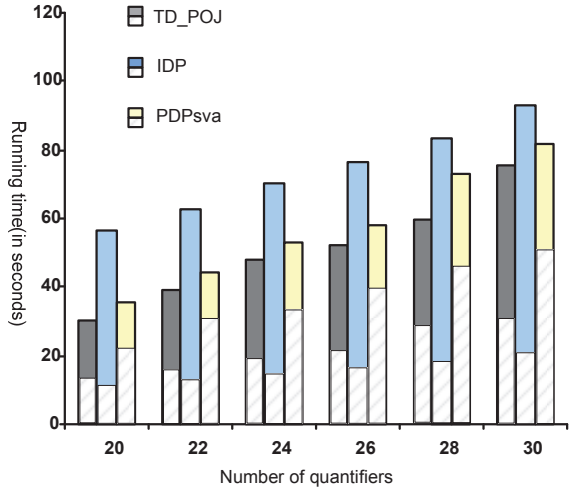
Fig. 5: Weighted Directed Join Graph



(a) Total time for clique queries



(b) Total time for star queries



(c) Total time for cycle queries

Fig. 6: Weighted Directed Join Graph

shorter than IDP in Figure 5 (a). However it should be noted, overall, the total time of TD_POJ is the shortest. Figure 5 (b) compares the running time for star queries. Figure 5 (c) compares the running time for cycle queries. Figure 5 (b) and Figure 5 (c) show similar experiments with Figure 5 (a).

Figure 5 shows that TD_POJ algorithm is optimal for clique, star and cycle queries.

6 Summary

In this paper, parallelizing top-down dynamic programming query based on CMP is completed by three phases. In the first phase, we need consider the relationship between edges within query graph to transforms undirected query to Weighted Edge Join Graph. In the second phase, based on WEJG we use CreatePOJ and CreatePOG algorithms to construct the partial order join and partial order graph. Finally, CreatePlans Algorithm is applied to solve the global solution using POGs achieved as a result of the second phase in parallel. By implementing our framework and analyzing the experiment results, OTDP_CJP gracefully degrade the complexity degree for top-down join enumeration with large number, impressive gains in the performance. Future work is still needed in expanding our algorithms to examine performance on other multithreaded processors and to support other operations.

Acknowledgments

This work is supported by science and technology project of fujian provincial education department: No. JA13196; the National Natural Science Foundation of China 61303131;The National Natural Science Foundation of China under Grant No. 60373099, No. 60973040; important science research project of jilin No.20130206051GX.

References

[1] P. Stenstrom. Proc.of the IPDPS, (2007).
[2] BB Pal, BN Moitra, U Maulik, A goal programming procedure for fuzzy multiobjective linear fractional programming problem, Fuzzy Sets and Systems, 139, 395-405 Elsevier (2003)
[3] Balkesen, C. Proc.of the ICDE(2013).
[4] PL Combettes, JC Pesquet, Proximal splitting methods in signal processing, Fixed-Point Algorithms for Inverse Problems in Science and Engineering Springer Optimization and Its Applications, 185-212 (2011)
[5] Spyros Blanas,Yinan Li, Jignesh M. Patel.Proc.of the SIGMOD,(2011).
[6] Arun N. Swami and Anoop Gupta.Proc.of the SIGMOD, (1998).
[7] S Blanas, JM Patel. Proc.of the SIGMOD, (2011).
[8] Goetz Graefe. Proc.of the CSRD, (2013).
[9] W.-S. Han, W. Kwak, J. Lee, G. M. Lohman, and V. Markl. Proc.of the VLDB, (2008).
[10] G. Moerkotte and T. Neumann. Proc.of the VLDB, (2008).
[11] G. Moerkotte and T. Neumann. Proc.of the VLDB, (2009).
[12] Leonard D. Shapiro, David Maier, Paul Benninghoff, Keith Billings, Yubo Fan, Kavita Hatwal, Quan Wang, Yu Zhang, Hsiao min Wu, and Bennet Vance. Proc.of the IDEAS, 2005.
[13] David DeHaan. Proc. of the ACM SIGMOD international conference on Management of data, (2007).
[14] Donald Kossmann and Konrad Stocker. Proc.of the ACM Trans. on Database Systems, (2000).
[15] Qiang Zhu, Yingying Tao, and Calisto Zuzarte. Knowl. Inf. Syst, 8 (2006).
[16] V.V. Meduri, http://scholarbank.nus.edu.sg/bitstream/handle /10635/2099/report.pdf?sequence=1, (2011).
[17] Meduri Venkata Vamsikrishna. Knowl. Inf. Syst, 8 (2011).
[18] John W. Raymond and Peter Willett. Journal of Computer-Aided Molecular Design, 8 (2004).
[19] Bertrand Cuissart and Jean-Jacques Hebrard. Proc.of the GBRPR, 8 (2005).
[20] AM Wazwaz, Fredholm Integral Equations, Linear and Nonlinear Integral Equations, Springer 119-173 (2011)

YongHeng Chen was born in Heilongjiang of China in Dec 1979 and received the Ph.D. degree at the Department of Computer Science and technology, Jilin University. His current main research interests include Query Optimization, Web Intelligence and Ontology Engineering and Information integration. He is a member of System Software Committee of China's Computer Federation. More than 20 papers of him were published in key Chinese journals or international conferences, 10 of which are cited by SCI/EI.

Kerui Chen received the PH.D. degree in Computer and Science from JiLin University. She currently is a lecturer in the School of Computer and Information Engineering, Hennan University of Economics and Law. Her currently research interests include Web Intelligence, Ontology Engineering and Information integration.

Yaojin Lin received the PH.D. degree. His research interests include data mining, granular computing. He has published more than 20 papers in many journals, such as Decision Support Systems, Applied Intelligence, Knowledge-Based Systems, and Neurologizing.