递归查询算法的研究*

怀进鹏

(北京航空航天大学计算机系,北京 100083)

摘要 本文介绍了DeDB的递归查询算法,提出为减少冗余及回溯计算的基本原理.根据该原理,提出了一种高效的递归查询算法GCQA,它包括2部分,一是预编译算法;另一个是递归编译算法.实验结果表明这种算法是高效的.

关键词 人工智能,数据库,递归查询,演绎数据库.

演绎数据库(Deductive Data Base,简称DeDB)是70年代末期出现的一个新课题,它将人工智能与数据库相结合,容数据处理与知识处理于一体,成为当今数据库技术中一个非常重要的发展方向,已引起国内外学者的广泛重视.

70年代末,H. Gallaire等人首先从逻辑角度用一阶谓词的解释及理论描述数据库特性:按解释的观点,查询和完整性限制被视为在真值语义下的解释环境中的求值,它是从语义的角度看待数据库;按理论的观点,查询和完整性限制被视为待证明的定理,它是从语法角度看待数据库.后来R. Reiter与R. A. Kowalski更详尽地研究了这两种方法,R. Reiter分别称为模型论和证明论观点;R. A. Kowalski称之为关系结构观点和逻辑数据库观点,进而可得到逻辑意义下的DeDB模型论或证明论模型.

本文针对DeDB研究中的关键问题——递归查询处理进行了分析与研究,提出了解决冗余计算,提高查询效率的递归查询优化算法GCQA (Goal_driven Compiling Query Algorithm),其基本思想是:用目标查询驱动的编译方法,将查询值分为约束与自由两类,以最大限度地减少冗余计算,并根据匹配(递归)谓词进行查询,以减少回溯.

1 递归查询算法

目前,递归查询方法主要分为2大类:一类是目标驱动的解释方法,又称自上而下方法;另一类是数据驱动的编译方法,又称自下而上方法.

1.1 解释方法

这种方法把DeDB视为一个PROLOG解释程序,它以Robinson的归结原理为基础,通过内涵数据库IDB中规则的合一匹配操作产生对外延数据库EDB的查询请求,以深度

* 本文1991-08-29收到,1992-09-09定稿

本课题得到国家自然科学基金及航空科学基金资助.作者怀进鹏,32岁,副教授,主要研究领域为人工智能与数据库.

本文通讯联系人:怀进鹏,北京100083,北京航空航天大学计算机系

优先原则通过搜索、回溯来实现查询求值.因此查询是在目标驱动下,自上而下地完成查询求值的,避免了求值的盲目性.但一次查询需多次访问外延数据库,因而查询过程中频繁访问外存,系统效率较低.目前典型的方法有:循环的 QSQI 方法和递归的 QSQR 方法[2].

1. QSQI(Interactive Query/Subquery)方法是以循环计算实现递归查询,适用于无算术谓词的、范围受限的集合.该方法首先建立有序偶〈Q,R〉,其中 Q 表示广义查询集,R 表示导出谓词集及其当前值,然后进入循环计算,方法是:

(1) 以 Q 中广义查询查 IDB 并传播常数;

(2) 查 EDB,建立导出谓词的新广义查询;

(3) 查询求值产生新元组,并送到 R 中;

(4) 将新广义查询送 Q 中,转(1),直到〈Q,R〉中无变化为止.

2. QSQR(Recursive Query/Subquery)方法则通过递归运算实现递归查询,也是以有序偶〈Q,R〉的形式统一查询,与 QSQI 相比,有较少的冗余计算,算法及其适用范围与 QSQI 类似.由于该算法在递归运算中需保护环境,从实现效率看,递归方法劣于循环方法,尤其是在空间效率上,需要为每次递归调用保存环境信息,当存在 n 次递归调用时,所需空间与 n 成正比.所以推理次数越多,递归调用过程就越多;而且若递归深度很深,则存贮空间的占用是巨大的;而在循环计算中,只需一个固定的存贮空间,与循环次数无关.但 QSQI 方法中有序偶〈Q,R〉的 Q 中含所有广义查询,原广义查询在下一次循环时被重复计算,所以存在冗余;而 QSQR 方法中每次递归查询均为新产生的广义查询,所以从减少冗余计算上看,QSQR 方法优于 QSQI 方法.

因此,若以循环方法实现递归查询,应尽力减少冗余计算,使循环方法在时空效率上优于递归方法.

1.2　编译方法

编译方法是在查询求值过程中,利用 IDB 产生许多辅助查询作为待证明的子目标,然后通过查询 EDB 实现辅助查询求值,这种方法一次查询可访问多个 EDB,执行效率比解释方法有明显提高;但它是在 EDB 中数据驱动下实现的,所以查询求值过程有很大的盲目性.典型的编译方法有 Naive、Semi－Naive 及 Magic Sets[3]等方法.

1. Naive 方法是典型的数据驱动循环型查询算法,其基本方法是:

(1) 选出与查询求值有关的所有规则;

(2) 为规则中导出谓词建立临时关系;

(3) 用非递归规则求初值,并建立递归规则的循环计算;

(4) 循环计算,直到无新元组产生为止.

2. Semi－Naive 法则是针对 Naive 中存在大量冗余计算,通过比较相邻两次计算产生的元组之间关系,仅计算新元组以减少冗余的一种改进的 Naive 方法,但该方法目前仅在线性递归时实现.

3. Magic Sets 法是通过对给定的规则集进行魔集转换产生与原规则集等价、易于优化的新规则集,其基本思想是通过内涵数据库中规则获取查询的引导信息,从而在自下而上编译求值时,只计算有用元组,以减少相关事实集.但它难以实现.

综上所述,解释方法是目标驱动的,在减少冗余计算与最小化相关事实集方面有明显的

优点，但它频繁访问外存，执行效率较低；编译方法是数据驱动的，较少访问外存，效率较高，但存在大量冗余计算而影响空间效率. 所以若能将这两种查询方法相结合，解决编译中求值的盲目性及解释中频繁访问外存的低效性，并以循环计算实现递归减少冗余计算，最小化相关事实集，则可获得较高效率，这些正是我们设计 GCQA——目标驱动的编译型查询求值算法的基本思想.

2 GCQA 的原理

设查询为？$-P(x_1,\cdots,x_n)$，为叙述方便，对变元采用以下定义，若已知 x_i 值，称 x_i 为约束值，否则称之为自由变元. 设关于 P 的非递归规则进行查询求值结果为 P 的初始集 IS (*Initial Set*)，IS 将作为递归求解初值.

定义 1. 若 IS 中某元素与含约束值的查询 P 合一，则称该元素为合一元素；由 IS 中合一元素组成的集称为 P 的约束集 BS(*Bound Set*)；由 IS 中非合一元素组成的集称为 P 的自由集 FS(*Free Set*).

由于 *GCQA* 是基于 IS 的目标驱动的查询，所以每次递归查询结果一定是 P 的合一元素，因而有：

引理 1. 对？$-P$ 的目标驱动的递归查询结果与 P 合一，即查询值属于 P 的约束集 BS.

证明：直接由定义及目标查询含义得出.

定义 2. 设 $IDB=\{r_1:-l_1,\cdots,r_w:-l_w\}$，则 $E=\{r_1\rightarrow l_1,\cdots,r_w\rightarrow l_w\}$ 称为 IDB 的展开规则集，$r_i\rightarrow l_i$ 称为展开规则.

定义 3. 设 E 为展开规则集，t 是一个项，若对展开规则 $r_i\rightarrow l_i$ 及项 t 的子项 u，存在置换 θ，使 $r_i^\theta=u$，则用 l_i^θ 代替 u 在 t 中出现，得项 t'，记为 $t[u\leftarrow l_i^\theta]$，称 t' 是在 E 下的展开项，记为 $t\rightarrow t'$.

定义 4. 设 S 是内涵及外延关系字典，E 是展开规则集，若 S 中存在项 p，使得对任何 $q\in S(p\neq q)$，p 均不能展开到 q，则称 p 为 S 在 E 下的终止元素.

定义 5. 项 a 的终止展开条件是：a 的展开项 b 中每一子项是：(1)终止元素，或(2)递归谓词，称 b 为 a 的终止展开.

引理 2. ？$-P$ 的终止展开中仅含递归谓词及终止元素.

证明：直接由定义得出.

由引理 2，可通过 E，对？$-P$ 进行展开直到每一子项为递归谓词或终止元素. 设 P 的初始化规则及递归规则集分别为

$P:-f_j(Q_1,\cdots,Q_m)$　　Q_i 与 p 无递归关系

$P:-\Phi_i(P,Q_1,\cdots,Q_m)$　　$(i=1,\cdots,n\quad j=1,\cdots,l)$

定理 3. 设递归查询谓词？$-P$ 的第 $i-1$ 次迭代计算 P 的约束集为 BS，自由集为 FS；第 i 次计算 P 增值为 dBS，则第 $i+1$ 次计算 P 的有效值是基于 $FS\cup dBS$ 求解.

证明：设对 $P:-\Phi_j(P,Q_1,\cdots,Q_m)$ 求值，由于第 $i-1$ 次对 P 查询值为 $FS\cup BS$，第 i 次计算 P 的增值 dBS 是以 $BS\cup FS$ 为初值而求得，记为 $BS\cup FS\overset{i}{\underset{p}{\Rightarrow}}dBS$.

由引理 1 知，dBS 为约束集，则在第 $i+1$ 次迭代计算时，初值为 $BS\cup FS\cup dBS$.

对任意集合 A,由幂等律有 $A \cup A = A$.

所以　$FS \cup BS \cup dBS = (FS \cup BS) \cup (FS \cup dBS) \cup (BS \cup dBS)$　(1)

所以,在 $FS \cup BS \cup dBS$ 下对 P 的查询求值等价于等式(1)右边三个集合下对 P 查询求值集的并,下面分别讨论这三种情况:

(1)因为 $FS \cup BS \overset{i}{\underset{P}{\Rightarrow}} dBS$,所以 $FS \cup BS \overset{i+1}{\underset{P}{\Rightarrow}} dBS$,即初值相同,增值相同

(2)设 $FS \cup dBS \overset{i+1}{\underset{P}{\Rightarrow}} dBS^0$

(3)设 $BS \overset{i+1}{\underset{P}{\Rightarrow}} BS^1$,由(1)知 $BS^1 \subseteq dBS$　(2)

设 $dBS \overset{i+1}{\underset{P}{\Rightarrow}} dBS^1$,由(2)知 $dBS^1 \subseteq dBS^0$　(3)

所以,由式(2)、(3)得,$BS \cup dBS \overset{i+1}{\underset{P}{\Rightarrow}} BS^1 \cup dBS^1 \subseteq dBS \cup dBS^0$　(4)

即在第 $i+1$ 求值(3)中计算的增值属于(1)(2)中求值,且(1)已在第 i 次计算中已求解,故(1)(3)为冗余.

所以,第 $i+1$ 次计算时仅(2)中对 P 的求值是非冗余的,即有效的.　证毕

定理 3 说明以循环求解时,可有效地减少冗余计算,从而提高查询效率.但若能把查询 P 中约束值传播到 P 的终止展开项中,改变从左到右、深度优先带回溯的求值策略:即以含约束值的递归谓词或终止元素为搜索起点进行求解,并继续向其两边传播当前已知值进行查询,则可以大大地减少回溯,避免无效置换,提高查询效率.

定义 6. 设? $-P$ 的终止展开项 L 的子项 t,若 P 的约束值传递到 t 中,则称 t 为匹配谓词 *MP*(*Matched Predicate*).若 t 与 P 是同名递归谓词,且为匹配谓词,则称 t 为匹配递归谓词 *MRP*(*Matched Recursive Predicate*);否则,称 t 为非 *MRP*.

推论 4. 对递归查询? $-P$ 求值时,若终止展开项中含有 *MRP*,则 *MRP* 以 *BS* 为迭代初值求解;而非 *MRP* 以 *FS* 中值求解.

证明:略.

推论 4 是显然的,因为 P 的初值及后继值计算均基于约束集和自由集中,因此,将 P 的查询结果分为约束与自由集对减少冗余计算,改善查询策略,提高查询效率是很有效的.

3　算法设计

GCQA 算法包括 2 部分:

(1)预编译算法:完成查询 P 的展开及初始集 *IS* 的计算;

(2)递归编译算法:以初始集开始计算 P.

预编译算法的基本思想是:① 构造查询谓词的 AND/OR 树,树中叶节点为终止元素及递归谓词;② 自底向上计算不含递归谓词 P 的路径,求 P 的初值 *IS*,并删除该路径.预编译后,得到 P 的初始集和递归 AND/OR 树.

定义 7. 初值可解节点(简称可初解).

(1)外延关系节点是可初解的;

(2)含有或后裔的非叶节点,如果至少有一个后裔可初解,则该非叶节可初解;

(3)含有与后裔的非叶节点,如果所有后裔均可初解,则该非叶节点可初解.

定义 8. 不可解节点.

(1)除目标查询节点外无后裔的内涵关系节点是不可解节点;

(2)含有或后裔的非叶节点,如果所有后裔均不可解,则该非叶节点不可解;

(3)含有与后裔的非叶节点,如果至少有一个后裔不可解,则该非叶节点不可解;

设已知过程 Solved 与 Unsolved 作用在 P 的 AND/OR 树上每条路径. Solved:自下而上对 P 可初解标记,计算 P 的查询值并删除该路经;Unsolved:完成对树的不可解标记,并删除该路径.

预编译算法

输入:目标查询? $-P$,展开规则集 E 及内涵与外延关系字典 D.
输出:P 的初始集 $BS \cup FS$ 与递归 AND/OR 树.
方法:
(1)把 P 存 OPEN 表,若 P 为外延关系,转(9); /* OPEN 中存临时扩展节点 */
(2)把 P 移出 OPEN 表,存 QPT 中,并把 P 后裔存 OPEN 表,建立返回到 P 的指针; /* QPT 存放 P 及其终止展开项 */
(3)if OPEN 为空,then 转(9);
(4)取出 OPEN 表中第一个节点 n,存入 QPT 中;
(5)if n 为外延节点,then 标记初值可解,转(3);
(6)if n 为递归节点,then 转(3);
(7)if n 可扩展,then n 后继节点存 OPEN 表尾,设置返回到 n 的指针;
(8)else n 被标记为不可解节点,调 Unsolved 转(3);
(9)以 P 中约束值对 IS 分类,得 $BS \cup FS$,置 $ptr0$ 为 BS 的栈底,$ptr1$ 为栈顶,设 ptr 为 $ptr1$;
(10)结束.

递归编译算法

输入:P 的初值集 $BS \cup FS$ 和递归 AND/OR 树.
输出:P 的查询结果.
方法:
(1)if QPT 为空,then 结束;
(2)自上而下传播 P 的约束值,并标记其中的叶节点为 MP 或 MRP;
(3)while P 的每条路径 do{
(4) if 存在 MRP,then{
(5) while 产生新元组,do{
(6) 将 MRP 用 BSS 中 $ptr1$ 与 $ptr0$ 间数据求值,并向两边传播查询,而 MRP 在 FST 中求值. /* 栈 BSS 存放 P 的约束集设 $ptr0$、$ptr1$ 的当前查询结果头尾指针;表 FST 存放自由集 */
(7) 自下而上传播查询值,得 dBS 存 BSS 中,置 $ptr1 \Rightarrow ptr0$,$ptr1$ 为新栈顶,转(5)}}
(8) else 把当前查询送查询栈 QS 中{
(9) while QS 不空,do{
(10) 取 QS 栈顶,以 MP 为查询起点求值,并向两边传播查询,直到终止展开项中全部子项求解完;
(11) if 某子项为非 MRP 且是 P 的新查询,then P 以 FS 为初值查询求值;
(12) 自下向上传播查询结果,得 dBS 且存 BSS 中. 置 $ptr0$ 为 $ptr1$,$ptr1$ 为栈顶;
(13) if $ptr1 = ptr0$ then 转(9);
(14) else 把 P 的新产生的查询送入 QS 中}}
(15)将 BSS 中新元素存结果栈,置 $ptr0$ 为 BSS 栈底,$ptr1$ 为 ptr};
(16)将 BSS 内容存入结果栈;
(17)结束.

4 示 例

本节将以家族关系 DeDB 为例来简要说明 GCQA 的查询求值过程.

例 1:设 EDB 中有关系 $Parent(x,y) = \{(a0,a1),(a11,a2),(a1,a2),(a2,a)\ (a,aa),$

(b,aa) (a,ab),(aa,aaa),(aa,aab), (aab,aaab),(aaa,aaaa),(c,ca),(aaaa,aaaaa),(aaaaa,aaaaaa)}

IDB:Ancestor(X,Y):−Parent(X,Y)

Ancestor(X,Y):−Parent(X,Z),Ancestor(Z,Y).

设查询为? −Ancestor(aa,Y),且 * 标记 ptr0,Δ 标记 ptr1.

执行过程:

Ⅰ.经预编译后,得 Ancestor(X,Y)初始集 IS,其中 BS={ * (aa,aaa),(aa,aab)Δ},FS={(a0,a1),(a11,a2),(a1,a2), (a2,a),(a,aa), (b,aa), (a,ab),(aab,aaab),(aaa,aaaa),(c,ca),(aaaa,aaaaa),(aaaaa,aaaaaa)},得递归 AND/OR 树并传播约束值后为:

Ancestor(aa,Y):−Parent(aa,Z)),Ancestor(Z,Y)进入编译循环.

Ⅱ.Parent(aa,Z)为 MP 查 EDB 得 Z={aaa,aab},建立 Ancestor({aaa,aab},Y)新查询,并查 FS 得 Y={aaab,aaaa} .所以 BSS={aa,aaa),(aa,aab), * (aa,aaab),(aa,aaaa)Δ};

Ⅲ.以 Ancestor(aaa,Y)查询,得 Z={aaaa},得 Ancestor(aaa,Y), 查 FS, Y={aaaaa} 所以 BS={…, * (aa,aaaaa)Δ},新查询送 QS 中;

Ⅳ.以 Ancestor(aaaa,Y)查询,得 Z={aaaaa},得 Ancestor(aaaaa,Y),Y={aaaaaa} 所以 BS={…, * (aa,aaaaaa)Δ}, 新查询送 QS 中;

Ⅴ.以 Ancestor(aaaaa,Y)查,得 Z={aaaaaa},并得 Y=空,所以 新查询不送 QS 中,BS={… (aa,aaaaaa) * Δ};

Ⅵ.以 Ancestor(aab,Y)查询,得 Z={aaab},新查询 Ancestor(aaab,Y)中 Y=空,所以新查询不送 QS 中,所以,BS={aa,aaa),(aa,aab),(aa,aaab)(aa,aaaa),(aa,aaaaa),(aa,aaaaaa) * Δ}.

例 2:EDB 与例 1 相同,IDB: Ancestor(X,Y): —Parent(X,Y)

Ancestor(X,Y):−Ancestor(X,Z),Ancestor(Z,Y).

设查询为? −Ancestor(X,aa).

执行过程:

Ⅰ.经预编译后,得 Ancestor(X,Y)初值集 IS,其中 BS={ * (a,aa),(b,aa)Δ},FS={a0,a1),(a11,a2),(a1,a2),(a2,a),(a,ab),(aa,aaa),(aa,aab),…}得递归 AND/OR 树并传播约束值后为: Ancestor(X,aa): −Ancestor(X,Z),Ancester(Z,aa).

Ⅱ.进入编译循环:因为 Ancestor(Z,aa)为 MRP,查 BS 得 Z={a,b}以 Ancestor(X,{a,b})查 FS,得 X={a2},BS={…, * (a2,aa)Δ}.

Ⅲ.在 BS 中查 Ancestor(Z,aa),得 Z={a2},所以 X={a11,a1}.

所以 BS={…, * (a11,aa),(a1,aa)Δ}进入循环.

Ⅳ.在 BS 中查 Ancestor(Z,aa)得 Z={a11,a1},所以 X={a11,a1}

BS={… * (a11,aa),(a1,aa)Δ}进入循环.

Ⅴ.在 BS 中查 Ancestor(Z,aa) 得 Z={a11,a1},所以 X={a0}

所以 BS={… * (a0,aa)Δ}.

Ⅵ.在 BS 中查 Ancestor(Z,aa)得 Z={a0} X=空

所以 $BS=\{(a,aa),(b,aa),(az,aa),(a11,aa),(a1,aa),(a0,aa)\}$ 结束

以上示例说明基于定理3得到的 *GCQA* 算法在减少冗余，提高效率方面是有效的，可适用于线性或非线性规则.

5 结 论

本文提出的目标驱动的编译型递归查询算法 GCQA 汲取了解释与编译查询求值的优点，克服了解释中低效性及编译求值的盲目性. 本文经过深入分析，提出了一种解决冗余计算的新方法，并提出了以匹配(递归)谓词为查询起点的求值策略，从而大大减少了回溯和无效置换，提高了递归查询求值效率. 目前，本文工作已在我们的 DQL/DDBMS[5] 中实现.

本算法虽然在减少冗余和回溯计算方面有显著特点，但尚有待于进一步完善，有不足之处，敬请各位专家、学者指教.

参考文献

1 Chang C. On the evaluation of queries containing decried relations in RDB. Advances in DB Theory Plenum Press NY, 1981.

2 Bancilhon F, Ramakrishan R. An amateurs introduction to recursive query processing strategies. Proc. SIGMOD Int. Con. on Management of Data, Washington D. C., May 1986.

3 Bancilhon F *et al*. Magic sets and other strange ways to implement logic programs. Proc. 5th ACM SIGMOD—SIGACT Sym. on Principles of DB System, 1986.

4 Vieille L. Recursive axioms in DDB: the query /subquery approach. Proc. 1st Int. Con. on Expert DB System, Charleston, 1986.

5 怀进鹏. 演绎 DB 研究与实现. 北京航空航天大学学报, 1991(1).

RESEARCH ON RECURSIVE RUERY ALGORITHMS

Huai Jinpeng

(*Department of Computer Science, Beijing University of Aeronautics and Astronautics, Beijing* 100083)

Abstract The basic concept and recursive query algorithms in DeDB are introduced. And an efficient recursive query algorithm, GCQA, has been proposed, which combines the bottom—up with the top—down and includes two parts: one is pre—compiling algorithm; the other is recursive compiling algorithm. The experimental result has shown that the algorithm is very effective.

Key words Artificial intelligence, data base, recursive query, deductive data base.