doi:10.3969/j.issn.1000-436X.2012.10.005

基于循环策略和动态知识的 deep Web 数据获取方法

鲜学丰[1,2]，崔志明[1,2]，赵朋朋[1,2]，梁颖红[2,3]，方立刚[2,3]

（1. 苏州大学 智能信息处理及应用研究所，江苏 苏州 215006；
2. 江苏省现代企业信息化应用支撑软件工程技术研发中心，江苏 苏州 215104；3. 苏州市职业大学，江苏 苏州 215000）

摘 要：针对目前 deep Web 数据集成在数据获取方面存在代价大和查询选择效率低等问题，提出了一种基于循环策略和动态知识的 deep Web 数据获取方法，该方法根据同领域数据源之间的关联关系，提出使用循环策略分多次完成数据源的数据获取，同时利用集成系统已获取的数据动态构建知识，并设计了基于集成系统动态知识的查询选择方法。与现有方法比较该方法能降低数据获取的代价，提高查询选择的准确性。实验结果表明，该方法有效地提高了 deep Web 数据集成的数据获取效率。

关键词：deep Web；数据集成；数据获取；动态知识；查询选择

中图分类号：TP392 **文献标识码**：A **文章编号**：1000-436X(2012)10-0035-09

Approach to deep Web data acquiring based on circular strategy and dynamic knowledge

XIAN Xue-feng[1,2], CUI Zhi-ming[1,2], ZHAO Peng-peng[1,2], LIANG Yin-hong[2,3], FANG Li-gang[2,3]

(1. The Institute of Intelligent Information Processing and Application, Soochow University, Suzhou 215006, China;
2. JiangSu Province Support Software Engineering R&D Center for Modern Information Technology Application in Enterprise, Suzhou 215104, China;
3. Suzhou Vocational University, Suzhou 215000, China)

Abstract: Concerning on the acquisition problems on deep Web data integration such as high cost and low efficiency of query selecting, a novel deep Web data acquiring method was proposed based on circular strategy and dynamic knowledge. According to the relationship between deep Web data sources from the same domain, the circulation strategic acquisition of data source in batches was applied in such method, as well as a designed method of query selecting on dynamic knowledge based on integrated systematic. Compared with current, the method reduces the acquisition cost and with more accuracy. Experimental results show that the method can raise the acquisition efficiency of deep Web data integration.

Key words: deep Web; data integration; data acquisition; dynamic knowledge; query selection

1 引言

目前主流搜索引擎还只能搜索 Internet 表面可索引的信息，在 Internet 深处还隐含着大量通过主流搜索引擎少量或无法涉及的海量信息，这些信息称之为深层网页(deep Web，又称为 invisible Web 或

收稿日期：2012-02-20；**修回日期**：2012-09-10

基金项目：国家自然科学基金资助项目（60970015，61003054，61170020，41201338）；江苏省自然科学基金资助项目（BK2012164）；江苏省高校自然科学研究基金资助项目（10KJB520018）；江苏省现代企业信息化应用支撑软件工程技术研发中心开放基金资助项目（SX201102，SX201105）

Foundation Items: The National Natural Science Foundation of China (60970015, 61003054, 61170020, 41201338); The Natural Science Foundation of Jiangsu (BK2012164); The Natural Science Foundation of Jiangsu College(10KJB520018); Jiangsu Province Support Software Engineering R&D Center for Information Technology Application in Enterprise(SX201102, SX201105)

hidden Web)[1]。Deep Web 的信息一般存储在服务器端的 Web 数据库中，与静态页面相比通常信息量更大、主题更专一、信息质量和结构更好。为了方便用户快捷高效地使用 deep Web 信息，国内外学者对 deep Web 数据集成进行了广泛的研究[2~4]。Deep Web数据集成的一种方案是与构建传统搜索引擎一样，将 deep Web 数据库中的内容爬取出来，存储到本地数据库中并建立索引，它能在最短时间内响应用户的查询要求[5]。目前，这种方案在许多特定领域已成为 deep Web 数据集成研究的主流。由于集成系统可能需要集成数十个甚至更多的 deep Web 数据源，因此，该方案中一个关键并十分有挑战性问题是如何高效地获取 deep Web 数据。

目前，deep Web 数据集成的实现方法为：首先独立穷尽获取每一个待集成的 deep Web 数据源，然后通过数据清洗、实体识别、合并去重等步骤完成获取数据的集成。这种实现方法在数据获取方面主要存在 2 个缺陷：1) 每个数据源数据获取的后期代价十分巨大，花费较大的代价仅仅获取极少的新数据，同时数据集成时需要处理来自不同数据源的大量重复数据，数据集成的代价也非常巨大；2) 每个数据源的数据获取独立进行，爬虫主要依据该数据源已获取数据的统计信息进行查询选择[6~8]，由于统计信息缺乏和查询候选池有限，该方法存在查询选择的准确性较差、数据获取覆盖率较低等问题。

经研究发现，集成系统中待集成的数据源之间并不是相互独立的，而是相互关联。数据源之间数据相互覆盖，甚至一些数据源之间相互依赖，例如：现实世界一些商业数据源彼此分享数据。根据这种关联关系，进一步研究发现同领域 deep Web 数据源之间具有 2 个重要特征。特征 1：在集成环境中，从某一数据源获取的数据，可能从另一个或一些待集成的数据源中获取，因此从某一数据源数据获取后期获取的数据，可能出现在另一个或一些数据源数据获取的前期或中期；特征 2：同领域的数据源之间具有相似的属性值并且这些属性值也具有相似的分布特征。基于上述发现本文提出一种基于循环策略和动态知识的数据获取方法。该方法根据特征 1 提出使用循环策略分多次完成数据源的数据获取，当获取某一数据源的效率下降到某一阈值时，停止该数据源的数据获取，爬虫开始获取下一个数据源的数据，依次类推直到把所有待集成数据源都获取一遍；然后再重复上述过程，直到所有待集成数据源都已达到结束获取条件。该方法使一部分应该从一些数据源数据获取后期获得的，从另一些数据源数据的前期或中期获得。与传统一次性穷尽数据获取方法相比该方法能减少数据源后期的数据获取，降低了数据获取的代价，同时也能减少重复数据的获取，降低了数据集成的代价。根据特征 2 提出利用集成系统已获取的数据动态构建知识，并设计基于集成系统动态知识的查询选择方法。该方法丰富了查询选择的知识，提高了查询选择的准确性，同时扩展了查询候选池，提高了数据获取的覆盖率。结合循环策略和动态知识进行数据获取时，对于每个数据源可以多次利用丰富后的集成系统动态知识进行查询选择能有效率提高查询选择的准确性，从而提高数据获取的效率。

事实上，不同领域的数据源之间也可能存在一定的关联关系，对存在关联关系的不同领域的数据源进行数据获取时，仍可以使用本文提出的基于循环策略和动态知识的数据获取方法提高数据获取的效率。但是由于同领域数据源之间的关联关系一般强于不同领域的数据源，所以同领域 deep Web 数据集成时使用本文提出方法的效率更高。

2 相关研究

在 deep Web 数据获取方面，Barbosa L 等人[6]第一次提出使用已获取数据中最高词频关键词作为下一个查询词的查询选择方法，然后，在实际中最高词频的查询词并不一直能返回更多的新记录，并且由于这种方法产生的查询词具有较高的关键词依赖，将会产生大量重复记录，增加数据获取的代价。Wu Ping 等人[7]通过查询选择高效的爬取 deep Web 数据，将结构化 Web 数据库看成一个属性—值图模型，将 Web 数据库爬行问题转化为图遍历问题，以最少的查询提交次数获取尽量多的 deep Web 内容，其问题在于每一轮查询结果都要扩充到属性—值图中用于产生下一轮待提交的查询词，这种做法代价非常高。Google 也提出了一种多领域跨语言的 deep Web 数据获取方法[9]，将 deep Web 内容爬取出来 Surface 化，将爬取出来的内容放进 Google 索引中，这样用户就可以通过 Google 搜索到部分 deep Web 内容。目前，Google 每秒钟可以爬取上千个 deep Web 动态页面，其不足在于该方法不能应用到 Post 方法提交的表单上，并且只获取 URL 而不进一步处理。Lu Jiang 等人[10]提出一种基于强化学

习的 deep Web 爬取框架，该框架爬虫作为 Agent，Web 数据库被作为环境。Agent 感知当前状态，并根据 Q-value 选择一个查询提交到 Web 数据库。最近，George V 等人[1]针对不需要获取查询所有结果的特定应用，提出一种 Rank-aware Hidden Web 爬取方法，该方法对所有潜在查询仅仅下载 top-k 个结果，从而避免了完全爬取巨大的 deep Web 数据。综上所述，这些研究主要针对如何提高单个 deep Web 的数据获取效率，没有考虑集成环境下进行数据获取时，deep Web 数据源之间的关联关系对数据获取的影响。

3　Deep Web 数据获取相关概念定义

Deep Web 数据获取方式：结构化的 Web 数据库可看作一张关系数据表 DB，DB 包含的数据记录为 $T=\{t_1,t_2,\cdots,t_n\}$，每条记录包含 m 个属性 $A=\{a_1, a_2,\cdots,a_m\}$。获取 deep Web 中的数据只能通过从查询接口上提交查询，从返回结果页面获得 deep Web 中包含该查询的记录集，对于一个潜在的查询 q_i，$R(q_i)$表示在 DB 上执行查询 q_i 所返回的记录集，即 DB 中所有包含 q_i 的记录集合（假设不考虑返回记录限制的情况），$R(q_i)$为 T 的一个子集。

Deep Web 数据获取代价模型：爬虫在 DB 上执行查询 q_i 和获取 q_i 所返回的记录集都需要一定的代价，如时间、网络带宽等。对于一个查询 q_i，使用 $Cost(q_i, DB)$表示爬虫在 DB 上执行查询 q_i 和获取 q_i 所返回记录集的各种代价总和（即 deep Web 数据获取代价）。对于结构化的 Web 数据库，数据获取的代价主要包括：爬虫提交查询到站点的查询代价、爬虫与 Web 服务器交互的代价、爬虫下载结果页面的代价。交互次数和查询提交次数是不一样的，每个结果页面通常包含固定 k 个匹配的数据记录，每次初始连接得到至多 k 个数据记录。例如，在图书数据库中有 104 个图书记录匹配属性值“书名，Java”并且每个结果页面显示 10（k=10）个数据记录，则获取所有结果记录集的总交互次数为[104/10]=11 次。即每获取下一页的数据记录，都需要和 Web 服务器交互一次。

定义爬虫提交一次查询的代价为 C_q，爬虫与 Web 服务器交互一次的代价为 C_m，爬虫下载一个结果页面数据记录的代价为 C_d。对于一个查询 q_i，在 DB 上执行查询 q_i 和获取 q_i 所返回记录集的各种代价总和 $Cost(q_i, DB)$可表示为

$$Cost(q_i, DB) = C_q + C_m M + C_d P \tag{1}$$

其中，C_q、C_m、C_d 为常量，M 为爬虫与 Web 服务器交互次数，P 为爬虫需下载的结果页面数量。

$$M = \begin{cases} \dfrac{num(q_i, DB)}{k}, \text{其他} \\ \dfrac{N}{k}, \text{存在最大返回记录} \end{cases} \tag{2}$$

其中，$num(q_i, DB)$为 DB 中所有匹配 q_i 的数据记录数，k 为一个结果页面最多可显示记录数，如果 DB 存在最大返回记录限制，则 N 为 DB 一次查询的最大返回记录数。爬虫需下载的结果页面数量 P 和爬虫与 Web 服务器交互次数 M 相等。

单个 deep Web 数据源的数据获取：对于一个 deep Web 数据源 DB_k，deep Web 数据获取问题可形式化定义为：寻找一组查询关键词集合 $Q_k=\{q_1, q_2,\cdots, q_n\}$使得 $P(q_1 \vee q_2 \vee \cdots \vee q_n, DB_k)$值最大，其约束条件是 $\sum_{i=1}^{n} Cost(q_i, DB_k) \leqslant t$，其中，$t$ 为爬虫获取 DB_k 中数据可使用的最大代价。对于一个给定的查询 q_i，$P(q_i,DB_k)$表示在 DB_k 上执行查询 q_i 所返回的结果记录数占 DB_k 总记录数的比例。

面向 deep Web 数据集成的数据获取：对于一个集成系统 I，$S=\{DB_1, DB_2, \cdots, DB_n\}$为 I 待集成的所有 deep Web 数据源的集合，面向 deep Web 数据集成的数据获取可形式化定义为：需找一组查询关键词集合 $Q=\{Q_1,Q_2,\cdots,Q_n\}$使得 $P(Q_1 \vee Q_2 \vee \cdots \vee Q_n)$最大，其约束条件是 $\sum_{i=1}^{n} Cost(Q_i, DB_i) \leqslant T$，其中，$T$ 为集成系统 I 的可使用的最大代价，Q_i 为获取第 i 个数据源所提交的查询集合 $Q_i=\{q_1,q_2,\cdots,q_n\}$，$P(Q_i)$为 $P(q_1 \vee q_2 \vee \cdots \vee q_n, DB_i)$。

4　Deep Web 数据集成的数据循环获取策略

4.1　数据循环获取策略

对于一个集成系统 I，$S=\{DB_1,DB_2,\cdots,DB_n\}$为 I 待集成的所有 deep Web 数据源的集合。针对传统的 deep Web 数据集成实现方法在数据获取方面存在的缺陷，本文基于同领域数据源之间的关联关系，提出使用循环策略分多次完成数据源的数据获取，该策略主要思想为：首先对 S 中的数据源，根据其可能对集成系统 I 贡献的效用大小进行排序，效用评价标准可以根据数据源的大小、数据源的数据质量等，或者由这些量组成的一个函数。然后从 S 中

排在第一位的数据源开始进行数据获取，数据获取的策略是当正在进行数据获取的当前数据源的特定特征达到阈值（特征和阈值将在停止条件进行详细描述），则停止获取该数据源，根据达到阈值的特征判断该数据源是继续保持在 S 中等待下一次获取，还是从 S 中删除该数据源，结束该数据源的获取任务；然后爬虫开始获取下一个数据源的数据，依次类推把 S 中的所有数据源都获取一遍；再重复上述过程，直到 S 为空，S 中的数据源都达到获取结束条件。循环获取的具体算法如下。

Input：$S=\{DB_1,DB_2,\cdots,DB_n\}$； // S 为待集成数据源

$T=\{t_1,t_2,\cdots,t_n\}$; // T 为数据源的最大代价集合

θ; //θ为一个查询获取新数据的效率阈值

ω; //ω为获取数据源的数据量阈值

α; //α为常量

β; //β为数据源最大数据获取次数阈值

Output：$S_{\text{Local}}= DB_{\text{Local1}} \cup DB_{\text{Local2}} \cup \cdots \cup DB_{\text{Local}n}$ // S_{Local} 为 I 从 S 中已获取的数据记录集

Begin:

Int $U=\{u_1,u_2,\cdots,u_n\}$; // u_k 统计 DB_k 已被获取的次数

Initialize $S_{\text{Local}}=Null$, $U=\{u_{1=1},u_{2=1},\cdots,u_{n=1}\}$;

$L=SourceSort(S)$;//$SourceSort()$为数据源排序函数

While($L <> Null$)

For every DB_k in L by order

x=0; // x 为计数器，统计对于 DB_k 连续多少个查询获取新数据的效率都不大于θ

While($x<\alpha$ & $|DB_{Localk}| \leqslant |DB_k| * \omega$&$Cost(DB_k) \leqslant t_k$)

$q_i=SelectQuery()$; //选择一个查询

$R(q_i)=Query(q_i,DB_k)$;//在 DB_k 上执行查询 q_i

$Dowload(R(q_i))$;

$DB_{\text{Local}k}= DB_{\text{Local}k}+ R(q_i)$;

$Cost(DB_k)=Cost(DB_k)+Cost(q_i,DB_k)$;

IF (q_i 从 DB_k 获取新数据的效率不大于θ)

$x= x +1$;

Else x=0; EndIF

Done

IF($u_k \geqslant \beta$ or $|DB_{\text{Local}k}| \geqslant |DB_k| * \omega$ or $Cost(DB_k) \geqslant t_k$)

$L = L-DB_k$;//满足结束条件，从 L 中删除 DB_k

EndIF

$u_k = u_k +1$; // DB_k 被循环获取次数加 1

EndFor

Done

S 中的数据源具有不同的特征，例如：数据源的大小、数据源的质量等；另外数据源之间的覆盖率也各不相同，一些数据源之间覆盖率较高，而另一些覆盖率较低，甚至可能包含另一些。因此不同的数据源对集成系统的贡献效用是有差异的。为了提高数据集成的效率，本文在开始数据获取前首先利用排序算法 $SourceSort()$对 S 中的数据源按它们可能对集成系统 I 贡献的效用大小进行排序，$SourceSort()$是一种数据源排序方法[12]，该方法主要依据数据源可能给集成系统贡献的效用大小进行排序，这里的效用是指数据源能为集成系统新增新数据量与新数据质量（数据的完整性、一致性和冗余性等）的函数。

完成对 S 中数据源的排序之后，算法开始进行数据获取，一次查询的数据获取流程为：首先由 $SelectQuery()$选择一个查询关键词 q_i（$SelectQuery()$将在下一节详细介绍），然后 $Query(q_i,DB_k)$在 DB_k 上执行查询 q_i，并返回结果页面记录集 $R(q_i)$，接着 $Dowload(R(q_i))$实现从结果页面下载数据记录到本地 $DB_{\text{Local}k}$；最后把该次数据获取的代价 $Cost(q_i,DB_k)$计入获取 DB_k 已耗费的总代价 $Cost(DB_k)$。数据获取的过程为不断重复该流程直到满足循环停止条件。

4.2 数据获取停止条件

对于该算法停止条件的设置非常重要，该算法的停止条件可以分为 2 类。第 1 类为暂时停止条件：对数据源 DB_k 的数据获取暂时停止，仍然保留在 L 中，等待下一次获取；第 2 类为结束条件：结束数据源 DB_k 的数据获取，并将该数据源从 L 中删除。

暂时停止条件设置：对于数据源 DB_k，如果 $SelectQuery()$连续选择的α个查询的新数据获取率都不大于θ，则说明 $SelectQuery()$在目前的知识（查询候选池和统计知识）下已经不能进行有效查询选择，继续对 DB_k 进行获取，代价将非常高，需要暂时停止对 DB_k 的数据获取，等待下一次循环时再继续获取。在下一次获取时则可利用丰富后的动态知识进行查询选择。

结束条件设置：当对数据源 DB_k 进行数据获取时，DB_k 的特征满足以下 3 种结束条件之一，即可从 L 中删除 DB_k，结束 DB_k 的数据获取。

1) 如果从 DB_k 中已获取的数据量$|DB_{\text{Local}k}|$达到

DB_k 估计大小的一定比例（例如：ω=95%），即 $|DB_{\text{Local}k}| \geqslant |DB_k|*\omega$，则结束 DB_k 的数据获取。

$|DB_{\text{Local}k}| \geqslant |DB_k| \times \omega$ 说明集成系统已获取 DB_k 的绝大部分数据，剩下的少量数据对集成系统的影响较小，并且获取这部分数据付出的代价也可能较高，所以可以结束 DB_k 的数据获取。

2) 如果集成系统 I 分配给 DB_k 的数据获取资源耗尽，即 $Cost(DB_k)=\sum_{i=1}^{n} Cost(q_i, DB_k) \geqslant t_k$，则结束 DB_k 的数据获取。

3) 如果 DB_k 被循环获取的次数 u_k 达到阈值 β，即 $u_k \geqslant \beta$，则结束 DB_k 的数据获取。

对于数据源 DB_k 经过了 β 次查询候选池扩展和统计知识丰富的循环获取后，从 DB_k 中继续获取新数据的可能性也较小，同时获取数据的代价随着循环获取的次数增加也不断增大，因此可以结束 DB_k 的数据获取。

5　基于集成系统动态知识的查询选择

根据同领域数据源之间的相关性，S 中数据源之间通常具有相似的属性值并且这些属性值也具有相似的分布特征，例如：在图书领域不同图书销售网站（数据源）所销售的图书具有一定的相似性，并且图书书名出现的频率也是相似的。本文提出利用集成系统已获取的数据构建动态知识，并设计基于集成系统动态知识的查询选择方法。与传统方法比较，该方法使爬虫获得更广泛的分类属性值，扩展了查询候选池，从而能避免信息孤岛问题，提高数据获取的覆盖率；同时动态知识使爬虫获得了更全面和时新的统计知识，利用动态知识可提高查询回报率估算的准确性，从而提高查询选择的效率。

5.1　动态知识构建

对于一个集成系统 I，$S=\{DB_1, DB_2, \cdots, DB_n\}$ 为 I 待集成的所有 deep Web 数据源的集合，在某一时刻 I 已获取的数据集合为：$S_{\text{Local}}= DB_{\text{Local}1} \cup DB_{\text{Local}2} \cup \cdots \cup DB_{\text{Local}n}$，$DB_{\text{Local}k}$ 为集成系统 I 从 DB_k 中已获取的数据，集成系统的动态知识（dynamic knowledge）DK 可定义为：从 S_{Local} 中得到的候选查询关键词以及该查询关键词在 S_{Local} 中出现的概率对的集合，即：$DK=\{i|<q_i, P(q_i, S_{\text{Local}})>\}$，$q_i$ 代表候选查询关键词，$P(q_i, S_{\text{Local}})$ 表示 q_i 出现在 S_{Local} 中概率。

$$P(q_i, S_{\text{Local}}) = \frac{num(q_i, S_{\text{Local}})}{|S_{\text{Local}}|} \tag{3}$$

随着数据获取工作的进行，S_{Local} 中的数据会动态变化，因此，DK 也需要根据 S_{Local} 保持动态更新，以便为查询选择提供更时新和全局的知识。理论上集成系统每执行一次查询（数据获取）都可能使 S_{Local} 变化，S_{Local} 的每一次变化又都可能使 DK 改变。例如：执行一次查询 q_k（数据获取）都有可能改变 DK 中已有候选查询关键词 q_i 的 $P(q_i, S_{\text{Local}})$ 和产生新的候选查询关键词，因此理论上每执行一次查询的数据获取都需要更新 DK。对于集成系统 I 和 S_{Local}，当一个候选查询 q_k 被选择在数据源 DB 上执行 $Query(q_k, DB)$，返回数据记录集合为 $R(q_k)$，更新 DK 主要有 2 个方面的工作。

1) 统计分析是否有新的候选查询产生，如果有新的候选查询 q_{new} 产生，则向 DK 添加候选查询 q_{new} 和 q_{new} 在 S_{Local} 中出现的概率对，新候选查询 q_{new} 在 S_{Local} 中出现的概率可由下式得到

$$P(q_{\text{new}}, S_{\text{Local}} \cup R(q_k)) = \frac{num(q_{\text{new}}, R(q_k))}{|S_{\text{Local}} \cup R(q_k)|} \tag{4}$$

2) 更新 DK 中所有候选查询在 S_{Local} 中出现的概率。对于 DK 中任一候选查询 q_i，q_i 在 S_{Local} 中出现的概率更新可由以下式计算

$$P(q_i, S_{\text{Local}} \cup R(q_k)) = \frac{num(q_i, S_{\text{Local}} \cup R(q_i))}{|S_{\text{Local}} \cup R(q_k)|} \tag{5}$$

但是如果每一次查询执行都动态维护 DK，那么代价将十分巨大，随着系统集成数据规模的增加，维护 DK 的代价将变得无法接受。本文发现在实际应用中对于一个集成系统 I，当 I 所集成的数据达到一定规模 M 后（即 DK 中知识达到一定规模后），一个查询甚至若干个查询的执行对 DK 的更新结果对查询选择的影响非常小。基于上述发现本文使用一种优化方式实现更新 DK。当集成系统 I 所集成的数据达到一定规模 M 之前，对每一次查询都更新 DK。当集成系统 I 所集成的数据达到一定规模 M 之后，执行 K 个查询后再更新 DK，K 可以随 M 的变化而变化，从而在不影响查询选择效率的前提下，尽可能减小更新 DK 的代价。

式(4)可重写为

$$P(q_{\text{new}}, S_{\text{Local}} \cup R(q_{[1,2,\cdots,K]})) = \frac{num(q_{\text{new}}, R(q_{[1,2,\cdots,K]}))}{|S_{\text{Local}} \cup R(q_{[1,2,\cdots,K]})|} \tag{6}$$

式(5)可重写为

$$P(q_i, S_{\text{Local}} \cup R(q_{[1,2,\cdots,K]})) = \frac{num(q_i, S_{\text{Local}} \cup R(q_{[1,2,\cdots,K]}))}{|S_{\text{Local}} \cup R(q_{[1,2,\cdots,K]})|} \quad (7)$$

其中，$R(q_{[1,2,\cdots,K]})$为$\{q_1,q_2,\cdots,q_k\}$在 DB 上执行所返回结果的集合。

5.2 基于动态知识的查询回报估算

DK 是 S 中所有数据源已获取的数据中得到的知识，因此爬虫拥有了一个更大的查询候选池和更全局、更时新的统计知识。对于正在进行数据获取的当前数据源 DB，DK 中的候选查询可分为 2 类：Q_{DB} 和 Q_{DK}。Q_{DB} 为包含在 DK 中，且已出现在 DB_{Local} 中的候选查询；Q_{DK} 为包含在 DK 中，但没有在 DB_{Local} 中出现候选查询。对于一个候选查询 q_i 是否能被选择执行，一个重要的因素是执行 q_i 能获得的查询回报，这里的查询回报指查询能够获取新数据的数量，下面将讨论如何估计 Q_{DB} 和 Q_{DK} 这 2 类查询的回报率。

1) Q_{DB} 查询回报估计：使用从集成系统的全局知识 DK 来估算 $q_i \in Q_{DB}$ 的回报率，查询回报估算公式如下

$$Reward(q_i,DB) = num(q_i,DB) - num(q_i,DB_{\text{Local}}) \quad (8)$$

$num(q_i,DB_{\text{Local}})$是 DB_{Local} 中与 q_i 匹配的数据记录数，$num(q_i,DB)$是目标数据源 DB 中与 q_i 匹配的记录数，$num(q_i,DB)$在 q_i 在 DB 上执行之前是未知的。

下面讨论如何估算 $num(q_i,DB)$，基于 S 中数据源之间通常具有相似的属性值并且这些属性值也具有相似的分布特征，在不考虑偏差的情况下，假定 q_i 出现在 DB 的概率等于它在全局数据上的概率 $P(q_i,S_{\text{Local}})$，$P(q_i,S_{\text{Local}})$能从 DK 中得到。基于这个假定，所有在 DB 上已提交的查询 $q_{[1,2,\cdots,m]}$出现在 DB 上的概率与全局数据上的概率也相同。因此使用如下式估算 $num(q_i,DB)$

$$num(q_i,DB) = \frac{|DB_{\text{Local}}| \times P(q_i, S_{\text{Local}})}{P(q_{[1,\cdots,m]}, S_{\text{Local}})} \quad (9)$$

2) Q_{DK} 查询回报估计：现在讨论如何估算 $q_i \in Q_{DK}$ 的回报率（查询 q_i 未出现在 DB_{Local} 中），同上本文假设 $P(q_i,DB)$ 等于 $P(q_i,S_{\text{Local}})$。因此 $Reward(q_i,DB)$的计算式如下：

$$Reward(q_i,DB) = P(q_i,S_{\text{Local}})\,|DB| \quad (10)$$

其中，$P(q_i,S_{\text{Local}})$为查询 q_i 在 S_{Local} 中概率，$P(q_i,S_{\text{Local}})$能从 DK 中得到，$|DB|$为数据源 DB 的估计大小（数据量）。

5.3 查询选择

Deep Web 数据集成爬虫的目的是在一定资源约束下尽可能多地获取数据。基于这个目的爬虫进行查询选择时必须考虑以下 2 个因素：第一，候选查询 q_i 在 DB 上执行的查询回报率；第二，候选查询 q_i 获取 DB 中数据需要花费的代价。例如：存在 2 个候选查询 q_k 和 q_j，如果它们获取 DB 的数据时需要花费的代价相同，但是 q_k 比 q_j 的查询回报率更高，q_k 应先于 q_j 被选择，同理，如果 q_k 和 q_j 具有相同的查询回报率，但 q_k 比 q_j 花费更少的代价，q_k 已应先于 q_j 被选择。因此候选查询 q_i 的效率可由下式计算。

$$Efficient(q_i) = \frac{Reward(q_i,DB)}{Cost(q_i,DB)} \quad (11)$$

$Reward(q_i,DB)$查询 q_i 在 DB 的查询回报率，$Cost(q_i,DB)$表示查询 q_i 获取 DB 中数据需要花费的代价。

因此，$Efficient(q_i)$估算单位代价情况下 q_i 能返回多少新记录。根据这个函数爬虫能够估计每一个候选查询 q_i 的效率，从而选择一个最高效率的查询首先执行。查询选择算法采用贪婪方法每次都选择具有最高潜在效率的查询，具体算法如下：

SelectQuery()
  Input: *DK*, *DB*;
  For every q_i in *DK* by order
    Efficient(q_i) ;
  Done
  Reture q_i with maximum *Efficient*(q_i)。

6 实验

6.1 数据准备

为了验证本文方法的有效性，本文使用 2 类测试数据：一类是人工构建的模拟 deep Web 数据源，另一类是真实的 deep Web 数据源。人工构建专利领域的模拟 deep Web 数据源，使用项目组已从 7 个国家 2 个组织获取的 63.32 万条专利数据，构建 5 个专利领域的 deep Web 数据源。为了模拟真实世界的同领域数据源之间存在一定的相关性（相互覆盖），又避免大量覆盖导致的实验代价过大的问题，设置每个数据源随机从 63.32 万条数据中抽取 15 万~25 万条不等的数据记录。同时，本文也在真实 deep Web 数据源上验证本文方法的有效性，从 BrightPlanet 公司的 CompletePlanet 网站的音乐领域

中选取 5 个 deep Web 数据源（music.aol.com，www.chicagoreader.com，www.onlinemusicdatabase.com，www. sheetmusicplus.com，www.bestwebbuys.com）。对于这些数据源大小通过 Arjun Dasgupta 等人[13]提出的数据源大小估计方法进行估算。

为了更好验证基于动态知识的查询选择方法 QuerySelect-DK 的效率，本文对 QuerySelect-DK 进行适当简化得到 QuerySelect-T，两者查询选择策略一样，两者的区别为：QuerySelect-DK 可利用集成系统已获取所有数据的动态知识进行查询选择，而 QuerySelect-T 仅能利用当前数据源已获取的有限知识进行查询选择。

本文主要从以下几个方面验证本文提出的方法的性能：①基于循环策略与动态知识的数据获取方法（Cycle-A）与基于 QuerySelect-T、图遍历 Graph-Based[7]和高词频 High-frequency[6]查询选择方法的独立数据获取方法性能比较；这里的独立数据获取方法是指利用上述查询选择方法独立穷尽获取每一个待集成的 deep Web 数据源，然后合并从各个数据源获取的数据，完成所有待集成数据源的数据集成。独立数据获取策略不考虑待集成的 deep Web 数据源之间的关联关系；②基于动态知识的查询选择方法（QuerySelect-DK）与基于 QuerySelect-T、图遍历 Graph-Based[7]和高词频 High-frequency[6]查询选择方法性能比较；③更新 *DK* 的查询间隔次数 *K* 对查询选择效率的影响。

6.2 实验结果

6.2.1 基于循环策略与动态知识的数据获取方法与独立数据获取方法的性能比较

为了比较本文提出的基于循环策略与动态知识的数据获取方法（Cycle-A）和采用基于 QuerySelect-T、图遍历 Graph-Based 和高词频 High-Frequency 查询选择方法的独立数据获取方法（Separate-A）的性能，集成系统分别使用 2 种方法集成专利和音乐领域的所有数据源，根据 deep Web 数据获取代价模型，2 种方法获取相同覆盖率的查询提交次数越少，deep Web 数据获取的代价也越小。因此实验时不考虑数据获取的代价因素，实验通过比较 2 种方法数据获取时的数据源覆盖率与查询提交次数的关系，测试基于循环策略与动态知识的数据获取方法的效率。实验中设置暂时停止条件为α=5，结束条件ω=95%，β=4，数据获取时代价条件不受限制。

图 1 给出了 2 种数据获取方法的效率对比，从图 1 可见，在专利领域 Cycle-A 的数据获取效率明显优于采用 3 种查询选择方法的 Separate-A（Separate-A-QST、Separate-A-GB 和 Separate-A-HF）。随着查询提交次数的增加，Cycle-A 的效率优势越来越明显，当覆盖率达到 80%时，Separate-A-QST 策略的查询提交次数几乎为 Cycle-A 策略的 5 倍，当覆盖率达到 95%时，Cycle-A 策略的查询提交次数约为 4 900 次而 Separate-A-QST 策略达到了 13 000 次左右，从音乐领域已得到类似的结果。这是因为 Cycle-A 能在较大程度上避免 Separate-A-QST 在每个数据源数据获取后期效率迅速降低的情况，同时也得益于基于集成系统动态知识的查询选择方法。同样 Cycle-A 性能与 Separate-A-GB 和 Separate-A-HF 相比已具有较大的优势。

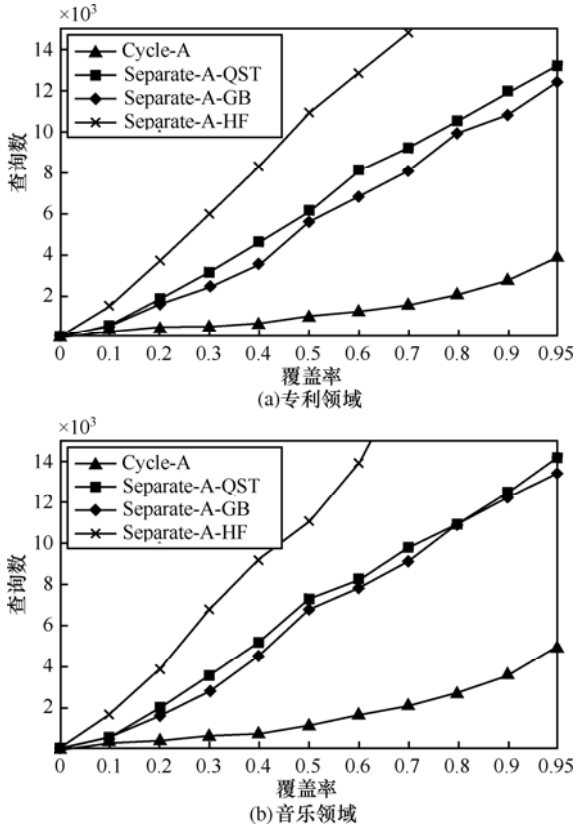


图 1　基于循环策略与动态知识的获取方法与独立获取方法的性能比较

6.2.2 QuerySelect-DK 与现有主要查询选择方法的性能比较

集成系统已获取了某领域 *n*−1 个数据源的数据，通过比较分别使用基于动态知识的查询选择方法 QuerySelect-DK 与基于 QuerySelect-T、图遍历 Graph-Based 和高词频 High-frequency 的查询选择方法获取第 *n* 个数据源的数据的效率，测试本文提出的

基于动态知识的查询选择方法 QuerySelect-DK 的效率。

实验具体设置为：对于专利和音乐领域，系统已分别集成了各领域 3 个数据源，现在分别使用上述 4 种查询选择方法获取第 4 个数据源的数据，对于 QuerySelect-DK 方法，动态知识 *DK* 根据集成系统已获取的前 3 个数据源的数据构建。图 2 给出了 4 种查询选择方法获取专利和音乐领域第 4 个数据源的效率。从图 2 可见，在音乐领域，QuerySelect-DK 方法提交大约 1 200 次查询已获取数据源大约 95%的数据，而 QuerySelect-T 方法相同查询提交次数仅仅获得大约 70%的数据。QuerySelect-DK 方法提交大约 400 次查询时可获得大约 80%的覆盖率，从获得相同覆盖率需要提交查询次数方面比较，QuerySelect-DK 方法的效率也明显优于 QuerySelect-T 方法。QuerySelect-DB 方法的效率略优于 QuerySelect-T，但与 QuerySelect-DK 的性能仍有较大差距，而 QuerySelect-HF 的效率最差。专利领域的实验结果与音乐领域类似。因此实验说明在不考虑代价的前提下，QuerySelect-DK 方法的效率高于具有相同查询选择策略的 QuerySelect-T。同时基于 QuerySelect-DK 方法的效率也明显高于现有主流的图遍历 Graph-Based 和高词频 High- frequency 的查询选择方法。

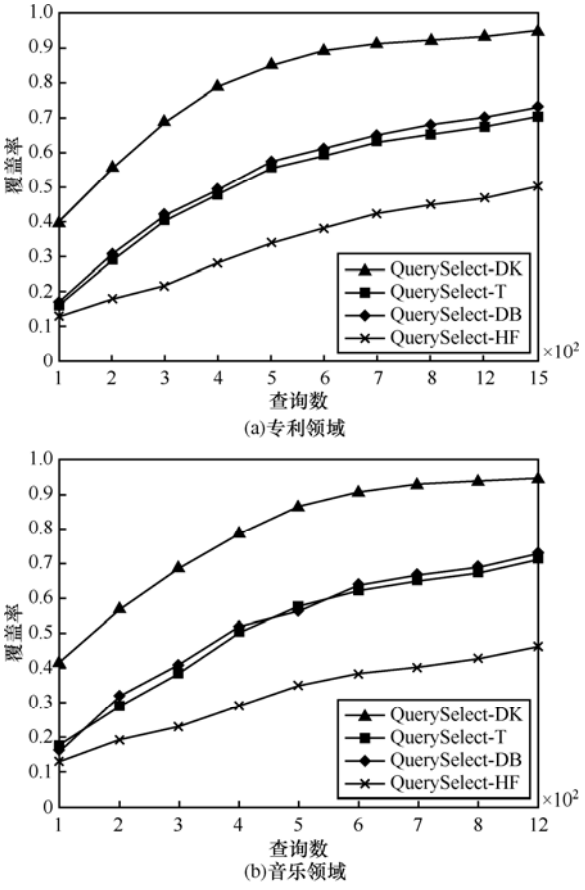


图 2 QuerySelect-DK 与现有主要查询选择方法性能比较

6.2.3 更新 *DK* 的查询间隔次数 *K* 对查询选择效率的影响

为了评估更新 *DK* 的查询间隔次数 *K* 对查询选择效率的影响，实验在 *DK* 为空和 *DK* 较丰富的这 2 种情况进行，比较不同情况下更新 *DK* 的查询间隔次数 *K* 对查询选择效率的影响。第 1 种情况，初始 *DK* 为空，从专利领域的第一数据源开始数据获取；第 2 种情况，系统已获取专利领域的前 2 个数据源的数据并构建了 *DK*，开始获取第 2 个数据源的数据。实验分别比较 *K* 为 1，10，30 时查询选择的效率，实验结果如图 3 和图 4 所示。

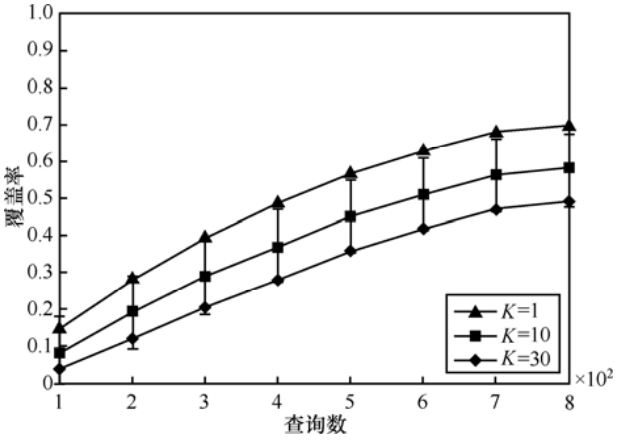


图 3 当 DK 较小时 *K* 对查询选择效率的影响

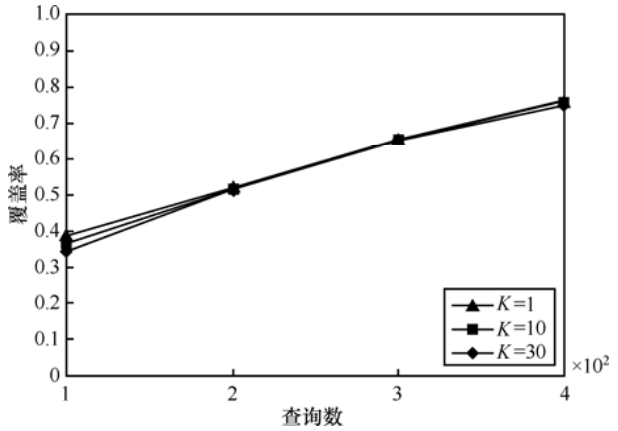


图 4 当 DK 较丰富时 *K* 对查询选择效率的影响

从图 3 中可见，在初始阶段 K=1 的效率明显高于其他 3 种情况，并且数据获取效率按照 K 值递增而递减。在大约 400 次查询提交之前，随着查询提交次数的增加 3 种情况的数据获取效率差距逐步扩大，但扩大幅度越来越小；在大约 400 次查询后 3 种 K 值数据获取效率差距已基本稳定，不再扩大，说明在此之后这 3 种 K 值的查询选择效率已基本相当。该实验说明在 DK 较小时，为了高效的数据获取需要使用较小的 K 值，虽然这样更新频率较高，但是由于此时 DK 较小，更新 DK 的代价也能接受。当随着 DK 的不断丰富，K 的大小对查询选择效率的影响逐步减小。从图 4 中可见，第 2 种情况下 3 种 K 的数据获取效率基本相同，说明在此时这 3 种不同 K 对查询选择效率的影响基本相同。因此，通过本实验可得出随着数据获取的进行，可以逐步调整 K 值，使集成系统在不太影响数据获取效率的前提下，尽量降低更新 DK 的代价。

7　结束语

针对集成环境下的 deep Web 数据获取问题，本文提出了一种新的数据获取方法。该方法根据同领域数据源之间关联性，使用循环获取策略减少了数据源后期数据的获取，较大程度上避免了传统方法数据获取后期代价巨大的问题；利用集成系统的动态知识进行查询选择，扩展了查询候选池、丰富了查询选择的知识，从而提高了数据获取的覆盖率和查询选择的准确性。实验表明，提出的基于循环策略和动态知识的 deep Web 数据获取方法能够很好地满足 deep Web 数据集成的需要，与传统方法比较，具有较高的数据获取效率。

参考文献：

[1] MICHAEL K B. The deep Web: surfacing hidden value[J]. Journal of Electronic Publishing, 2001 7(1):1174-1175.

[2] HE B, PATEL M, ZHANG Z, CHANG K C C. Accessing the deep Web: a survey[J]. Communications of the ACM, 2007,50(5):94-101.

[3] 刘伟, 孟小峰, 孟卫一. deep Web 数据集成研究综述[J]. 计算机学报, 2007,30(9):1475-1489.
LIU W MENG X F, MENG W Y. A survey of deep Web data integration[J], Chinese Journal of Computers, 2007,30(9):1475-1489.

[4] NAN Z, GAUTAM D. Exploration of deep Web repositories[A]. Proceedings of the 37th International Conference on Very Large Data Bases[C]. Tutorial, Westin, Seattle, WA, 2011.

[5] JAYANT M, SHAWN J, SHIRLEY C, *et al*. Web-scale data integration: you can afford to pay as you go[A]. Proceedings of 3rd International Conference Innovative Data Systems Research[C]. Asilomar, CA, 2007. 342-350.

[6] LUCIANO B, JULIANA F. Siphoning hidden-Web data through keyword-based interfaces[A]. Proceedings of the 19th Brazilian Symposium on Database[C]. Brasilia, Brazil, 2004.309-321.

[7] WU P, WEN J R, LIU H, *et al*. Query selection techniques for efficient crawling of structured Web sources[A]. Proceedings of the 22th International Conference on Data Engineering[C]. Atlanta,GA,USA, 2006.47-56.

[8] NTOULAS A, ZERFOS P, CHO J H. Downloading textual hidden web content by keyword queries[A]. Proceedings of the Joint Conference on Digital Libraries[C]. Denver, Colorado, USA, 2005. 100-109.

[9] JAYANT M, DAVID K, LUCJA K, *et al*. Google's deep-Web crawl[A]. Proceedings of 34th International Conference on Very Large Data Bases[C]. Auckland, New Zealand, Springer,2008.1241-1252.

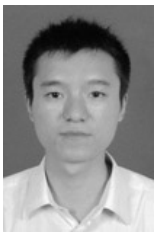
[10] JIANG L, WU Z H, FENG Q, *et al*. Efficient deep Web crawling Using reinforcement learning[A]. Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining[C]. Hyderabad, India, Springer, 2010. 428-439.

[11] GEORGE V, ALEXANDROS N, DIMITRIOS G. Rank-aware crawling of hidden Web sites[A]. Proceedings of the International Workshop on the Web and Databases[C]. Athens, Greece, 2011.

[12] XIAN X F, ZHAO P P, YANG Y F, *et al*. Efficient selection and integration of hidden Web database[J]. Journal of Computers, 2010, 5(3): 500-507.

[13] ARJUN D, Xin J, BRADLEY J, *et al*. Unbiased estimation of size and other aggregates over hidden Web databases[A]. Proceedings of the ACM SIGMOD International Conference on Management of Data, Indianapolis[C]. Indiana, USA, ACM Press, 2010. 855-866.

作者简介：



鲜学丰（1980-），男，四川南充人，苏州大学博士生，主要研究方向为 Web 数据管理、数据挖掘和智能信息处理。

崔志明（1961-），男，上海人，苏州大学教授、博士生导师，主要研究方向为智能信息处理和计算机网络。

赵朋朋（1980-），男，江苏南通人，博士，苏州大学副教授，主要研究方向为 deep Web 和 Web 数据挖掘。

梁颖红（1970-），女，黑龙江哈尔滨人，博士，苏州市职业大学教授，主要研究方向为智能信息处理和自然语言处理。

方立刚（1980-），男，安徽黄山人，博士，苏州市职业大学副教授，主要研究方向为计算机网络和 Web GIS。