# 基于星型模式的一个多路 top-$k$ join 算法

曹立新 高 宏
(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

**摘 要** top-$k$ join 查询返回用户最感兴趣的 $k$ 个连接结果. 近来 top-$k$ join 已经成为一个重要的研究课题,且在 Web 数据库、信息抽取和数据挖掘中均有应用. 星型模式的数据仓库在实际应用中也存在 top-$k$ join 查询,如有时决策者只想查询星型连接结果中他最感兴趣的 $k$ 个. 然而,现有 top-$k$ join 算法不适合星型模式. 为了在星型模式上有效地支持 top-$k$ join 查询,文中提出两类索引并基于这两类索引提出一个适用于星型模式的多路 top-$k$ join 算法. 该算法通过采用一个比现有算法更优的上界和一个剪枝策略获得了更高的效率. 此外,实验也表明文中的算法比现有算法效率更高.

**关键词** 数据仓库;星型模式;星型连接;top-$k$;多路 top-$k$ join 算法
**中图法分类号** TP311 **DOI 号**: 10.3724/SP.J.1016.2011.01926

## A Multiple Top-$k$ Join Algorithm Based on the Star Schema

CAO Li-Xin GAO Hong
(*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

**Abstract** Top-$k$ join query returns $k$ join results that users are most interested in. Top-$k$ join has become one of the main research issues recently, and it's dominant in many emerging applications, e.g., web databases, information retrieval and data mining. Top-$k$ join query also exists in data warehouse based on the star schema in practical application. For example, sometimes just the top-$k$ join results that the decision maker is most interested in are desirable. However, the current existing algorithms aren't suitable for the data warehouse based on the star schema. In order to efficiently support top-$k$ join query on star schema, we propose two kinds of indices and a multiple top-$k$ join algorithm that is suitable for star schema based on these indices. By using a tighter upper bound than current existing algorithms and a pruning strategy, the algorithm is more efficient than the current existing algorithms. Furthermore, the experiment also shows that the algorithm is more efficient than the current existing algorithm.

**Keywords** data warehouse; star schema; star join; top-$k$; multiple top-$k$ join algorithm

## 1 引 言

top-$k$ join 查询返回用户最感兴趣的 $k$ 个连接结果. 近来 top-$k$ join 已经成为一个重要的研究课题,且在 Web 数据库、信息抽取和数据挖掘中均有应用.

在数据仓库应用中,星型模式是最常用的数据表示模型,包括一个事实表和多个维表. SSBM[1] (Star Schema Benchmark)数据集中,存在星型模式,如图 1 所示:事实表 LINEORDER 和 4 个维表 CUSTOMER、SUPPLIER、PART 和 DATE. 在基于星型模式的 OLAP(Online Analytical Processing)查

收稿日期:2011-07-18;最终修改稿收到日期:2011-08-07. 本课题得到国家自然科学基金(60933001)资助.**曹立新**,男,1986 年生,硕士研究生,主要研究方向为数据仓库、top-$k$ 算法. E-mail: caolixin11@sina.com. **高 宏**,女,1966 年生,教授,博士生导师,主要研究领域包括并行数据库、无线传感器网络.

询中，涉及最多的操作就是维表和事实表的连接，又被称为星型连接. 星型连接返回连接的全部结果，是 OLAP 查询中代价最高的操作之一.

星型模式的数据仓库的实际应用中也存在 top-$k$ join 查询. 如有时决策者只想查询星型连接结果中他最感兴趣的 $k$ 个. 例如，在图 1 所示的星型模式中，决策者只想查看那些订单时间距今较短、供应商地址距离公司较近、客户信誉较好、配件品牌较好、订单采购量较大的 $k$ 个订单，其对应的查询如例 1 所示.

**例 1.**　SELECT *
FROM Lineorder $F$, Customer $C$, Supplier $S$, Part $P$, Date $D$
WHERE $F$.custkey $=C$.custkey and $F$.suppkey $=S$.suppkey and $F$.partkey $=P$.partkey and $F$.orderdate $=D$.datekey
ORDER BY $F.s_{\text{quantity}}+C.s_{\text{credit}}+S.s_{\text{address}}+P.s_{\text{brand1}}+D.s_{\text{time}}$
STOP AFTER $k$;

对于该查询，一个直观的做法是先生成星型连接的结果，再对这些结果排序得到上述的 top-$k$ join 查询结果. 由于星型连接的结果极多，故此方法效率极低. 但如果我们在数据仓库中创建一些索引并应用 top-$k$ join 算法，那么无需产生连接的全部结果就能得到 top-$k$ join 查询结果. 故基于星型模式的 top-$k$ join 问题的研究具有重要意义. 更重要的是，我们发现现有 top-$k$ join 算法因没有充分考虑星型模式的固有特点而不适合星型模式，因而本文提出一个基于星型模式的多路 top-$k$ join 算法.

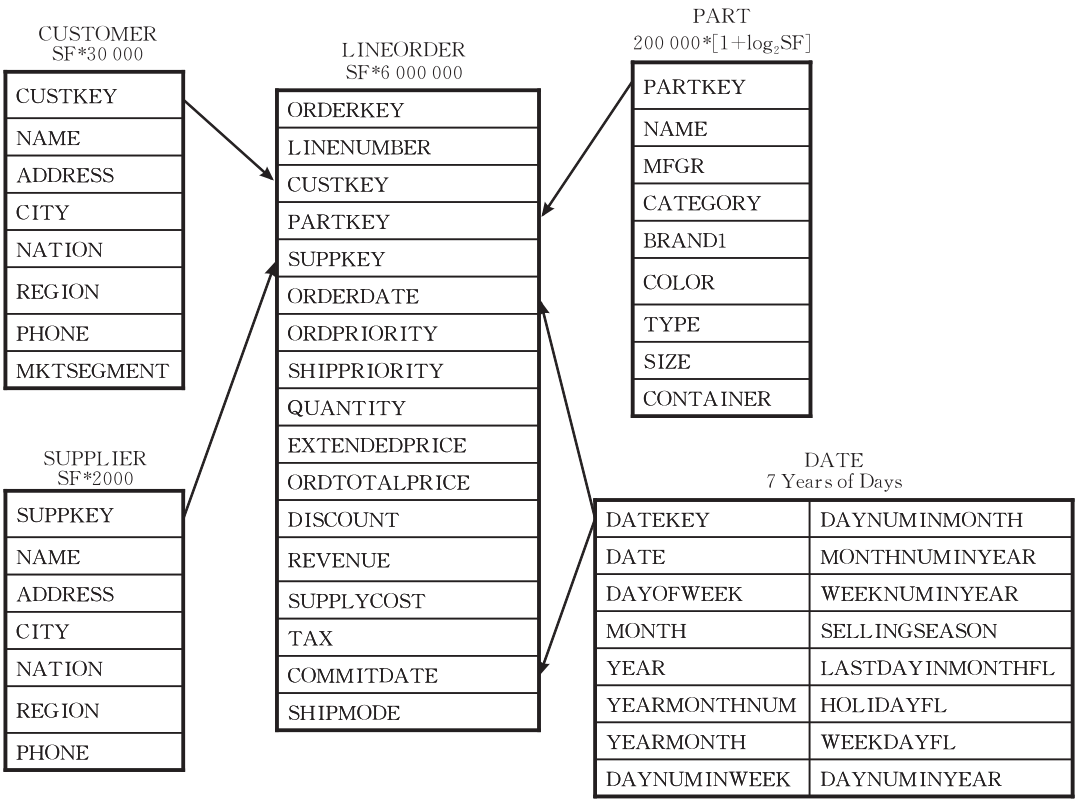


图 1　SSBM 数据集中的星型模式

本文的主要贡献如下：

(1) 为了有效地支持星型模式上的 top-$k$ join 查询，本文提出了两类索引；

(2) 基于这两类索引，本文提出了一个适用于星型模式的多路 top-$k$ join 算法. 该算法通过采用一个比现有算法更优的上界和一个剪枝策略获得了更高的效率.

本文第 2 节介绍相关工作；第 3 节介绍问题定义以及相关索引的数据结构；第 4 节给出基于星型模式的多路 top-$k$ join 算法；第 5 节通过实验验证该算法的效率；第 6 节对全文进行总结.

## 2　相关工作

通常，top-$k$ join 算法中有如下 3 个假设：(1) 每个元组包含若干打分属性(score attribute). 打分属

性为度量某种语义的属性，如一个关于酒店信息的关系表中，可能含有若干分别表示酒店的卫生程度、交通便利程度等的打分属性.（2）数据访问模型(data access model)分为数据有序访问模型和(或)数据随机访问模型.数据有序访问是指数据可按照打分属性值的大小被顺序访问；数据随机访问是指数据的任何一个元组可以被随机访问.（3）算法的打分函数一般只考虑线性单调函数.

top-$k$ join 问题已经有很多的研究成果，相关工作按照输入数据的确定性、数据访问模型种类、打分函数的性质、连接的类型、输出结果的精确性可以分成若干类.不失一般性，本文的研究算法以确定数据作为输入，数据访问模型为一个表只支持有序访问，其它表同时支持有序访问和随机访问，打分函数取线性单调函数，只考虑等值连接并输出精确结果.

rank-join[2]、PBRJ[3]和 FPRA[4]均以确定数据作为输入，数据访问模型为大部分表只支持有序访问，小部分表同时支持有序访问和随机访问，打分函数取线性单调函数，只考虑等值连接并输出精确结果.rank-join 是基于关系数据库的多路 top-$k$ join 算法，其一般化算法[2]利用某些表对随机访问的支持以获得更小的上界.PBRJ[3]和 FPRA[4]考虑打分函数在每个表中取多个打分属性作为输入.FPRA 是 PBRJ 的后续改进算法.虽然 FPRA 的 IO 很少，但其计算复杂度非常高.而且当打分函数在每个表只取一个打分属性作为输入时，FPRA 退化为 rank-join.此外，FPRA 有个限制：所有输入必须按照打分函数值的上界降序排列.这也就是说当打分函数在每个表取 $e(e\geq 2)$ 个打分属性作为输入时，FPRA 必须预先知道打分函数.这在 ad-hoc 查询中是不可能的.

和 rank-join 相比，J*[5]考虑的连接条件更宽泛，不局限于等值连接.但在等值连接时，相比于 rank-join，J* 因中间结果集较大而效率较低.

J* e-approx[5]产生近似的 top-$k$ join 结果.OPTU-Topk[6]、MS_Topk[7]是在不确定数据上的 top-$k$ join 算法.

此外，还有一类研究问题被称为 top-$k$ selection，其相当于 top-$k$ join 问题的一种特例.它假设每个表包含的元组都来自同一组对象集合.NRA[8]、TA[8]、CA[8]、Upper/Pick[9]、Mpro[10]都属于这一类问题的算法.

综上所述，rank-join、PBRJ 和 FPRA 这 3 个算法与本文研究的算法相近.FPRA 是 PBRJ 的改进算法，其 IO 虽然很少，但计算复杂度却非常高，而且当打分函数在每个表中取 $e(e\geq 2)$ 个打分属性作为输入时，FPRA 必须预先知道打分函数，这在 ad-hoc 查询中是不可能的.当 $e=1$ 时，FPRA 退化为 rank-join.故本文只考虑将 rank-join 作为对比算法.rank-join 经过一些修改可以适合星型模式，但在实验中我们发现其效率较低.

## 3 问题定义与索引结构

### 3.1 问题定义

**定义 1**(星型模式). $m$ 维的星型模式包含一个事实表 $F$ 和 $m$ 个维表 $D_1, D_2, \cdots, D_m$.其中事实表包含 3 类属性：(1) 主键；(2) 对应维表的外键；(3) 度量属性.维表包含两类属性：(1) 主键；(2) 描述属性——描述维详细信息的属性.维表中的这两类属性在本文中被统称为维属性.

**定义 2**(扩展星型模式). 在 $m$ 维的星型模式中，为了支持其上的 top-$k$ join 查询，对于决策者感兴趣的每个度量属性 $a$ 和维属性 $b$，我们在相应的事实表和维表中引入新属性 $s_a, s_b$，分别度量 $a$ 和 $b$ 语义的分值，称之为打分属性.打分属性值的大小$\in[0,1]$，表示决策者对 $a$ 和 $b$ 的感兴趣程度.打分属性值越大表示决策者对其相应属性值的感兴趣程度越高.本文称上述加入打分属性之后的星型模式为 $m$ 维的扩展星型模式.

打分属性值大小可由决策者指定，也可由相应的度量属性或维属性计算得到.如在 SSBM 数据集中，系统可根据维属性 DATEKEY 的值计算其对应的打分属性值.

基于 $m$ 维的扩展星型模式，本文考虑的打分函数以若干打分属性作为输入项.对应决策者的不同查询，这些打分属性可能来自事实表，也可能来自维表，并且同一个维表或事实表中可取 $e(e\geq 0)$ 个打分属性作为打分函数的输入.当 $e\geq 2$ 时，系统可以先将这 $e$ 个打分属性的打分函数值计算出来，然后将这个函数值看做一个打分属性.这样打分函数在一个维表或事实表中取多个打分属性作为输入的问题就可以转化为取一个打分属性作为输入的问题.所以接下来本文考虑的打分函数的形式为 $Score(s_1, s_2, \cdots, s_n, s_{n+1})$，其中的 $n+1(n\in\{1,2,\cdots,m\})$ 个打分属性分别来自不同的维表和事实表.该打分函数记作 $Score$.例 1 即为以 $sum$ 为打分函数的 top-$k$ join 查询，其中的 $F.s_{\text{quantity}}$、$C.s_{\text{credit}}$、$S.s_{\text{address}}$、$P.s_{\text{brand1}}$、$D.s_{\text{time}}$ 分别代表订单采购量、客户信誉、供应商地址、配件品牌和订单时间对应的打分属性.

为了方便叙述，本文不失一般性地假设打分函数中的打分属性 $s_1, s_2, \cdots, s_n$ 和 $s_{n+1}$ 分别来自 $D_1, D_2, \cdots, D_n$ 和 $F$，称 $n$ 为打分函数中输入的维数. 根据上述描述，本文研究问题的形式化定义如下.

**定义 3**(问题定义). 给定 $m$ 维的扩展星型模式，其上的 top-$k$ join 问题以一个 $n+3$ 元组 $(F, D_1, D_2, \cdots, D_n, Score, k)$ 为输入，输出按打分函数值降序排列的前 $k$ 个连接结果，每一个结果 $\in((F \bowtie D_1) \bowtie D_2) \cdots \bowtie D_n$. 其中：(1) $F$ 支持有序访问，$D_1, D_2, \cdots, D_n$ 支持有序访问和随机访问；(2) 打分函数 $Score$ 的形式为上述的 $Score(s_1, s_2, \cdots, s_n, s_{n+1})$；(3) $0<k\leq|((F \bowtie D_1) \bowtie D_2) \cdots \bowtie D_n|$.

### 3.2 索引结构

显然，一个数据库中支持有序访问和(或)随机访问的表越多，top-$k$ join 算法的打分函数值上界就会越优. 因此，我们尽可能最大化 $m$ 维的扩展星型模式中支持有序访问和(或)随机访问的表的数目.

为了使维表支持有序访问，本文提出一类维信息索引.

**定义 4**(维信息索引). 给定一个 $m$ 维的扩展星型模式，一个打分属性 $s_j(s_j \in D_i)$ 对应一个维信息索引 $DI_i^j$(Dimension information Index). $DI_i^j$ 为一个三元组 $(score, key, count)$，其中：(1) $score$ 表示 $D_i$ 中的打分属性 $s_j$，且 $DI_i^j$ 中元组按 $score$ 值大小降序排列；(2) $key$ 表示打分属性 $score$ 对应的 $D_i$ 主键；(3) $count$ 被称为频率属性，记录 $key$ 在事实表中作为外键出现的频数.

实际应用中，对应决策者在维表中感兴趣的每个打分属性，系统分别建立一个维信息索引. 图 1 中 SUPPLIER 的打分属性 $s_{\text{address}}$ 对应的维信息索引的形式如表 1. 其中 $s_{\text{address}}$ 为 SUPPLIER 中维属性 address 对应的打分属性，用来度量供应商地址距离公司的远近.

**表 1　SUPPLIER 中 $s_{\text{address}}$ 对应的维信息索引**

| $s_{\text{address}}$ | suppkey | *count* |
|---|---|---|
| 0.5735221 | 533 | 4272 |
| 0.5735119 | 227 | 4261 |

同理，为了使事实表支持有序访问，本文提出一类事实表信息索引.

**定义 5**(事实表信息索引). 给定一个 $m$ 维的扩展星型模式，一个打分属性 $s_i(s_i \in F)$ 对应一个事实表信息索引 $FI_i$(Fact information Index). $FI_i$ 为一个 $m+2$ 元组 $(score, key, FK_1, FK_2, \cdots, FK_m)$. 其中：(1) $score$ 表示 $F$ 中的打分属性 $s_i$，且 $FI_i$ 中元组按 $score$ 值大小降序排列；(2) $key$ 表示打分属性 $score$ 对应的 $F$ 主键；(3) $FK_i$ 是 $key$ 对应 $D_i$ 的外键，$i \in \{1, 2, \cdots, m\}$.

实际应用中，对应决策者在事实表中感兴趣的每个打分属性，系统分别建立一个事实表信息索引. 图 1 中 LINEORDER 的打分属性 $s_{\text{quantity}}$ 对应的事实表信息索引的形式如表 2. 其中的 $s_{\text{quantity}}$ 为 LINEORDER 中度量属性 quantity 对应的打分属性；orderkey 和 linenumber 为 LINEORDER 的主键；其它属性为外键.

**表 2　LINEORDER 中 $s_{\text{quantity}}$ 对应的事实表信息索引**

| $s_{\text{quantity}}$ | orderkey | linenumber | custkey | suppkey | partkey | orderdate |
|---|---|---|---|---|---|---|
| 0.5437236 | 357 | 2 | 143 | 1344 | 1890 | 19950718 |
| 0.5424128 | 435 | 1 | 9765 | 1723 | 156390 | 19980821 |

因为本文考虑的打分函数在同一个维表或事实表中只取一个打分属性作为输入. 故为了方便叙述，下文中的 $DI_i^j$ 一律用 $DI_i$ 表示，$FI_i$ 一律用 $FI$ 表示.

由于事实表通常较大，其对应事实表信息索引也较大(一般不能被读入内存)，因此事实表不支持随机访问. 而维表一般较小(在 $scale\ factor=1$ 的 SSBM 数据集中，维表大小不到事实表的 4%)，其对应的维信息索引也较小，所以维信息索引 $DI_1, DI_2, \cdots, DI_m$ 可支持随机访问. 因为 top-$k$ join 算法中需要的打分属性、主键等均在维信息索引中，所以维信息索引支持随机访问也就相当于维表支持随机访问. 通过 $D_i(i \in \{1, 2, \cdots, m\})$ 的主键(记作 $KEY_i$)对 $DI_i$ 随机访问的方法为：

系统对应每一个维信息索引 $DI_i$ 建立一个位置转换表 $POS_i$. 该表用于将 $KEY_i$ 转换为其在 $DI_i$ 中的索引位置(记作 $order_i$). 索引位置表 $POS_i$ 为一个 Hash 表，包括两项：(1) $KEY_i$；(2) $order_i$. 基于位置转换表 $POS_i$，系统可通过 $KEY_i$ 随机访问 $DI_i$ 而得到 $DI_i$ 在 $order_i$ 的详细信息(记作 $DI_i[order_i]$).

## 4 基于星型模式的多路 top-$k$ join 算法

### 4.1 算法相关概念与原理

基于维信息索引和事实表信息索引，本文提出一个适用于星型模式的多路 top-$k$ join 算法：MTJS. 算法采用一个优先级队列(记作 $Q$)来保存算法的中间结果，$Q$ 中最小的打分函数值记作 $S^{\min}$.

当 MTJS 顺序访问事实表信息索引中的一个元组时，就将该元组外键在相应维信息索引中对应

的 $count$ 值减 1. 基于 $count$ 值的更新，我们给出若干定义如下.

**定义 6**(上边界位). 设 $DI_i$ 为 $m$ 维的扩展星型模式上的一个维信息索引，其对应的上边界位 $H_i=\max\{p|DI_i[p].count\neq 0\}$. 其中：(1) $p$ 表示 $DI_i$ 中索引项的位置(简称为索引位置)，且 $p\in[0, Card(DI_i)-1]$. $p$ 越小的项的打分属性值越高，$Card(DI_i)$ 表示 $DI_i$ 中 $key$ 的基数. (2) $DI_i[p].count$ 表示 $DI_i$ 在 $p$ 的频率属性值.

**定义 7**(维最高分). 设 $DI_i$ 为 $m$ 维的扩展星型模式上的一个维信息索引，其对应的维最高分 $DI_i.max=DI_i[H_i].score(i\in\{1,2,\cdots,m\})$. 其中 $DI_i[H_i].score$ 表示 $DI_i$ 在 $H_i$ 处的打分属性值.

MTJS 通过顺序访问 $FI$ 得到元组 $t$ 并随机访问相应的 $DI_i$ 来构成连接结果. 由于 $DI_i$ 中 $count$ 值为零的元组在将来不会再与顺序访问 $FI$ 得到的元组构成连接结果，所以 $DI_i.max$ 为当前未见连接结果在 $DI_i$ 上的最高可能打分属性值.

**定义 8**(维上界). $DI_i$ 中 $p$ 处的维上界 $upperbound(p,i)=Score(FI.latest, DI_i[p].score, DI_1.max,\cdots,DI_{i-1}.max, DI_{i+1}.max,\cdots,DI_n.max)$. 其中：(1) $FI.latest$ 表示 MTJS 对 $FI$ 最近一次顺序访问得到的打分属性值；(2) $p$ 表示 $DI_i$ 中的索引位置；(3) $DI_i[p].score$ 表示 $DI_i$ 在 $p$ 处的打分属性值；(4) $DI_j.max(j\in\{1,2,\cdots,n\}$ and $j\neq i)$ 表示 $DI_j$ 的维最高分.

**定义 9**(下边界位). 设 $DI_i$ 为 $m$ 维的扩展星型模式上的一个维信息索引，其对应的下边界位 $L_i=\min\{p|upperbound(p,i)>S^{\min}\}$. 其中：(1) $upperbound(p,i)$ 表示 $DI_i$ 中 $p$ 处的维上界；(2) $S^{\min}$ 为中间结果中最小的打分函数值.

根据上述定义，我们有如下定理.

**定理 1.** 给定 $m$ 维的扩展星型模式，设 $t$ 为 $FI$ 中元组，$FK_i$ 为 $t$ 对应 $DI_i$ 的外键，$order_i$ 为 $DI_i$ 中主键为 $FK_i$ 元组的索引位置，在 MTJS 算法中，如果存在 $order_i>L_i(i\in\{1,2,\cdots,n\})$，则 $t$ 一定不是最终结果之一.

证明. 根据定义 8，$DI_i$ 中 $order_i$ 处对应的维上界 $upperbound(order_i,i)=Score(FI.latest, DI_i[order_i].score, DI_1.max,\cdots, DI_{i-1}.max, DI_{i+1}.max,\cdots, DI_n.max)$.

其中因为 $FI$ 中被顺序访问过的元组不会再构成新的连接结果，故 $FI.latest$ 表示当前未见连接结果在 $FI$ 的最高可能打分属性值，又因为 $DI_j.max$ 表示当前未见连接结果在 $DI_j$ 的最高可能打分属性值，进而根据打分函数的单调性可知 $upperbound(order_i,i)$ 是 $DI_i$ 中 $order_i$ 处的打分属性在当前所构成连接结果的打分函数值上界. 再根据 $L_i$ 定义和 $order_i>L_i$ 有 $upperbound(order_i,i)\leq S^{\min}$，即 $t$ 构成的连接结果的打分函数值上界小于或等于 $S^{\min}$，故 $t$ 一定不是最终结果之一. 证毕.

本文基于定理 1 提出一个剪枝策略：当 MTJS 顺序访问 $FI$ 得到一个元组 $t$ 时，如果 $t$ 对应的某个外键在其维信息索引中的索引位置大于该外键所在维信息索引的下边界位，则 MTJS 可以不必随机访问维信息索引以生成 $t$ 对应的连接结果并计算该连接结果的打分函数值. 根据定理 1，该剪枝策略不会剪去任何一个最终结果，故其是正确的. 设 $L_i$ 处频率属性值为 $c$，维数为 $m$，那么根据定理 1，只要 $L_i$ 增加 1，就最多可能有 $c\times m$ 次随机访问和 $c$ 次打分函数值计算被剪枝. 而维信息索引中的频率属性值一般较大，故该剪枝策略具有重要意义.

top-$k$ join 算法中，通常有一个上界用于表示当前未见连接结果的打分函数值上界，本文称之为总上界.

**引理 1.** MTJS 的总上界为 $Score(FI.latest, DI_1.max,\cdots,DI_n.max)$.

证明. 因为 $FI.latest, DI_1.max,\cdots,DI_n.max$ 均为当前未见连接结果在 $FI, DI_1, DI_2,\cdots, DI_n$ 的最高可能打分属性值，故根据打分函数的单调性，$Score(FI.latest, DI_1.max,\cdots,DI_n.max)$ 是 MTJS 的总上界. 证毕.

**定理 2.** 在 $n$ 个维信息索引 $DI_1, DI_2\cdots, DI_n$ 上，MTJS 按定义更新 $L_i$、$H_i$ 和 $Q$. 只要某时刻有一个上、下边界位满足 $L_i<H_i$，MTJS 就可以结束并返回正确的 top-$k$ join 结果.

当 $L_i<H_i$ 时，由下边界位定义可知 $upperbound(H_i,i)\leq S^{\min}$(式(1)). 其中 $upperbound(H_i,i)=Score(FI.latest, DI_i[H_i].score, DI_1.max,\cdots, DI_{i-1}.max, DI_{i+1}.max,\cdots, DI_n.max)$，而 $DI_i[H_i].score=DI_i.max$，故 $upperbound(H_i,i)=Score(FI.latest, DI_1.max,\cdots, DI_n.max)$(式(2)). 由引理 1，$Score(FI.latest, DI_1.max,\cdots, DI_n.max)$ 为 MTJS 的总上界. 于是根据式(1)，式(2)有 MTJS 的总上界 $\leq S^{\min}$，故此时 MTJS 可以结束，并且当前 $Q$ 中的 $k$ 个连接结果为正确结果. 证毕.

### 4.2 算法描述

MTJS 的主要思想是：首先，算法根据维信息索引中的频率属性值来更新上边界位，从而使总上界在维信息索引上对应的维最高分得到更新. 其次，根

据定理 1，算法利用下边界位过滤不必要的随机访问和计算. 最后根据定理 2，当存在某个上边界位大于其对应的下边界位时，算法结束.

MTJS 分为两步；第 1 步初始化相关变量；第 2 步计算 top-$k$ join 的结果. 第 2 步中，MTJS 首先顺序访问事实表信息索引得到一个元组，并更新该元组外键在相应维信息索引中对应的频率属性值；接下来重新确定上、下边界位；然后依据定理 1 判断是否可以剪枝；如果不能剪枝，则算法生成该连接结果并考虑其能否成为中间结果. 如此下去，直到存在一个维信息索引的下边界位小于其上边界位时算法结束.

为了避免在每次更新频率属性值之后都重新计算每一个下边界位对应的维上界和判断能否更新各个下边界位，MTJS 中引入最佳维的概念.

**定义 10**(最佳维). 如果 $DI_{\text{best}}$ 在该时刻位于其下边界位处的维上界在所有维信息索引中最小，称 best 为某时刻的最佳维.

MTJS 只需考虑更新 $DI_{\text{best}}$ 中 $L_{\text{best}}$ 处对应的维上界和 $L_{\text{best}}$，原因如下：(1) 在被选为最佳维时，最佳维下边界位 $L_{\text{best}}$ 处的维上界最小，故 $L_{\text{best}}$ 将来被更新的可能性最大，进而 MTJS 可获得较好的剪枝效果；(2) 不考虑其它下边界位的更新，算法也可正确结束. 因为如果某时刻 MTJS 的总上界$\leq S^{\min}$(式(1))，那么一定有 $L_{\text{best}}<H_{\text{best}}$，进而根据定理 2，算法可正确结束. 假设如果此时 $L_{\text{best}}$ 不小于 $H_{\text{best}}$，那么有 $upperbound(L_{\text{best}},\text{best})\leq$MTJS 的总上界(式(2))，故可由式(1)、式(2)推出 $upperbound(L_{\text{best}},\text{best})\leq S^{\min}$(式(3)). 而根据下边界位定义和式(3)，$L_{\text{best}}$ 会一直更新至 $H_{\text{best}}-1$，故假设失败，算法此时可正确结束.

MTJS 的输入为一个四元组$(k,G,P,Score)$，称之为扩展星型模式下 top-$k$ join 的输入四要素. 其中：(1) $k$ 为查询结果大小；(2) $G$ 为 top-$k$ join 查询相关的维信息索引和事实表信息索引的集合；(3) $P$ 为对应 $G$ 的位置转换表集合；(4) $Score$ 为 2.1 节中的打分函数. 为了方便下面的算法描述，本文接下来不失一般性地假设 $G=\{DI_1,DI_2,\cdots,DI_n,FI\}$，$P=\{POS_1,POS_2,\cdots,POS_n\}$. MTJS 的详细内容如下.

**算法 1.** MTJS.

输入：扩展星型模式下 top-$k$ join 的输入四要素

输出：$((FI\bowtie DI_1)\bowtie DI_2)\cdots\bowtie DI_n$ 中打分函数值最高的前 $k$ 个连接结果

第 1 步. 初始化.

1. 初始化相关数据结构：$Q,DI_1,DI_2,\cdots,DI_n,FI,POS_1,POS_2,\cdots,POS_n$；
2. for each $i(1\leq i\leq n)$
   $\{H_i=0;L_i=Card(DI_i)-1;DI_i.max=DI_i[0].score;\}$
3. while($Q.size()\leq k$) //先生成 $k$ 个连接结果
   { 顺序访问 $FI$ 得到下一个元组 $t$；
   for each $i(1\leq i\leq n)$
   $\{order_i=POS_i(t.FK_i)$；/* $t.FK_i$ 表示 $t$ 中的外键 $FK_i$，$order_i$ 为 $t.FK_i$ 在 $DI_i$ 中对应的索引位置 */ $DI_i[order_i].count=DI_i[order_i].count-1$；
   }//end for
   读取 $DI_1[order_1].score,\cdots,DI_n[order_n].score$ 和 $t$ 对应的 $score$ 用来计算 $t$ 对应连接结果的打分函数值；$Q.push(t$ 对应的连接结果$)$；}
   //end while
4. 根据定义计算 $S^{\min},H_i,DI_i.max,FI.latest,L_i$；
5. $best=Min()$；//$Min()$返回当前的最佳维

第 2 步. 计算 top-$k$ join 结果.

6. if ($L_1<H_1$ or $L_2<H_2$ or $\cdots$ or $L_n<H_n$)
   //终止条件判断
   then 返回 $Q$ 中 top-$k$ join 结果；end if
7. 顺序访问 $FI$ 得到下一个元组 $t$，更新 $FI.latest$；
8. for each $i(1\leq i\leq n)$
   $\{order_i=POS_i(t.FK_i)$；
   $DI_i[order_i].count=DI_i[order_i].count-1;\}$
9. for each $i(1\leq i\leq n)$ //更新上边界位
   { while($DI_i[H_i].count=0$ and $L_i\geq H_i$)
   $\{H_i++;DI_i.max=DI_i[H_i].score;\}\}$
10. 更新最佳维在 $L_{\text{best}}$ 的维上界 $upperbound(L_{\text{best}},\text{best})$；
11. if($upperbound(L_{\text{best}},\text{best})\leq S^{\min}$ and $L_{\text{best}}\geq H_{\text{best}}$) then while($upperbound(L_{\text{best}},\text{best})\leq S^{\min}$ and $L_{\text{best}}\geq H_{\text{best}}$) $\{L_{\text{best}}--;\}$//更新下边界位 $best=Min()$；end if
12. if ($order_1>L_1$ or $order_2>L_2$ or $\cdots$ or $order_n>L_n$) then goto 6；end if//剪枝判断
13. 读取 $DI_1[order_1].score,\cdots,DI_n[order_n].score$ 和 $t$ 对应的 $score$ 用来计算 $t$ 对应连接结果的打分函数值；
14. if($t$ 对应连接结果的打分函数值$>S^{\min}$) then $\{Q.pop();Q.push(t$ 对应的连接结果$);$ $S^{\min}=Q.top();\}$end if
    //$Q.top()$为中间结果中的最小打分函数值
15. goto 6.

### 4.3 算法分析

(1) 若在 $m$ 维的扩展星型模式上利用上述索引应用 rank-join 的一般化算法，其总上界大于 MTJS 的总上界. Rank-join 的总上界为以 $FI.latest$ 和每个维信息索引的第一个打分属性值 $DI_i.top$ 作输入

得到的打分函数值：$Score(FI.latest, DI_1.top, \cdots, DI_n.top)$. 因为 $DI_i.max$ 会随着频率属性值的更新而不断更新，而 $DI_i.top$ 固定不变且 $DI_i.top \geq DI_i.max$，所以根据打分函数的单调性，rank-join 的总上界大于 MTJS 的总上界. 因为总上界更小，故和 rank-join 相比，MTJS 具有更少的顺序访问次数.

(2) 参考 Theorem3[2] 关于 rank-join 近似最优(instance optimal[8])的证明，不难得出 MTJS 也是近似最优的. 近似最优的定义如下.

**定义 11**(近似最优). "设 $\mathcal{A}$ 是一类算法，$\mathcal{D}$ 是一类数据库，对应算法 $A \in \mathcal{A}$ 和数据库 $D \in \mathcal{D}$，算法代价函数 $cost(A,D)$ 返回在 $D$ 上应用 $A$ 导致的代价，称一个算法 $B$ 在 $\mathcal{A}$ 和 $\mathcal{D}$ 上是近似最优的，如果 $B \in \mathcal{A}$，并且对于任意 $A \in \mathcal{A}$ 和任意 $D \in \mathcal{D}$，$cost(B,D)=O(cost(A,D))$ 成立. 即对于任意 $A \in \mathcal{A}$ 和任意 $D \in \mathcal{D}$，存在常数 $c, c'>0$，使 $cost(B,D) \leq c \times cost(A,D)+c'$. $c$ 被称为最优比(optimality ratio)"[8].

**定理 3.** 设 $\mathcal{D}$ 是 $m$ 维的扩展星型模式，包括 1 个只支持有序访问的事实表，$m$ 个既支持有序访问又支持随机访问的维表，$\mathcal{A}$ 是一类算法，其在 $\mathcal{D}$ 上产生正确的 top-$k$ join 结果. $cost(A,D)=SumCost(A,D)$，则 MTJS 在 $\mathcal{A}$ 和 $\mathcal{D}$ 上是近似最优的.

其中，算法代价函数 $SumCost(A,D)$ 度量算法 $A(A \in \mathcal{A})$ 在 $D(D \in \mathcal{D})$ 上对 $FI$ 顺序访问和对 $DI_i(i \in \{1,2,\cdots,n\})$ 随机访问的总代价. 下面用 $C_S(A,D)$，$C_R(A,D)$ 分别表示 $A$ 在 $D$ 上应用导致的对 $FI$ 的顺序访问次数和对 $DI_i$ 的随机访问次数. 因为随机访问速度(内存读写速度)大约是顺序访问速度(磁盘读写速度)的 $10^3$ 倍，所以我们令 $SumCost(A,D)=C_R(A,D) \times 0.001+C_S(A,D)$. 虽然 0.001 很小，但是因为 $C_R(A,D)$ 很大且打分函数值的计算次数和 $C_R(A,D)$ 成正比，故 $SumCost$ 中不能忽略 $C_R(A,D)$. 下面给出定理 3 的详细证明.

证明. 设 $A$ 为 $\mathcal{A}$ 中任意算法. 如暂不考虑 $C_R(A,D)$ 的代价，参考 Theorem3[2] 的证明，显然 MTJS 的最优比至多为 1，故有 $C_S(\text{MTJS},D) \leq C_S(A,D)+c'$(式(1))成立. 本文考虑 MTJS 中的一次顺序访问在 $D$ 中至多对应 $m$ 次随机访问，故 $SumCost(\text{MTJS},D) \leq (1+0.001 \times m) \times C_S(\text{MTJS},D)$(式(2)). 再根据式(1)有 $(1+0.001 \times m) \times C_S(\text{MTJS},D) \leq (1+0.001 \times m) \times C_S(A,D)+c'$(式(3))，又因为 $C_S(A,D) \leq SumCost(A,D)$，故 $(1+0.001 \times m) \times C_S(A,D)+c' \leq (1+0.001 \times m) \times SumCost(A,D)+c'$(式(4))成立，综合式(2)、式(3)和式(4)可知 $SumCost(\text{MTJS},D) \leq (1+0.001 \times m) \times SumCost(A,D)+c'$，即 MTJS 是近似最优的且最优比为 $1+0.001 \times m$. 证毕.

## 5 实验结果与分析

### 5.1 实验配置

**数据集.** 因为 top-$k$ join 算法一般只访问每个表的前一部分就能结束，所以数据集的大小不作为其实验参数之一[4]. 本文使用 scale factor 为 1 的 SSBM 数据集作为实验数据集. 对应 $m$ 维的扩展星型模式，我们在 SSBM 数据集上做了一个修改：在每个维表和事实表中分别引入一个打分属性. 按照打分属性值分布的不同，本文的实验数据集分为指数分布的数据集和正态分布的数据集，简称为指数分布数据和正态分布数据. 指数分布中 $\lambda=2$、正态分布中 $\mu=0.6, \sigma=0.2$. 此外，实验考虑了 Bruno 等人[9] 在 ICDE2002 中提出的相关系数 $cf$(correlation factor)对算法性能的影响. 按照 $DI_i(i \in \{1,2,\cdots,n\})$ 与 $FI$ 的相关系数的不同，指数分布数据和正态分布数据又分别对应一组实验数据集. 相关系数的定义如下.

**定义 12**(相关系数). 设 $L$ 是按打分属性 $L_s$ 降序排列的表，$R$ 是按打分属性 $R_t$ 降序排列的表，$R$ 的外键是 $L$ 的主键，对于 $L$ 任意一个主键 $key$，其在 $L$ 中对应的 $L_s$ 值为 $s$，在 $R$ 中对应的 $R_t$ 值为 $t$，$cf \in [-1,1]$，我们称 $L$ 与 $R$ 正相关，如果 $cf \geq 0$，且 $t \in [\max(0, s-(1-cf)), \min(1, s+(1-cf))]$；称 $L$ 与 $R$ 负相关，如果 $cf<0$，且 $t \in [\max(0, 1-s-(1-|cf|)), \min(1, 1-s+(1-|cf|))]$. $cf$ 被称为 $L$ 与 $R$ 的相关系数.

相关系数的含义为，$key$ 在 $L$ 中打分属性值和 $key$ 在 $R$ 中打分属性值的相关程度. 观察定义 12，当 $cf=0$ 时，$t \in [0,1]$，即 $t$ 的大小与 $s$ 无关. 因为打分属性值高低与排序位置相关，故此时 $key$ 在 $R$ 中的对应元组可以出现在 $R$ 中任意位置. 当 $cf>0$ 时，$s$ 如果越高，$t$ 就可能越高. 如 $s=0.5, cf=0.9$，则 $t \in [0.4, 0.6]$. 此时 $key$ 在 $L$ 排得越靠前，那么 $key$ 在 $R$ 中排得也越可能靠前. 故 $L$ 与 $R$ 正相关时，顺序访问 $R$ 中元组而生成的连接结果的打分函数值会较高(打分属性值高的元组连接打分属性值高的元组)，此时 top-$k$ join 算法相应地会较快结束. 反之，当 $cf<0$ 时，$key$ 在 $R$ 排得越靠前，那么 $key$ 在 $L$ 中排得越可能靠后. 故此时顺序访问 $R$ 中元组而生成的连接结果的打分函数值会较低(打分属性值高的元组连接打分属性值低的元组)，此时

top-$k$ join 算法相应地会结束较慢.

**实验环境**. 主机 CPU：P4 3.4 GHz；磁盘大小：80 GB；磁盘读写平均速度：50 MB/s；操作系统：Windows XP；编译环境：vs2005；内存大小：1 GB.

**测度**. 我们以执行时间(记作 *time*)和 *SumCost* 的函数值为测度衡量实验中算法的性能.

**对比算法**. 为了使 rank-join 适合星型模式的特点，本文将其修改如下(修改后的 rank-join 记作 RJ_Adapt)：(1) rank-join 没有必要顺序访问 $DI_i(i\in\{1,2,\cdots,n\})$，因为顺序访问 $DI_i$ 得到的元组之间不能连接，故会产生若干不完整的连接结果，而这些结果不可能被输出，因为它们只是维信息索引中的一个元组，故其打分函数值下界必定小于或等于 1，而其总上界一般大于 $n$. 此外，顺序访问 $DI_i$ 不能降低其总上界. (2) 由修改(1)，rank-join 没有必要考虑对 $DI_1, DI_2, \cdots, DI_n$ 和 $FI$ 顺序访问的优先次序.

### 5.2　MTJS 与 RJ_Adapt 的性能比较

不失一般性，实验中的打分函数取线性加权函数(其它线性单调函数的实验也表明 MTJS 的效率较高，但由于篇幅所限，这里略去). 权值的赋值可分为两种情况，即或者事实表打分属性的权值较大，或者维表打分属性的权值较大，故我们用后者与前者的比值大小来表示权值的不同赋值，本文称之为相对权值，记作 $W$. 这里我们分别在不同的打分属性值分布、相关系数($cf$)、查询结果大小($k$)、相对权值($W$)与打分函数中输入的维数($n$)下进行实验. 默认设置打分属性值分布为正态分布，$cf=0.5$，$k=100$，$W=1$，$n=4$. 之所以 $DI_i(i\in\{1,2\cdots,n\})$ 与 $FI$ 的 $cf$ 默认取 0.5，是因为一般 $DI_i$ 和 $FI$ 正相关. 例如，如果一个订单的供应商地址距离公司越近，客户信誉值越高，那么一般该订单的采购量也越大. 5 组实验的详细对比结果如下.

#### 5.2.1　打分属性值分布的影响

图 2 比较了 Join、RJ_Adapt、MTJS 3 个算法在打分属性值取不同分布时的性能差异. 图中 Normal 代表正态分布，Exp 代表指数分布. Join 是最直观的做法，先连接再排序得到 top-$k$ join 结果. 由图 2 可知，Join 的效率很低，所以本文在下面的对比实验中忽略此算法. 和 RJ_Adapt 相比，MTJS 的性能更优：在指数分布中，其 *SumCost* 的值有 32%左右的减少，*time* 有 34%左右的减少；在正态分布中，其 *SumCost* 的值有 15%左右的减少，*time* 有 18%左右的减少. 其 *time* 的减少多于 *SumCost* 的值的减少是因为：MTJS 除减少了顺序访问次数和随机访问次数之外，还避免了多次打分函数值的计算，且使用了最佳维等方法.

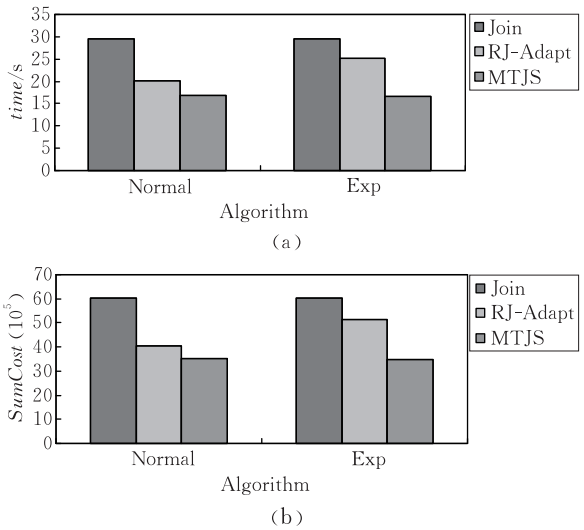


图 2　打分属性值分布对算法性能的影响

#### 5.2.2　相关系数($cf$)的影响.

图 3 比较了 RJ_Adapt、MTJS 在 $DI_i(i\in\{1,2,\cdots,n\})$ 与 $FI$ 的 $cf$ 不同时的性能差异. 当 $cf<0$ 时，因为 $DI_i$ 与 $FI$ 负相关，$H_i$ 对应的维主键一般在 $FI$ 中的索引位置较大，故当顺序访问 $FI$ 的次数较少时，$H_i$ 处的频率属性值一直不能被更新至零. 此时，MTJS 的 $H_i$ 更新得较少，其总上界相应地更新得也较少，所以 MTJS 和 RJ_Adapt 的性能基本持平：如 $cf=-0.5$ 时，和 RJ_Adapt 相比，MTJS 的 *SumCost* 的值大约减少 1%，*time* 大约减少 1%. 反之，当 $cf>0$ 时，MTJS 的 $H_i$ 会更新较多，其总上界相应的更新也较多，故和 RJ_Adapt 相比，MTJS 的性能更优：如 $cf=0.75$ 时，其 *SumCost* 的值大约减少 24%，*time* 大约减少 26%.

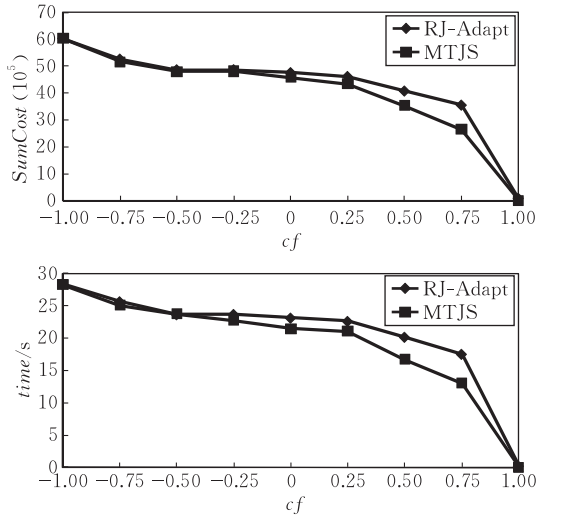


图 3　$cf$ 对算法性能的影响

5.2.3 查询结果大小($k$)的影响：

由图 4 可知，在 $k$($k \leq 10$)较小时，两个算法的性能基本持平. 因为在 $k$ 较小时，两个算法返回得都很快. 此时 MTJS 顺序访问 $FI$ 的次数较少，所以 $H_i$ 更新得较少，其总上界相应地更新得也较少，因而二者的性能差异不大. 如 $k=10$ 时，和 RJ_Adapt 相比，MTJS 的 $SumCost$ 的值有 3% 左右的减少，$time$ 有 4% 左右的减少. 相反，在 $k$ 较大时，MTJS 的 $H_i$ 有了较多的更新，其总上界相应地更新得也较多，所以 MTJS 的性能和 RJ_Adapt 相比有了较大提升. 如 $k=1000$ 时，$SumCost$ 的值有 28% 左右的减少，$time$ 有 30% 左右的减少.

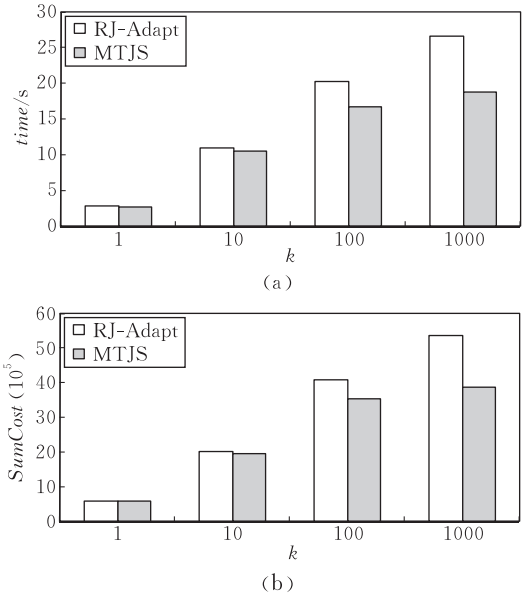


图 4 $k$ 对算法性能的影响

5.2.4 线性加权函数中相对权值($W$)的影响：

图 5 比较了在线性加权函数中相对权值不同时两个算法性能的差异. 这里相对权值 $W$ 分别取五组数据：0.25，0.5，1，2，4. $W$ 较小时，因为事实表打分属性的权值较大，故 MTJS 的总上界大小更依赖 $F.latest$，而 $F.latest$ 与 $FI$ 中当前未见的打分属性值的误差更小，故此时两个算法的总上界都较优，进而两个算法一般都会很快结束. 由于 MTJS 很快结束时，频率属性值更新得较少，上边界位相应地更新得较少，其总上界相应地更新得也较少，故两个算法的性能差异不大. 如 $W=0.5$ 时，和 RJ_Adapt 相比，MTJS 的 $SumCost$ 的值有大约 0.5% 的减少，$time$ 有大约 1% 的减少. 反之，$W$ 较大时，根据上边的分析，两个算法都不会很快地结束. 因而 MTJS 对 $FI$ 的顺序访问次数较多，$H_i$ 随之更新得也较多，其总上界相应地更新得也较多，故此时 MTJS 的性能优势较为明显. 如 $W=2$ 时，和 RJ_Adapt 相比，MTJS 的 $SumCost$ 的值有大约 34% 的减少，$time$ 有大约 36% 的减少.

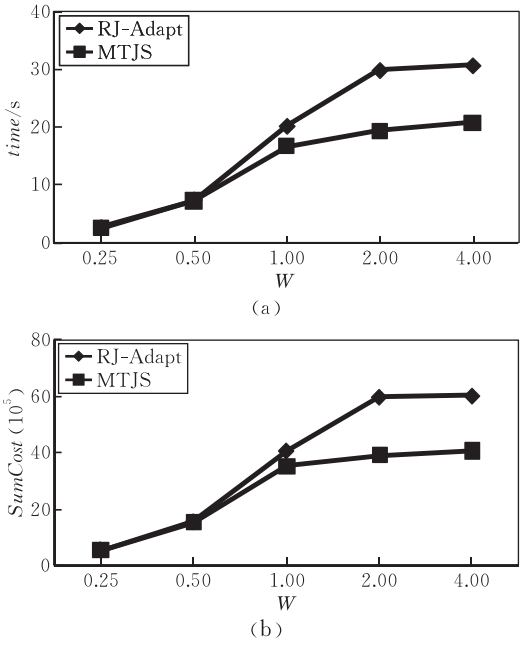


图 5 $W$ 对算法性能的影响

5.2.5 打分函数中输入的维数($n$)的影响：

图 6 比较了在 $n$ 不同时两个算法的性能差异. 为完成 $n=5$ 时的实验，我们在 SSBM 数据集中添加了一个维，$n=6$ 时同理. 当 $n$ 较小时，两个算法的总上界中取最高可能打分属性值的项数都较少，所以它们的总上界都相对较小，故两个算法都较早结束. 因此，MTJS 的 $H_i$ 更新得较少，其总上界相应地更新得也较少，故其和 RJ_Adapt 的性能差异并不大. 如在 $n=3$ 时，和 RJ_Adapt 相比，MTJS 的 $SumCost$ 的值减少了 2.5% 左右，$time$ 减少了 3% 左右. 反之，当 $n$ 较大时，MTJS 的 $H_i$ 更新得较多，其

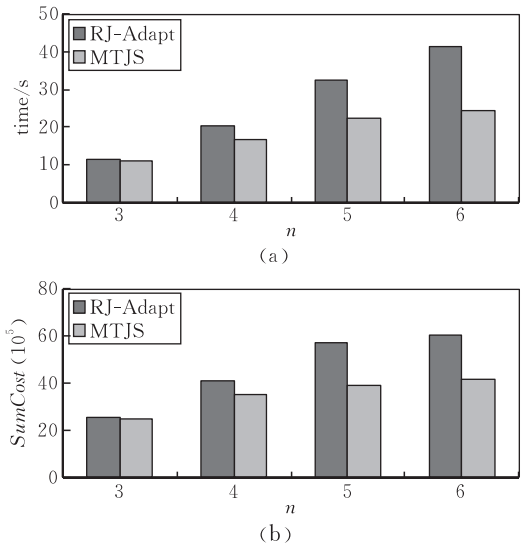


图 6 $n$ 对算法性能的影响

总上界相应地更新得也较多，故其性能优势明显. 如在 $n=5$ 时，和 RJ_Adapt 相比，MTJS 的 *SumCost* 的值减少了 31%左右，*time* 减少了 33%左右.

## 6 结　论

本文考虑基于星型模式的数据仓库上的 top-$k$ join 查询，提出两类索引并基于这两类索引给出一个适用于星型模式的多路 top-$k$ join 算法. 该算法通过采用一个比现有算法更优的上界和一个剪枝策略而获得了更高的效率. 以执行时间和一个算法代价函数的值为测度，我们对本文算法和 rank-join 的一个适应算法进行了若干组对比实验. 这些实验验证了本文算法的性能优于对比算法.

## 参 考 文 献


[1] O'Neil P, O'Neil E, Chen X. The Star Schema Benchmark, 2007. http://www.cs.umb.edu/~poneil/StarSchemaB.pdf

[2] Ilyas I F, Aref W G, Elmagarmid A K. Supporting top-$k$ join queries in relational databases//Proceedings of the International Conference on Very Large Data Bases(VLDB). Berlin, Germany, 2003: 754-765

[3] Schnaitter K, Polyzotis N. Evaluating rank joins with optimal cost//Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS). Vancouver, Canada, 2008: 43-52

[4] Finger J, Polyzotis N. Robust and efficient algorithms for rank join evaluation//Proceedings of the ACM SIGMOD International Conference on Management of Data. New York, USA, 2009: 415-428

[5] Natsev A, Smith J R, Li C S et al. Supporting incremental join queries on ranked inputs//Proceedings of the International Conference on Very Large Data Bases(VLDB). Roma, Italy, 2001: 281-290

[6] Soliman M A, Ilyas I F, Chang K C-C. Top-$k$ query processing in uncertain databases//Proceedings of the International Conference on Data Engineering (ICDE), Istanbul, Turkey, 2007:896-905.

[7] Re C, Dalvi N, Suciu D. Efficient top-$k$ query evaluation on probabilistic data//Proceedings of the International Conference on Data Engineering(ICDE). Istanbul, Turkey, 2007: 886-895

[8] Fagin R, Lotem A, Naor M. Optimal aggregation algorithms for middleware//Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS). Santa Barbara, California, 2001: 614-656

[9] Bruno N, Gravano L, Marian A. Evaluating top-$k$ queries over Web-accessible databases//Proceedings of the International Conference on Data Engineering. San Jose, CA, 2002: 369

[10] Chang K C, Hwang S. Minimal probing: Supporting expensive predicates for top-$k$ queries//Proceedings of the ACM SIGMOD International Conference on Management of Data. Santa Barbara, CA, 2002: 346-357



**CAO Li-Xin**, born in 1986, M. S. candidate. His research interests include data warehouse, top-$k$.

**GAO Hong**, born in 1966, professor, Ph. D. supervisor. Her research interests include parallel database, wireless sensor networks.


**Background**

This work is partially supported by the National Natural Science Foundation of China under grant No. 60933001.

In recent years, top-$k$ join has become one of the main research issues, and it's dominant in many emerging applications, e. g., Web databases, information retrieval and data mining. In data warehouse, star schema is the most commonly used multidimensional data model. Among the star schema's OLAP queries, the largest number of operations is the join between the dimension table and the fact table, also known as star join. Star join returns all the joining results, so it's the most expensive OLAP operation. Top-$k$ join query also exists in data warehouse based on the star schema in practical application. For example, sometimes just the top-$k$ join results that the decision maker is most interested in are desirable. However, the current existing algorithms aren't suitable for the data warehouse based on the star schema. In order to efficiently support top-$k$ join query on star schema, we propose two kinds of indices and a multiple top-$k$ join algorithm that is suitable for star schema based on these indices. By using a tighter upper bound than current existing algorithms and a pruning strategy, the algorithm is more efficient than the current existing algorithms. Furthermore, the experiment also shows that the algorithm is more efficient than the current existing algorithm.