概念格的分布式集成算法研究

范淑媛 王黎明 姜 琴 张 卓
(郑州大学信息工程学院 郑州 450001)

摘 要 随着大数据时代的到来,海量数据的分布存储和分布计算变得越来越重要,其中概念格的分布式集成变得尤为紧迫。为了解决概念格的构格时间较长的问题,提出了面向概念格的分布式集成算法。概念格的集成是先对子概念格中的概念按内涵个数递减进行排序,再将排序后的子概念格集成为全局概念格。构造全局概念格选择两种集成方式:1)添加式集成方式,即主节点接收并集成来自所有子节点的子概念格;2)二路归并式集成方式,即各个子节点处的所有子概念格先集成,而后将所得的概念格提交给主节点接收并完成最终集成。实验表明,这两种概念格的分布式集成策略各有优缺点,但都能够减少概念格的构格时间。

关键词 子概念格,全局概念格,分布式,添加式集成,二路归并式集成

中图法分类号 TP181 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.6.045

Research of Distributed Integration Algorithm on Concept Lattices

FAN Shu-yuan WANG Li-ming JIANG Qin ZHANG Zhuo
(School of Information Engineering, Zhengzhou University, Zhengzhou 450001, China)

Abstract With the advent of the era of big data, the distributed storage and computing of massive data are increasingly important, and the distributed integration of concept lattices is particularly urgent. In order to solve the problem of long constructing time of concept lattice. This paper put forward the concept lattices oriented distributed integration algorithm. Integration of concept lattice are defined as follows: concepts in sub-concept lattice are sorted according to the decreases of intent number, and then the sub-concept lattices are integrated into a global concept lattice. Two types of integration are selected to construct the global concept Lattice in this paper: one is the add lattice merge, the master node receives and integrates sub-concept lattices that come from all child nodes; the other way is called two-way merge, firstly, its sub-concept lattices from all child nodes are integrated, and then the master node receives and integrates the sub-concept lattices. The experiments show that two kinds of distributed integration strategy of concept lattice have their advantages and disadvantages, but both of them can effectively reduce the time of constructing concept lattice.

Keywords Sub-concept lattice, Global concept lattice, Distributed, Add lattice merge, Two way merge

1 引言

形式概念分析(Formal Concept Analysis,FCA)理论[1]由德国的 R. Wille 教授于 1982 年提出,其核心数据结构概念格已经广泛地应用于数据挖掘、知识发现、信息检索、数据抽取等诸多领域[2-4]。概念格表示各个概念与概念之间的关系,也是描述客观世界中的一种简化形式。

对于构造概念格已被广泛研究,从最早的枚举型 Next-Closure 算法[5],到后来的多层次构造算法,如 Bordat[6]、Linding[7]算法;再到渐增式的算法,其中以 Godin[8]算法为代表;直到 21 世纪初 ValtchevP[9,10]提出了集成构造算法。后来国内刘宗田团队根据概念格的特性,对概念格的合并原理[11]分析,得出合并是应该遵循的相应关系,文献[12]基于频繁概念直乘分布的全局闭频繁项集挖掘算法,实现对于频繁的形式背景下的经典概念格的并置合并方法。文献[13,14]实现并行环境下,由多个不同的处理器产生不同的节点,其中,如何分配处理器来产生概念是关键。文献[15]针对随机决策形式背景来构造概念格,提出的随机概念格构造算法能够有效地对自然界中大量的随机现象所产生的随机概念进行处理。文献[16]从约简概念格的属性和对象两个方面考虑来减少够格的时间复杂度。文献[17]从负载均衡的并行方面考虑来减少构造概念时间。文献[18]采用面向形式背景进行概念格的合并构造,即是将整个形式背景进行纵向划分为多个子形式背景,然后以多个子形式背景为基础构造子概念格,最后对各子概念格进行合并,形成最终的概念格,而相对于传统的概念格构造算法相比,构格时间得到了明显的提高。

目前所研究的构造算法都是面向形式背景的,概念格的构造过程具有指数级时间复杂度,其极低的构造效率是目前

到稿日期:2015-03-09 返修日期:2015-06-06 本文受国家自然科学基金项目(61303044)资助。

范淑媛(1987-),女,硕士生,主要研究方向为机器学习、形式概念分析及应用、数据挖掘,E-mail:happy_syfan@163.com;王黎明(1963-),男,博士,教授,CCF 高级会员,主要研究方向为现代软件工程、人工智能和数据挖掘等;姜 琴(1989-),女,硕士生,主要研究方向为机器学习、数据挖掘;张 卓(1978-),男,博士,讲师,CCF 会员,主要研究方向为形式概念分析及应用等。

在研究和应用过程中面临的主要问题之一，针对以上问题，本文提出了面向概念格的分布式集成方法。由于目前的数据基本都是存储在数据库中，当读出数据时就会耗费大量的时间，为了减少存储数据的时间，本文提出一种将本领域内所涉及的数据（子概念格）放在文件夹中，而非是在数据库中，这样就可以在读取子概念格以及再对这些子概念格集成时，省去很多时间。本文对于面向概念格的集成，提出了两种概念格的分布式集成算法：添加式集成算法和二路归并式集成算法，算法的核心是对两个子概念格自底向上进行集成，在集成方式中选择不同的策略，最终得到全局概念格。其中待集成的子概念格可以采用不同的构造算法来得到，这两种方式对于概念格的分布式集成效率都有所提高，在大数据时代具有更深远的意义。

2 相关概念

在文献[1-13]中，分别定义了形式背景、概念格、横向合并、纵向合并等相关概念，在格定义的基础上，本文做出如下定义。

定义 1（同构分布式子概念格） 对于一个拥有 n 个节点分布式环境，每个节点都拥有各自的概念格[1] $L_i(K_i)$，并且任意两个节点所对应的形式背景 K_i 和 K_j 满足 $O_i \cap O_j = \varnothing$，$A_i \cap A_j \neq \varnothing$，那么所有节点的概念格的集合 $L_D = \{L_1, L_2, L_3, \cdots, L_n\}$ 称为同构分布式全局概念格。其中每个节点的概念格 L_i 称为同构分布式子概念格。

定义 2（异构分布式子概念格） 对于一个拥有 n 个节点分布式环境，每个节点都拥有各自的概念格[1] $L_i(K_i)$，并且任意两个节点的所对应的形式背景 K_i 和 K_j 满足 $A_i \cap A_j = \varnothing$，$O_i \cap O_j \neq \varnothing$，那么所有节点的概念格的集合 $L_D = \{L_1, L_2, L_3, \cdots, L_n\}$ 称为异构分布式全局概念格。其中每个节点的概念格 L_i 称为异构分布式子概念格。

定义 3（全局概念格） 同构（异构）分布式环境下的子概念格集成所得到所有的概念及其上的偏序关系[1]，是一个完备格，对应的概念格称为全局概念格，记作 L_G。

定义 4（子概念格叠置） 由同构分布式子概念格 L_i 和 L_j 进行集成，通过比较概念格中的外延相应的关系得到全局概念格的过程称为子概念格 L_i 和 L_j 的叠置，叠置集成后得到的全局概念格表示为 $L_{i,j}$ 或 L_{k_i/k_j}。

定义 5（子概念格并置） 由异构分布式子概念格 L_i 和 L_j 进行集成，通过比较概念格中的内涵相应的关系得到全局概念格的过程称为子概念格 L_i 和 L_j 的并置，并置集成后得到的全局概念格表示为 $L_{i,j}$ 或 L_{k_i/k_j}。

定义 6（合成概念的下界概念的集合） 由全局格中的概念 $C(X,Y)$ 或表示为 (c_i, c_j) 是由子概念格中的两个概念节点 c_i, c_j 合并得到，其中 $c_i \in L_1, c_j \in L_2$，那么概念 (c_i, c_j) 的下界概念为 (c_i, c_j') 和 (c_i', c_j) 的总和。其中 c_i' 为概念 c_i 在子概念格 L_1 中的直接下界，c_j' 为概念 c_j 在子概念格 L_2 中的直接下界。

定义 7（概念上/下邻节点的集合） 由概念 $C(X,Y)$ 的所有父概念/子概念组成的集合称为概念 C 的上/下邻节点的集合，表示为 $Cov^u(C)$，$Cov^l(C)$。相应的概念的外延和内涵的上下邻节点的集合也是类似的表示。

3 相关理论

对于异构环境下的两个子概念格分别记作 L_1 和 L_2，其中概念 $c_1 = (X_1, Y_1) \in L_1$，$c_2 = (X_2, Y_2) \in L_2$。由于概念格具有完备性[8]，即是具有相应的排序关系，在本文对于子概念格中的概念首先按照内涵递减的顺序进行排序，即先用一个一位数组存放子概念格中的所有概念，然后分别比较概念的内涵个数的大小，按内涵个数递减的顺序排列后的概念存放在另一个一维数组中，待后续运用。

文献[10]在两个子概念格进行合并时，要求两个子概念格所对应的形式背景中的外延和内涵具有下面的关系：$A_i \cap A_j = \varnothing$，$O_i = O_j$，而本文中的子概念格在进行集成时并不要求其对象集必须是相等的，只要是相交不为空即可，下面对于这一条件，证明对应 $O_i = O_j$ 的理论同时适用于本文。

定理 1 对于任意两个异构分布式子概念格中的两个子概念 $c_1 = (X_1, Y_1) \in L_1$，$c_2 = (X_2, Y_2) \in L_2$ 和它们所对应的全局模糊概念格 L 中的概念 $c = (X, Y)$ 之间的映射关系 φ，则 $X = ((X \cap O_1), (X \cap O_1)')$，$Y = ((X \cap O_2), (X \cap O_2)')$ 是正确的。

证明：由定义 2 知，异构环境下的子概念格之间的概念存在 $A_1 \cap A_2 = \varnothing$，$O_1 \cap O_2 \neq \varnothing$ 关系，一共可以分为 3 种情况：1）$O_1 = O_2$，这种情况下与文献[10]中的情况是一样的，故有对于 $A_1 \cap A_2 = \varnothing$，$O_1 = O_2$ 成立，在这里本文不再做详细证明；2）$O_1 \subset O_2$，在这种情况下，在合并过程中 $O_1 \cap O_2 = O_1$，对于合并后的格中的对象域和属性域分别是 $O = (O_1 \cap O_2)$，$A = (A_1 \cup A_2)$，根据概念格的偏序关系可知合并后的概念格中的对象域会相应地减小，而属性域会相应地扩大，假设有概念 $c_1 = (X_1, Y_1) \in L_1$，$c_2 = (X_2, Y_2) \in L_2$ 合并后得到的概念 $c_p = (X_p, Y_p) \in L$，则有 $X_p = X_1 \cap X_2$，$Y_p = X_p'$，又因为 $O_1 \subset O_2$，所以 $X_1 \cap X_2 = X_1 = X_p$，根据概念格的定义可知 $Y_1 = X_1'$（属性域为 A_1），$Y_p = X_p'$（属性域为 A），而根据从子概念格到全局概念格的偏序关系[10] 知 $\varphi(X, Y) = (((Y \cap A_1)', (Y \cap A_1)), ((Y \cap A_2)', (Y \cap A_2)))$，合并后，若是全局概念格中的概念，那么必须存在，$X = ((X \cap O_1), (X \cap O_1)')$，$Y = ((X \cap O_2), (X \cap O_2)')$。故定理 1 在 $O_1 \subset O_2$ 情况下是成立的；3）$O_1 \Leftrightarrow O_2$，证明与 2）中情况类似，可知定理 1 也是成立的。

定理 2 对于全局概念格 L 中的概念 $c = (X, Y)$ 和子概念格中，分别有 $c_1 = (X_1, Y_1) \in L_1$，$c_2 = (X_2, Y_2) \in L_2$，两个概念之间存在着 $\varphi(c) = (c_1, c_2)$，那么全局概念与子概念之间的对应关系为：$X = X_1 \cap X_2$，$Y = Y_1 \cup Y_2$。

证明：假设有概念 $c_1 = (X_1, Y_1) \in L_1$，$c_2 = (X_2, Y_2) \in L_2$ 合并后得到的概念 $c_p = (X_p, Y_p) \in L$，则有 $X_p = X_1 \cap X_2$，$Y_p = X_p'$，由命题 1 知，$X_1 = X \cap O_1$，$Y_1 = (X \cap O_1)'$，$X_2 = X \cap O_2$，$Y_2 = (X \cap O_2)'$，在异构环境下知，$Y_1 = X_1'$（在格 L_1 中）$= X_p'$（在格 L 中），$Y_2 = X_2'$（在格 L_2 中）$= X_p'$（在格 L 中），那么有 $Y_p = X_p' = X_1'$（在格 L_1 中）$\cup X_2'$（在格 L_2 中），又知 $\varphi(c) = (c_1, c_2)$，可知 $Y_1 \cup Y_2$ 是概念 C_p 的内涵，即 $Y_p = Y_1 \cup Y_2$，即定理 2 得证。

定理 3 异构分布式子概念格并置集成后的全局概念格 L_D 与所有子概念格相应的形式背景的集合所得到的概念格 L_G 是同构的，即 $L_D \cong L_G$。

证明：由定理 1 和定理 2 知，在异构分布式子概念格并置后得到的全局概念格 L_D 的构造是有子概念格由函数 φ, ψ[10]

经过相应的映射得到，而从文献[10]中知，函数 φ,ψ 都具有偏序关系，而概念格即具有偏序关系，又具有完备性，故在从子概念格集成得到全局概念格的过程中，不存在着元素(概念)的丢失，易知，两种方法得到的概念格是同构的，即是 $L_D \cong L_G$。

由上面的理论可知，文献[10]中的理论和算法同时适用于本文异构环境下概念格的分布式集成，合并算法在文献[10]中有了详细的介绍。

4 异构环境下概念格的分布式集成

为提高异构环境下概念格的分布式集成的效率，降低构造概念格的时空复杂度，面对多个子概念格(可以用任意算法构造的)，本文将子概念格存储在文件中，而不是将其存储在数据库中。由于存储在数据库中，每次都需要从数据库中找到相应的数据，然后才能够运用这些数据，这将会耗费大量的时间，而本文中所涉及到的数据可以直接存储在文件中，这样不管是构造子概念格，还是由子概念格来集成构造全局概念格，都比直接从数据库中读取数据节省不少的时间。

关于异构环境下概念格的分布式集成，本文在第 1 部分相关概念和第 2 部分相关理论的基础上，只设置一个主节点，其他为子节点(多个)，主节点集成得到最终的全局概念格，而在子节点处，主要分为两种方式进行集成，一种是添加式集成，另一种是二路归并式集成。基于同构环境下子概念格的分布式集成与基于异构环境下子概念格的分布式集成方法类似，集成的思想相同，在这里将不对同构环境下的叠置做过多的介绍。下面详细说明异构环境下子概念格分布式集成的相关理论及过程和相关的算法。

本文两两概念格的集成是在 ValthevP 所构造概念格[9]的基础上进行的，下面为只有两个子概念格运用自底向上的顺序构造全局概念格的过程。

步骤 1：接收异构环境下子概念格；

步骤 2：对子概念格中的概念进行按照内涵递减排序；

步骤 3：依次对排好序的概念进行两两外延相交，找到共同的外延；

步骤 4：根据定理 2 找到所有的全局格中的概念；

步骤 5：对步骤 4 中找到的概念，找其直接相连的下界节点概念进行连接；

步骤 6：重复步骤 4，5，直到生成全局概念格结束。

子概念格里的概念是按照内涵递减的顺序进行排列，给异构环境下全局概念格的构造带来了很大的方便，这样在整个概念格的更新过程中，避免产生一些冗余信息，减少了不必要的遍历，由文献[10]知，生成的概念的下界节点都先于此节点，故减少了整个概念格的遍历更新时间。

下面用一个例子来说明两两概念格的集成，给出两个子概念格分别为 L_1，L_2，如图 1、图 2 所示。

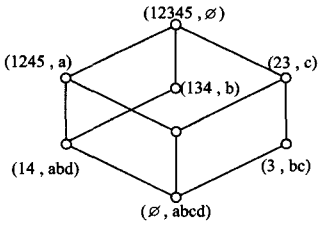

图 1　子概念格 L_1

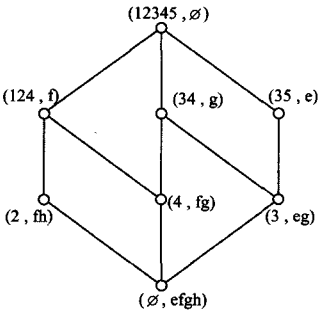

图 2　子概念格 L_2

对图 1 子概念格 L_1 中的概念按内涵递减排序后得到的概念序列是{∅,$abcd$}，{14,abd}，{2,ac}，{3,bc}，{1245,a}，{134,b}，{23,c}，{12345,∅}，对图 2 子概念格 L_2 中的概念按内涵递减排序后得到的概念序列是{∅,$efgh$}，{3,eg}，{4,fg}，{12,fh}，{35,e}，{24,f}，{34,g}，{12345,∅}。接着对 L_1 中的概念{∅,$abcd$}与 L_2 中的概念{∅,$efgh$}外延相交为∅，其下界不存在，故由这两个概念组成的概念是全局格中的概念，两个概念的内涵进行并运算得到全局格中的最小概念为{∅,$abcdefgh$}，而后对 L_1 中的概念{∅,$abcd$}与 L_2 中的概念{3,eg}外延相交为∅，其下界集合为({∅,$abcd$}，{∅,$efgh$})的外延为空，故有{∅,$abcd$}与{3,eg}组成的概念不是全局格中的概念，依次这样比较下去，直到 L_1 中的概念{14,abd}与 L_2 中的概念{4,fg}的外延相交为 4，其下界组合{({14,abd}，{∅,$efgh$})，({∅,$abcd$}，{4,fg})}中概念的外延都是∅，与({14,abd}，{4,fg})外延的个数不等，故({14,abd}，{4,fg})组成的概念是全局概念格中的概念{4,$abdfg$}，然后按照函数 *UpDate-Order*[10] 依次连接构格，在全局概念格中概念{4,$abdfg$}是概念{∅,$abcdefgh$}的直接上界，概念{∅,$abcdefgh$}是概念{4,$abdfg$}的直接下界，如此反复直到最终得到全局概念格中的所有概念。最终合并之后得到的全局概念格 L_{12} 如图 3 所示。

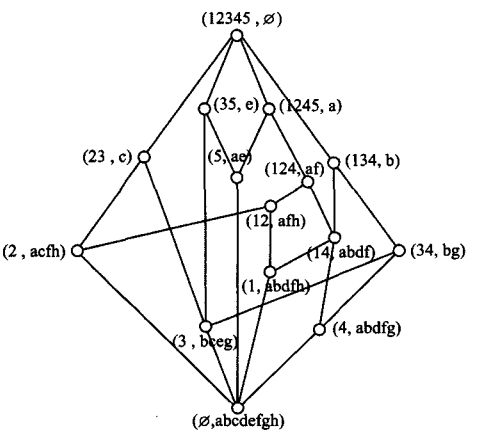

图 3　子概念格 L_{12}

1)添加式集成算法

所谓添加式集成方式即是对于所有子概念格的集成只在主节点处进行，子节点的作用是向主节点提供子概念格，供主节点进行集成。下面用一个例子说明，设异构环境下共有 1 个主节点，4 个子节点，分别为节点 1－4，8 个子概念格 L_1－L_8，在子节点 1 处的子概念格为 L_1，L_2；子节点 2 处的子概念格为 L_3，L_4；在子节点 3 处的子概念格为 L_5，L_6；子节点 4 处的子概念格为 L_7，L_8，其集成过程如图 4 所示。

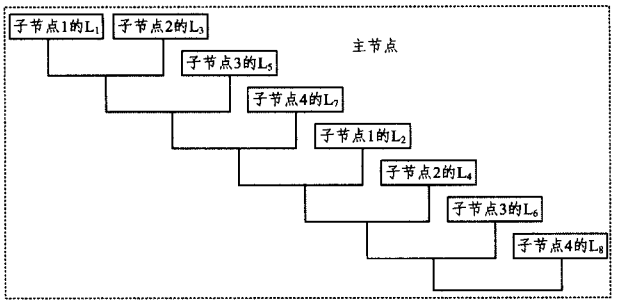

图 4　添加式集成方式

主节点先接收来自子节点 1 提供的子概念格 L_1，然后接收来自子节点 2 提供的子概念格 L_3，当主节点处有两个子概念格时，就开始运用集成算法 Assembly[10] 进行子概念格的集成为 L_{13}，然后接收来自子节点 3 提供的子概念格 L_5 与刚集成的概念格 L_{13} 进行集成，然后接收来自子节点 4 提供的子概念格 L_7 与主节点处刚集成的进行集成，如此反复，依次加入式的集成子概念格 L_2，L_4，L_6 及 L_8，得到最后的全局概念格 L。

算法思想：对于异构环境下的两个及两个以上的子概念格合并成一个全局概念格的过程，首先对来自子节点的两个子概念格进行合并，合并后的概念格再与来自子节点的子概念格进行合并，如此重复，直到最后形成包含 n 个子概念格的全局概念格为止。算法描述如下。

算法 1　ALMerge(AddLatticeMerge)($L_1 \cdots L_n$)

输入：L_1，…，L_n，n 个异构环境下的子概念格

输出：全局概念格 $L=L_1 \cup L_2 \cup \cdots \cup L_n$

1. L←Ø;
2. L←L_1;
3. for(int i=2,i<n+1,i++)
4.　L=Assembly(L,L_i);
5. end for
6. return L

算法 1 中第 3、4 行是整个算法的核心部分，即是循环调用 Assembly 算法，最终集成得到全局概念格 L。由于算法 ALMerge 最终的集成是在主节点处进行的，因此对于海量的子概念格都可以进行集成，即算法 ALMerge 可以运用到大数据集下。

2）二路归并式集成算法

所谓二路归并式集成方式即是主节点处只接收来自子节点的一个在该子节点处的所有的子概念格先进行集成后的概念格，即是在主节点处和子节点处的工作方式是一样的，都是两两进行集成，最终集成为该节点处的全局概念格。下面用一个例子说明，设异构环境下共有 1 个主节点，4 个子节点，8 个子概念格，跟上文中的分布情况相同，其集成过程如图 5 所示。

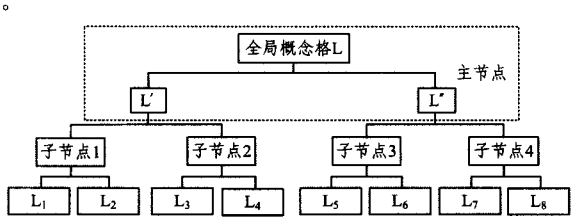

图 5　二路归并式集成方式

运用二路归并式集成子概念格时，主节点只接收来自子节点的集成概念格。主节点依次接收来自子节点 1 的集成概念格 L_{12} 和来自子节点 2 的集成概念格 L_{34} 运用 Assembly 算法进行集成为 L'，接收来自子节点 3 的集成概念格 L_{56} 和来自子节点 4 的集成概念格 L_{78} 运用 Assembly 算法进行集成为 L''，然后再将 L' 和 L'' 进行集成为最终的全局概念格。

算法思想：对于异构环境下的两个及两个以上的子概念格合并成一个全局概念格的过程，对 n 个子概念格进行两两合并，得到$\lceil n/2 \rceil$个集成的概念格，再进行两两合并，如此重复，直到最后形成包含 n 个子概念格的全局概念格为止。算法描述如下。

算法 2　TWMerge(TwoWayMerge)($L_1 \cdots L_n$)

输入：L_1，…，L_n，n 个异构环境下的子概念格以及概念格的个数的最大和最小值 low，high

输出：全局概念格 $L=L_1 \cup L_2 \cup \cdots \cup L_n$

1. L←Ø;
2. L_1←Ø;L_2←Ø;
3. a=low;b=high;
4. if n>1
5. P=(b+a)/2;
6. L_1=TWMerge($L_1 \cdots L_p$,a,P);
7. L_2=TWMerge($L_{p+1} \cdots L_n$,P+1,b);
8. L=Assembly(L_1,L_2);
9. else
10. L=L_1;
11. return L

算法 2 中第 4 行是判断待合并的概念格的个数，如果是只有一个，最终的全局概念格即为这个子概念格，若不是，就将所有的子概念格分成 2 部分（第 5 行），然后对这两部分再分别调用 TWMerge 算法，合成为概念格 L_1 和 L_2（第 6－7 行），第 8 行对于全局格来看，合成的全局概念格 L 是由合成过的 L_1 和 L_2 集成得到的。

算法 TWMerge 主节点处集成的是所有子节点的集成后的概念格，故在主节点处可以集成海量的子概念格，而每个子节点处也是进行该子节点处所有子概念格的集成，故此算法在大数据集下，在主节点处最终所集成的两个概念格是规模比较大的概念格，而算法 ALMerge 最终所集成的两个概念格是一个规模比较大的概念格，而另一个则是规模比较小的概念格，故在理论上当子概念格的数目比较多时，算法 ALMerge 比算法 TWMerge 好些，而且两种算法都可以运用到海量的子概念格中。

算法 Assembly[10] 的时间复杂度与 L_1 和 L_2 格中概念的内涵和外延都有关系，设概念格中概念的对象和属性的个数分别为 $\|O\|$，$\|A\|$，L_1 和 L_2 中的概念个数分别用 l_1 和 l_2 来表示，由文献[10]可知，算法 Assembly 的时间复杂度为 $O((\|O\|+\|A\|)(l_1 * l_2 + nm))$。算法 ALMerge 中当有 n 个子概念格要集成时，需要调用$(n-1)$次 Assembly 算法，所以算法 ALMerge 的时间复杂度为 $O((\|O\|+\|A\|)(l_1 * l_2 + nm) * n)$。算法 TWMerge 中当有 n 个子概念格要集成时，需要$\lceil log_2 n \rceil$次集成，故采用二路归并式集成方式构造全局概念格的时间复杂度为 $O((\|O\|+\|A\|)(l_1 * l_2 + nm) * n\log n)$。

5　实验设计与分析

实验环境：Win7 操作系统，单处理器八核计算环境，Java

实验平台，除操作系统外，无其它程序同时运行。实验目的：测试 ALMerge 算法及 TWMerge 算法的完备性，进一步分析 ALMerge 算法和 TWMerge 算法的特性。实验数据集 1－6 为随机产生的稀疏数据集和稠密数据集由 MATLAB 自动生成，运用 Construct 算法[10] 而得到的子概念格。具体说明如下。

数据集 1：稀疏数据集，拥有 10 个对象，属性个数从 5 个依次增加 1 个分别直到 14、24、…、104 个，平均非零项分别占 30%，得到的子概念格数从 10 个每次增加 10 个直到 100 个；

数据集 2：稠密数据集，拥有 10 个对象，属性个数从 5 个依次增加 1 个分别直到 14、24、…、104 个，平均非零项分别占 70%，得到的子概念格数从 10 个每次增加 10 个直到 100 个；

数据集 3：稀疏数据集，拥有的对象个数从 10 个每次增加 1 个分别直到 19、29、…、109 个，6 个属性，平均非零项分别占 30%，得到的子概念格数从 10 个每次增加 10 个直到 100 个；

数据集 4：稠密数据集，拥有的对象个数从 10 个每次增加 1 个分别直到 19、29、…、109 个，6 个属性，平均非零项分别占 50%，得到的子概念格数从 10 个每次增加 10 个直到 100 个；

数据集 5：稀疏数据集，拥有的对象个数从 10 个每次增加 1 个直到 29 个，属性个数从 5 个依次增加 1 个分别直到 6、8、…、24 个，平均非零项分别占 30%，得到的子概念格数从 40 个每次增加 40 个直到 400 个；

数据集 6：稀疏数据集，拥有的对象个数从 10 个每次增加 1 个直到 29 个，属性个数从 5 个依次增加 1 个分别直到 8、12、…、44 个，平均非零项分别占 30%，得到的子概念格数从 80 个每次增加 80 个直到 800 个。

1）ALMerge 算法和 TWMerge 算法的完备性验证

本文对于由形式背景（由 MATLAB 自动生成）直接生产的概念格记作 L，而由子概念格集成得到的概念格记作 $L_{1,2}$，运用 Construct 算法所建立的概念格 L 与运用算法 ALMerge 和算法 TWMerge 所建立的概念格 $L_{1,2}$ 中的概念节点数目的比较，来验证算法 ALMerge 和算法 TWMerge 的完备性。

表 1 中的 3 个数据集分别是取上文中所描述的数据集中的某两个子形式背景，由表 1 可以看出：在相同的形式背景下，运用算法 Construct[10] 所建立的概念格 L 与运用算法 ALMerge 和算法 TWMerge 所建立的概念格 $L_{1,2}$ 中的概念数目一样，由概念格的偏序关系可知，当两个概念格中概念个数相同，两个概念格是相同的（在相同的形式背景下，两个概念格中概念个数相等，而概念格不等是小概率事件，可忽略），从而验证了算法 ALMerge 和算法 TWMerge 的完备性。

表 1　不同数据集下，运用不同的算法所产生的概念格中概念的数量

	数据集 3		数据集 5		数据集 6	
L_1	38	38	329	329	3732	3732
L_2	57	57	318	318	3308	3308
L	83	83	1294	1294	8588	8588
$L_{1,2}$	ALMerge 83	TWMerge 83	ALMerge 1294	TWMerge 1294	ALMerge 8588	TWMerge 8588

2）ALMerge 算法和 TWMerge 算法在数据集下的实验结果

对于数据集 1、数据集 2、数据集 3 和数据集 4 得到的子概念格集成得到的全局概念格中概念数目如表 2 所列。在表 2 中 L_n 表示的是子概念格的数目，$L_{concept}$ 分别表示不同的数据集下集成后的全局概念格中概念数目。

表 2　数据集 1、2、3、4 运用不同的算法所产生的概念格中概念的数量

L_n	10	20	30	40	50	60	70	80	90	100
数据集 1$L_{concept}$	53	86	127	184	189	253	256	275	303	325
数据集 2$L_{concept}$	125	240	359	431	516	579	650	690	698	743
数据集 3$L_{concept}$	28	65	65	80	92	108	117	125	156	237
数据集 4$L_{concept}$	61	91	129	163	201	267	367	482	609	823

对于数据集 1 和数据集 2 其形式背景只有非零项所占比例不同，对于数据集 3 和数据集 4 其形式背景也是只有非零项所占比例不同，从表 2 可以看出当非零项所占比例越大，所得到的全局概念格中的概念数就会相应地增加；而且随着子概念格数目的增多，集成后所得到的概念格中概念的数目也相应地增加。数据集 1、2 集成所消耗的时间对比如图 6、图 7 所示；数据集 3、4 集成所消耗的时间对比如图 8、图 9 所示。

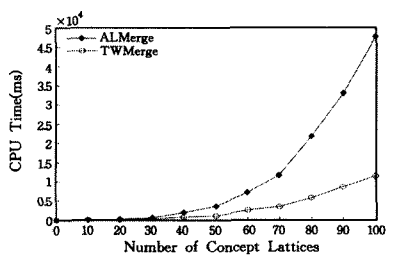

图 6　ALMerge 与 TWMerge 对于数据集 1 的子概念格进行集成耗时的对比

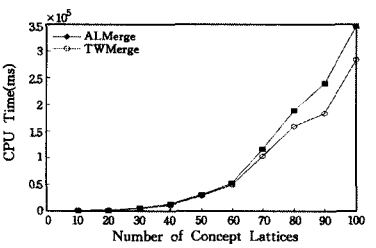

图 7　ALMerge 与 TWMerge 对于数据集 2 的子概念格进行集成耗时的对比

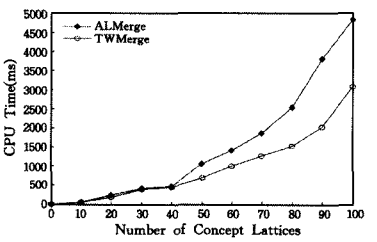

图 8　ALMerge 与 TWMerge 对于数据集 3 的子概念格进行集成耗时的对比

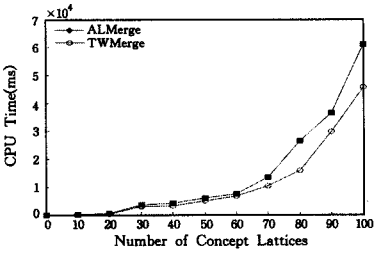

图 9 ALMerge 与 TWMerge 对于数据集 4 的子概念格进行集成耗时的对比

从图 6—图 9 可以看到，这两种算法耗时随着子概念格数目增加都呈现递增的趋势，在图 6 中当子概念格数目小于 30 时，两种算法所消耗的时间基本一致，当子概念格的数目大于 30 时，运用算法 ALMerge 集成所耗的时间明显比运用算法 TWMerge 所耗的时间要多，而且增加的比例也比 TWMerge 快得多。从图 7 中可以看到，在子概念格数目小于 60 时运用两种算法所消耗的时间差不多，在子概念格数目大于 60 时，算法 TWMerge 稍微比算法 ALMerge 耗时较少，但与图 6 比较可以看到，当在稀疏数据集下，运用算法 TWMerge 明显比算法 ALMerge 好，而在稠密数据集下，两种算法相差不多。比较图 6 与图 7 纵坐标可知，当子概念格中概念的个数增多时，两种算法耗时也会相应地增加。

从图 8、图 9 中可以看到当形式背景中的属性数目固定，随着对象个数的增加，两种算法在概念格的集成过程中时间的消耗也呈现增加的趋势，在图 8 中当子概念格的数目小于 40 时，两种算法消耗的时间差别不大，而当在子概念格的数目大于 40 时，算法 ALMerge 比 TWMerge 耗时多，与图 6 相比可知，两种算法耗时不及图 8 中多，那么可知对象个数不变，属性个数的增加对于算法耗时的影响比相同非零数据项相同的情况下，属性个数不变，对象个数增加的影响要大。

在图 9 中算法 ALMerge 耗时比 TWMerge 耗时要多，与图 8 对比，非零数据项所占比例增加，可知耗时的数量级也会相应增加。但是当子概念格中概念的个数增多时，在图 9 中两种算法的差别趋势没有图 8 中两种算法的差别大，可知当非零数据项小的情况下，算法 TWMerge 更优于算法 ALMerge。图 9 与图 7 对比可知，图 9 中两种算法的耗时比图 7 耗时要多，也可以得出对象个数不变，属性个数的增加对于算法耗时的影响比相同非零数据项相同的情况下，属性个数不变，对象个数增加的影响要大。

图 6—图 9 都是在子概念格从 10 增加到 100 进行概念格的集成，此数据集在海量数据的背景下，可以看作是小数据集，通过上面的分析可知，在子概念格小于 100 的情况下应该选择算法 TWMerge 进行概念格的集成。

对于数据集 5 和数据集 6 得到的子概念格集成得到的全局概念格中概念数目分别如表 3、表 4 所列。在表 3、表 4 中 L_n 表示的是子概念格的数目，$L_{concept}$ 分别表示不同的数据集下集成后的全局概念格中概念数目。两种算法对于数据集 5 和数据集 6 构格所消耗的时间对比分别如图 10、图 11 所示。

表 3 数据集 5 运用不同的算法所产生的概念格中概念的数量及构格时间

L_n	40	80	120	160	200	240	280	320	360	400
数据集 5 $L_{concept}$	76	126	266	393	515	639	863	1287	2298	4336

表 4 数据集 6 运用不同的算法所产生的概念格中概念的数量及构格时间

L_n	80	160	240	320	400	480	560	640	720	800
数据集 6 $L_{concept}$	134	187	575	1395	2443	5776	11399	23434	55464	158465

从表 3、表 4 中可以得出当子概念格的形式背景中其对象和属性的个数都递增时，集成后得到的全局概念格中节点的数目呈现递增的趋势。

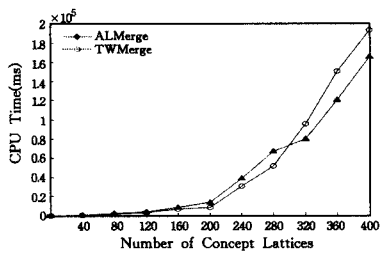

图 10 ALMerge 与 TWMerge 对于数据集 5 的子概念格进行集成耗时的对比

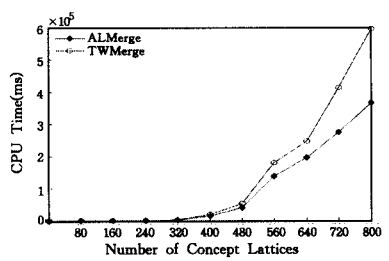

图 11 ALMerge 与 TWMerge 对于数据集 6 的子概念格进行集成耗时的对比

从图 10、图 11 中可以看出，运用两种算法所消耗的时间随着子概念格的增加也会增多，在图 10 中，当子概念格的数目小于 280 时，运用 TWMerge 算法与 ALMerge 算法耗时少，而当子概念格的数目大于 280 时，运用算法 ALMerge 所消耗的时间比算法 TWMerge 少；在图 11 中，当子概念格的数目小于 400 时，运用 TWMerge 算法与 ALMerge 算法所耗的时间相当，差别不大，而当子概念格的数目大于 400 时运用算法 ALMerge 所消耗的时间比算法 TWMerge 少，而且随着子概念数目越多，规律越明显。从图 10、图 11 可以得出，当子概念格的形式背景中其对象和属性的个数都递增时，集成后得到的全局概念格中节点的数目呈现递增的趋势，运用两种算法耗时都随着子概念格数目的增多而增加；当子概念格的数目达到一定的数量时（图 10 中大于 280，图 11 大于 400），则运用算法 ALMerge 较好。

从以上的实验数据可以得出当子概念格的数目较少时，选择算法 TWMerge，而当子概念格的数目达到一定数量（300 以上），则采用算法 ALMerge 较好，这与两种算法的理论相一致。

结束语 随着处理数据量的剧增，面对自然分布的多个数据集，在构造概念格时不再面向形式背景，而是面向概念格。对大量概念格的分布式集成，本文主要介绍了异构环境

（下转第 275 页）

梶田秀司. 仿人机器人[M]. 北京:清华大学出版社,2007

[13] Fu Gen-ping. Research on Gait Planning and Walking Control for Humanoid Robot[D]. Guangzhou:Guangdong University of Technology. 2013(in Chinese)
付根平. 仿人机器人的步态规划和步行控制研究[D]. 广州:广东工业大学,2013

[14] Tizhoosh H R. Opposition-Based Learning: A New Scheme for Machine Intelligence[C]//CIMCA-IAWTIC'06. 2005:695-701

[15] Rahnamayan S, Tizhoosh H R, Salama M M A. Opposition-based differential evolution[J]. IEEE Transactions on Evolutionary Computation,2008,12(1):64-79

[16] Zhang L, Huang Q, Lv S, et al. Humanoid motion design considering rhythm based on human motion capture[C]// 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE,2006:2491-2496

[17] Min Ying-ying, Liu Yun-gang. Barbalat Lemma and its application in analysis of system stability[J]. Journal of Shandong University(Engineering Science),2007,37(1):51-55(in Chinese)
闵颖颖,刘允刚. Barbalat 引理及其在系统稳定性分析中的应用[J]. 山东大学学报(工学版),2007,37(1):51-55

(上接第 228 页)

下概念格的分布式集成的两种算法:添加式集成算法和二路归并式集成算法。这两种方式具有各自的优缺点,当子概念格的数目较少时,可以选择算法 TWMerge,而当子概念格的数目达到一定数量(300 以上),则采用算法 ALMerge 较好。在不同的情况下,采取不同的集成方式,最终提高了概念格集成过程中的时空效率。异构环境下概念格的分布式集成中相关理论和思路,对于同构环境下概念格的分布式集成也是适用的。因此,概念格的分布式集成在现实生活中具有广阔的应用前景。

下一步的工作是模糊概念格的分布式集成的拓展,进一步构造基于模糊概念格的分布式集成算法。在此基础上,如何进一步提高概念格的构造效率,以及概念格的集成在现实中的应用也是未来的工作方向。

参 考 文 献

[1] Ganter B, Wille R. Formal Concept Analysis: Mathematical Foundations[M]. Germany:Springer-Verlag,1999

[2] Chai Yu-mei, Wang Chun-li, Wang Li-ming. An Algorithm for Mining Complement-Alternative Relationship Based on Frequent Itemsets[J]. Pattern Recognition and Artificial Intelligence,2012,25(1):157-165(in Chinese)
柴玉梅,王春丽,王黎明. 基于频繁项集的互补替代关系挖掘算法[J]. 模式识别与人工智能,2012,25(1):157-165

[3] Wang Li-ming, Zhang Zhuo. An Algorithm for Mining Closed Frequent Itemsets Based on Apposition Assembly of Iceberg Concept Lattices [J]. Journal of Computer Research and Development,2007,44(7):1184-1190(in Chinese)
王黎明,张卓. 基于 iceberg 概念格并置集成的闭频繁项集挖掘算法[J]. 计算机研究与发展,2007,44(7):1184-1190

[4] Zhang Zhuo. Research on Web Database Extraction Based on Formal Concept Analysis[D]. Wuhan: Wuhan University, 2011 (in Chinese)
张卓. 基于形式概念分析的 Web 数据库抽取研究[D]. 武汉:武汉大学,2011

[5] Fu H, Nguifo M. A parallel algorithm to generate formal concepts for large data[C]// New York: Springer-Verlag, 2004. 2014:394-401

[6] Kuznetsov S O, Obiedkov S A. Comparing performance of algorithms for generating concept lattices[J]. Journal of Experimental and Theoretical Artificial Intelligence, 2002, 14(2/3): 189-216

[7] Lindig C. Fast Concept Analysis:2000[C]// Aachen. Germany: Shaker-Verlag,2000:152-161

[8] Godin R, Missaoui R, Alaoui H. Incremental concept formation algorithms based on Galois (concept)lattices[J]. Computational Intelligence,1995,11(2):246-267

[9] Belohlavek R. Fuzzy Galois Connections[J]. Math. LogicQuarterly,1999,45(4):497-504

[10] Valtchev P, Missaoui R. Building Concept (Galois) Lattices from Parts: Generalizing the Incremental Methods[M]// Conceptual Structures: Broadening the Base. Springer Berlin Heidelberg,2001:290-303

[11] Valtchev P, Missaoui R, Lebrun P. A partition-based approach towards constructing Galois (concept) lattices[J]. Discrete Mathematics,2002,256(3):801-829

[12] Zhi Hui-lai, Zhi Dong-jie, Liu Zong-tian. Theory and Algorithm of Concept Lattice Union[J]. Actaelectronica Sinica, 2010, 38(2):455-459(in Chinese)
智慧来,智东杰,刘宗田. 概念格合并原理与算法[J]. 电子学报,2010,38(2):455-459

[13] Chai Yu-mei, Zhang Zhuo, Wang Li-ming. An Algorithm for Mining Global Closed Frequent Itemsets Based on Distributed Frequent Concept Direct Product[J]. Chinese Journal of Computers,2012,35(5):990-1001(in Chinese)
柴玉梅,张卓,王黎明. 基于频繁概念直乘分布的全局闭频繁项集挖掘算法[J]. 计算机学报,2012,35(5):990-1001

[14] Krajca P, Outrata J, Vychodil V. Parallel Recursive Algorithm for FCA[J]. Concept Lattices and Their Applications,2008,433(2008):71-82

[15] Liu Bao-xiang, Li Yan. Construction Principles and Algorithms of Concept Lattice Generated by Random Decision Formal Context[J]. Conputer Science,2013,40(6A):90-92(in Chinese)
刘保相,李言. 随机决策形式背景下的概念格构建原理与算法[J]. 计算机科学,2013,40(6A):90-92

[16] Krajca P, Outrata J, Vychodil V. Parallel algorithm for computing fixpoints of Galois connections[J]. Annals of Mathematics and Artificial Intelligence,2010,59(2):257

[17] Zhang Zhuo, Du Juan, Wang Li-ming. Load balance-based algorithm for parallelly generating fuzzy formal concepts[J]. Control and Decision,2014,29(11):1935-1942(in Chinese)
张卓,杜鹃,王黎明. 基于负载均衡的模糊概念并行构造算法[J]. 控制与决策,2014,29(11):1935-1942

[18] Ma F, Yu J, Zeng Z, et al. A Distributed Concept Lattices Vertical Union Method[J]. International Conference on Artificial Intelligence & Computational Intelligence,2010,3:469-473