非对称容差关系粗糙近似集的 GPU 加速方法

吴正江，王梦松，武星晨
（河南理工大学计算机科学与技术学院，河南 焦作 454003）

摘　要：大数据时代信息来源纷繁复杂，收集数据形成的信息系统很难保证其完整性。在越来越大的不完备信息系统（IIS）中，使用基于非对称容差关系的粗糙集理论进行知识蒸馏，提升近似集的计算速度成为其应用之前必须要解决的问题。针对非对称容差关系的冗余容差类问题，提出非对称容差关系的改进方案，设计不完备信息系统中上、下近似集的布尔矩阵表示方法，计算非对称容差关系的粗糙近似集的矩阵分块算法，并在图像处理单元（GPU）上实现近似集计算过程的加速，提高近似集的计算效率。另外，针对 GPU 存储空间有限的现状，构建不完备信息系统中对象之间的层次结构及其算法，将全局关系矩阵转换成多个局部关系矩阵，缓解计算最近容差类过程中 GPU 存储压力。在 UCI 数据集和生成数据集上的实验结果表明，基于最近容差关系的容差类数量相较于基准情况明显减少，同时，矩阵分块算法在 GPU 上实现了对近似集计算过程的有效加速，相比 CPU 串行计算和分布式并行计算，GPU 分块算法执行速度平均提高了 16.69 倍和 3.89 倍。

关键词：图像处理单元；近似集；最近容差关系；矩阵分块；不完备信息系统
中图分类号：TP18　　**文献标志码：**A　　**DOI：**10.19678/j.issn.1000-3428.0070473

GPU Acceleration Method for Rough Approximation Sets of Asymmetric Tolerance Relation

WU Zhengjiang, WANG Mengsong, WU Xingchen
(School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo 454003, Henan, China)

【Abstract】 In the era of big data, information sources are diverse and complex, making it difficult to ensure the integrity of the information systems formed from the collected data. In increasingly Incomplete Information Systems (IIS), using rough set theory based on asymmetric tolerance relations for knowledge distillation and improving the calculation speed of approximation sets are critical issues that must be addressed before their application. An improved scheme is proposed for the redundant tolerance class problem of asymmetric tolerance relations. A Boolean matrix representation method for the upper and lower approximation sets in incomplete information systems is designed, and a matrix-partitioning algorithm for calculating rough approximation sets of asymmetric tolerance relations is proposed. Furthermore, the acceleration of the approximation set calculation process is implemented on a Graphics Processing Unit (GPU) to improve the calculation efficiency of the approximation sets. Furthermore, given the limited storage space of GPUs, a hierarchical structure and its algorithm for objects in incomplete information systems are constructed, converting the global relationship matrix into multiple local relationship matrices to alleviate the storage pressure on GPUs during the calculation of the nearest tolerance classes. Experimental results on the UCI and generated datasets show that the number of tolerance classes based on the nearest tolerance relations is significantly reduced compared to the baseline case. Additionally, the matrix-partitioning algorithm effectively accelerates the approximate set calculation process for GPUs. Compared with CPU serial computation and distributed parallel computation, the GPU partitioning algorithm executes at an average speed of 16.69 times and 3.89 times faster.

【Key words】 Graphic Processing Unit (GPU); approximation sets; nearest tolerance relation; block-matrix; Incomplete Information System (IIS)

0　引言

由于数据缺乏有效的组织和收集，不可避免地会出现带有缺省值的数据，这些数据组织起来形成了不完备信息系统（IIS），这对于数据清洗与计算工作造成了困难。粗糙集理论是一种强大的数学工具，用来处理不精确、不一致的数据，于 1982 年由波兰数学家 PAWLAK[1] 提出。粗糙集理论无需先验

基金项目：国家自然科学基金（61972134，62372156）。
作者简介：吴正江（CCF 会员），男，教授、博士，主研方向为粗糙集、粒计算；王梦松、武星晨，硕士研究生。
收稿日期：2024-10-12　　**修回日期：**2025-02-13
E-mail：wuzhengjiang@hpu.edu.cn

知识就可以发现数据中的潜在规律，对数据进行处理。因此，该理论成功应用在数据挖掘[2-3]、图像处理[4]等领域。

容差关系粗糙集[5-7]是一种能够直接处理缺省值数据的粗糙集模型，在处理不精确数据中具有显著的优势。KRYSZKIEWICZM[8]首先提到容差关系粗糙集模型，这为处理不确定性数据提供了理论基础，但这种关系会导致容差类划分错误。STEFANOWSKI 等[9]提出对象间的非对称容差关系，这为在不完备信息系统中通过非对称性来描述对象间的关系提供了理论基础，但通过这种方式计算的容差类中存在冗余类。为了应对以上问题，研究人员提出限制性容差关系[10]，但难以处理属性值都是不确定值的对象。

许多学者利用非对称容差关系处理不完备信息系统。WU 等[11]通过拟单层覆盖粗糙集与容差关系粗糙集比较，改进近似算子，提高近似集计算效率。孙妍等[12]在不完备信息系统中基于极大相容块提出一种属性约简算法，将容差关系应用在属性约简上。HAMED 等[13]在不完备信息系统中利用非对称容差关系提出了 block 的概念，并以此设计计算近似集算法。ROHMAT 等[14]提出一种粗糙集的相对容差关系来处理不完整信息系统，提高数据分类的灵活性和精确性。DERIS 等[15]利用 2 个对象间的相似类和容差关系，删除了一些不完整数据达到约简效果，但这种删除不完整数据的方式会影响约简率。

粗糙集近似集是数据挖掘中特征选择和知识获取的关键步骤[16-18]。目前，提高近似集的计算效率主要有两种策略，第一种策略通过分布式技术。ZHANG 等[19]通过 MapReduce 技术实现对不完备信息系统中近似集的计算。FARYAL 等[20]提出一种规则并行方法计算近似集。RAZA 等[21]基于优势关系并行方法计算近似集。分布式技术虽然可以计算大规模数据集[22]，但系统复杂且成本较高，不能实时获得结果。另一种策略是使用 GPU 加速，ZHANG 等[6,23]提出一种复合关系粗糙集模型，并设计一种基于布尔矩阵计算复合关系近似集的方式，在多 GPU 上实现了 334.9 倍的加速效果。WU 等[24]提出一种拟单层覆盖近似集矩阵化表示方法，在 GPU 上实现了 2.16～11.3 倍的加速效果。JING 等[25]使用 GPU 加速一种最先进的基于基数排序的粗糙集近似集串行计算算法。GAO 等[26]使用 GPU 加速邻域近似集的计算过程。HU 等[27]设计基于经典粗糙集的并行属性约简算法，并在 GPU 上实现加速。大多数文献使用 GPU 加速是基于关系矩阵来计算近似集的，这种方式存在较高的时间复杂度和空间复杂度，使得计算效率不高。

为了解决非对称容差关系的冗余类问题，提高近似集的计算效率，本文的贡献如下：

1）提出一种最近容差关系来解释对象间存在的依赖关系，消除非对称容差关系下存在的冗余容差类，降低计算中的数据量，实现对不完备信息系统中的缺省值处理。

2）将计算近似集的过程矩阵化，设计一种矩阵分块算法计算近似集，可以在较小显存的 GPU 上实现对近似集计算过程的加速，节约计算空间。

1 基础知识

1.1 不完备信息系统

不完备信息系统由(U,A,V,f)四元组构成，其中$U=\{u_1,u_2,\cdots,u_n\}$是一个非空有限集，称为论域，$A=\{a_1,a_2,\cdots,a_k,d\}$是属性集，$a_k$ 称为条件属性，d 称为决策属性，V 是所有属性值的集合，$f:U\times A=V$ 是一个信息函数，$f(x,a_k)$表示对象 x 在属性值 a_k 上所表达的值。不完备信息系统如表 1 所示，$*$ 表示缺省值。

表 1 不完备信息系统

Table 1 An incomplete information system

U	a_1	a_2	a_3	a_4	d
u_1	2	*	2	1	0
u_2	1	2	2	1	1
u_3	*	2	*	1	1
u_4	2	1	1	2	1
u_5	1	2	*	1	0
u_6	1	2	2	1	1
u_7	2	1	*	2	1
u_8	2	2	2	1	0
u_9	2	*	*	1	0
u_{10}	2	*	*	2	1

在不完备信息系统中，如果对象 u 的条件属性值中不存在缺省值“$*$”，对象 u 称为可靠元[24]，表示为 u_r，若条件属性值中存在缺省值“$*$”，则对象 u 称为争议元，表示为 u_c。如表 1 中对象 u_2,u_4,u_6,u_8 是可靠元，对象 $u_1,u_3,u_5,u_7,u_9,u_{10}$ 是争议元。

1.2 非对称容差关系

非对称容差关系最早由 STEFANOWSKI 等[9]

提出，用来处理不完备信息系统的缺省值，之后经过发展形成了包含非对称关系在内的容差关系。

非对称容差关系 R 描述为：

$$R=\{(x,y)\mid a_k\in A:a_k(x)=a_k(y)\vee a_k(x)=*\}$$

非对称容差关系具有自反性，但不具有对称性和传递性。

若 R 为非对称容差关系，对于 $x\in U$ 的容差类记为 $R(x)$，关于 x 的容差类 $R(x)$ 表示为：

$$R(x)=\{y\mid(x,y)\in R\}$$

以表 1 展示的不完备信息系统为例，u_4,u_7,u_{10} 会产生 3 个容差类，分别为：$T_{u_4}=\{u_4,u_7,u_{10}\}$，$T_{u_7}=\{u_7,u_{10}\}$，$T_{u_{10}}=\{u_{10}\}$。

1.3 改进的非对称容差关系粗糙集

由于非对称容差关系存在冗余容差类，因此为解决此问题，本文对非对称容差关系做出改进，提出一种最近容差关系和最近容差类来构建近似集。

定义 1（最近容差关系） 对于 IIS=(U,A,V,f)，$U=\{u_1,u_2,\cdots,u_n\}$，$\forall u_i,u_j\in U(i\neq j)$ 满足：(u_i,u_j)，且不存在对象 u_k，使得 (u_i,u_k)，(u_k,u_j)，则称 u_i 与 u_j 是最近容差关系，表示为 $u_i\xrightarrow{N}u_j$。

不同于非对称容差关系，最近容差关系具有自反性，但不具有传递性和对称性。

定义 2（最近容差类） 设 1 个 IIS=(U,A,V,f)，$U=\{u_1,u_2,\cdots,u_n\}$，$u_r\xrightarrow{N}u_c^1\xrightarrow{N}u_c^2\xrightarrow{N}\cdots\xrightarrow{N}u_c^{n-1}\xrightarrow{N}u_c^n$ 表示一个最近容差关系序列，这个序列是以 u_r 为起点，以争议元 u_c^n 为终点，且对于 u_c^n 满足：$\nexists u_k\in U$，使得 $u_c^n\xrightarrow{N}u_k$。这样一个序列中的对象集称为最近容差类，使用 NT_{u_r,u_c^n} 表示。

定义 3（基于最近容差关系的上、下近似集） 设一个 IIS=(U,A,V,f)，$U=\{u_1,u_2,\cdots,u_n\}$，NTS=$\{\mathrm{NT}_1,\mathrm{NT}_2,\cdots,\mathrm{NT}_k\}$ 是一个最近容差关系序列集，$U/d=\{d_1,\cdots,d_j,\cdots,d_m\}$。关于 d 的最近容差关系的上、下近似集定义为：

$$\overline{\mathrm{NT}}(d)=\bigcup_{j=1}^{m}\{\mathrm{NT}_{u_r,u_c}\in\mathrm{NTS}\mid\mathrm{NT}_{u_r,u_c}\cap d_j\neq\varnothing\}\tag{1}$$

$$\underline{\mathrm{NT}}(d)=\bigcup_{j=1}^{m}\{\mathrm{NT}_{u_r,u_c}\in\mathrm{NTS}\mid\mathrm{NT}_{u_r,u_c}\subseteq d_j\}\tag{2}$$

以表 1 展示的不完备信息系统为例，根据非对称容差关系的 $u_2\xrightarrow{N}u_5$，$u_6\xrightarrow{N}u_5$，$u_5\xrightarrow{N}u_3$，$u_8\xrightarrow{N}u_3$，$u_8\xrightarrow{N}u_1$，$u_1\xrightarrow{N}u_9$，$u_4\xrightarrow{N}u_7$，$u_7\xrightarrow{N}u_{10}$。非对称容差关系到最近容差关系的转换如图 1 所示。这些最近容差关系在图 1 中左侧展示为实线，虚线为非最近容差关系的容差关系。从图 1 左侧中可以很直观地发现可靠元为起点的序列：$u_{2,6}\xrightarrow{N}u_5\xrightarrow{N}u_3$，$u_8\xrightarrow{N}u_3$，$u_8\xrightarrow{N}u_1\xrightarrow{N}u_9$，$u_4\xrightarrow{N}u_7\xrightarrow{N}u_{10}$。基于序列所表示的最近容差类为：$\mathrm{NT}_{u_{2,6},u_3}=\{u_2,u_3,u_5,u_6\}$，$\mathrm{NT}_{u_8,u_3}=\{u_8,u_3\}$，$\mathrm{NT}_{u_8,u_9}=\{u_8,u_1,u_9\}$，$\mathrm{NT}_{u_4,u_{10}}=\{u_4,u_7,u_{10}\}$，如图 1 的右侧所示。

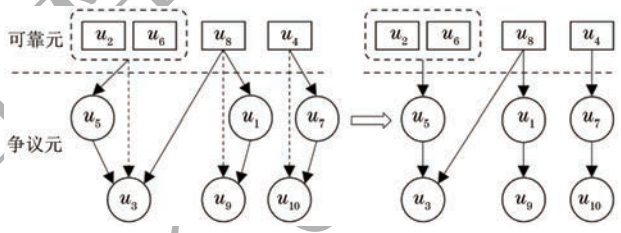

图 1 非对称容差关系到最近容差关系的转换

Fig.1 Conversion of the asymmetric tolerance relation to the nearest tolerance relation

以决策集划分的集合 $d_1=\{u_1,u_5,u_8,u_9\}$，$d_2=\{u_2,u_3,u_4,u_6,u_7,u_{10}\}$，根据定义 3，得 $\overline{\mathrm{NT}}(d)=\{u_1,u_2,u_3,u_4,u_5,u_6,u_7,u_8,u_9,u_{10}\}$，$\underline{\mathrm{NT}}(d)=\{u_1,u_4,u_7,u_8,u_9,u_{10}\}$。

2 最近容差关系的矩阵表示

2.1 矩阵表示

基于最近容差关系，定义 3 介绍了近似集的集合表示方法。为了使用矩阵的方式实现近似集计算的加速，在本节将引入基于最近容差关系计算上、下近似集的布尔矩阵方法。将近似集从集合表示形式转化为有利于 GPU 加速的矩阵表示形式。

定义 4（集合的特征表达式） 设 $U=\{u_1,u_2,\cdots,u_n\}$，$X\subseteq U$，则关于 X 的布尔表达式 $\boldsymbol{G}(X)=[r_1,r_2,\cdots,r_n]$ 定义为：

$$r_i=\begin{cases}1, & u_i\in X\\ 0, & u_i\notin X\end{cases}\tag{3}$$

定义 5（数据矩阵） 设一个 IIS=(U,A,V,f)，$U=\{u_1,u_2,\cdots,u_n\}$，$u_r\in U$，则关于 u_r 的数据矩阵 $\boldsymbol{U}_{u_r}$ 定义为：

$$\boldsymbol{U}_{u_r}=[u_1,\cdots,u_i,\cdots u_m]\tag{4}$$

数据矩阵 $\boldsymbol{X}_{u_r}$ 中对象满足：(u_i,u_r)。通过数据矩阵，构建局部关系矩阵。

定义 6（局部关系矩阵） 设 IIS=(U,A,V,f)，$U=\{u_1,u_2,\cdots,u_n\}$，$u_r\in U$ 则关于 $\boldsymbol{U}_{u_r}$ 局部关系矩阵 $\boldsymbol{M}_R=(r_{ij})_{m\times m}$ 定义为：

$$\boldsymbol{M}_R=\boldsymbol{U}_{u_r}^{\mathrm{T}}\circ\boldsymbol{U}_{u_r}\triangleq\begin{bmatrix}u_1\circ u_1 & u_1\circ u_2 & \cdots & u_1\circ u_m\\ u_2\circ u_1 & u_2\circ u_2 & \cdots & u_2\circ u_m\\ \vdots & \vdots & & \vdots\\ u_m\circ u_1 & u_m\circ u_2 & \cdots & u_m\circ u_m\end{bmatrix} \tag{5}$$

等价于 $r_{ij}=u_i\circ u_j\triangleq\begin{cases}1, (u_i,u_j)\in R\\ 0, (u_i,u_j)\notin R\end{cases}$。

局部关系矩阵的方式有助于在计算过程中降低时间复杂度和空间复杂度，提高数据集的计算效率。

推论 1 若 $u_i\xrightarrow{N}u_j$，则 u_i 的行向量与 u_j 的列向量内积为 2。这是由于 u_i 的自反性和与 u_j 非对称性的结果，通过此推论可以计算出局部关系矩阵中的最近容差关系。

定义 7（最近容差关系矩阵） 设一个 IIS=(U,A,V,f)，设 $U=\{u_1,u_2,\cdots,u_n\}$，最近容差关系矩阵 $\boldsymbol{M}_L=(l_{ij})_{l\times n}$ 定义为：

$$\boldsymbol{M}_L=(\boldsymbol{G}(\mathrm{NT}_1),\cdots,\boldsymbol{G}(\mathrm{NT}_j),\cdots,\boldsymbol{G}(\mathrm{NT}_k))^{\mathrm{T}} \tag{6}$$

式中：$\boldsymbol{G}(\mathrm{NT}_j)=(r_{j1},r_{j2},\cdots,r_{jn})$ 表示最近容差关系序列的布尔向量。

2.2 布尔矩阵运算

为构建近似集的矩阵运算，集合形式的交叉、包含等运算将被转换为布尔矩阵运算。

定义 8 设$\odot$为两个矩阵的布尔运算，$\boldsymbol{A}_{m\times n}=(a_{ik})_{m\times n}$ 和 $\boldsymbol{B}_{n\times l}=(b_{kj})_{n\times l}$ 是 2 个矩阵，$\boldsymbol{C}_{m\times l}=\boldsymbol{A}\odot\boldsymbol{B}=(c_{ij})_{m\times l}$ 定义如下：

$$c_{ij}=\bigvee_{k=1}^{n}(a_{ik}\wedge b_{kj}) \tag{7}$$

式中：$i=1,2,\cdots,m$；$j=1,2,\cdots,l$。

定义 9 设$\otimes$为 2 个矩阵的布尔运算，$\boldsymbol{A}_{m\times n}=(a_{ik})_{m\times n}$，$\boldsymbol{B}_{n\times l}=(b_{kj})_{n\times l}$，$\boldsymbol{C}_{m\times l}=\boldsymbol{A}\otimes\boldsymbol{B}=(c_{ij})_{m\times l}$ 定义如下：

$$c_{ij}=\bigwedge_{k=1}^{n}[(1-a_{ik})\vee b_{kj}] \tag{8}$$

式中：$i=1,2,\cdots,m$；$j=1,2,\cdots,l$。

定义 10 设$\circledast$为 2 个矩阵的布尔运算，矩阵 $\boldsymbol{A}_{m\times n}=(a_{ij})_{m\times n}$，$\boldsymbol{B}_{n\times l}=(b_{ik})_{m\times l}=[\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2,\cdots,\boldsymbol{\alpha}_m]^{\mathrm{T}}$，其中 $\boldsymbol{\alpha}_i=[r_{i1},r_{i2},\cdots,r_{il}]$，$\boldsymbol{C}_{m\times n}=\boldsymbol{A}\circledast\boldsymbol{B}=(c_{ij})_{m\times n}$ 定义如下：

$$c_{ij}=a_{ij}\times(\vee\boldsymbol{\alpha}_i) \tag{9}$$

式中：$\boldsymbol{\alpha}_i=r_{i1}\vee r_{i2}\vee\cdots\vee r_{il}$，$i=1,2,\cdots,m$，$j=1,2,\cdots,n$。

定义 11 设$\oplus$为矩阵的一元运算符，矩阵 $\boldsymbol{A}_{m\times n}=(a_{ij})_{m\times n}=[\boldsymbol{\alpha}_1,\cdots,\boldsymbol{\alpha}_j,\cdots,\boldsymbol{\alpha}_n]$为布尔矩阵，$\boldsymbol{\alpha}_j=[b_{1j},b_{2j},\cdots,b_{mj}]^{\mathrm{T}}$，$\boldsymbol{A}_{1\times n}=\oplus\boldsymbol{A}_{m\times n}=[\beta_1,\cdots,\beta_j,\cdots,\beta_n]$，定义如下：

$$\beta_j=\vee\boldsymbol{\alpha}_j=b_{1j}\vee b_{2j}\vee\cdots\vee b_{mj} \tag{10}$$

2.3 最近容差关系近似集的矩阵表示

定义 12（决策矩阵） 设一个 IIS=$(U,C\cup D,V,f)$，$U=\{u_1,u_2,\cdots,u_n\}$，$U/D=\{\boldsymbol{D}_1,\cdots,\boldsymbol{D}_j,\cdots,\boldsymbol{D}_m\}$为 D 在 U 上的划分。对于任意的 $\boldsymbol{D}_j\in U/D$，$\boldsymbol{D}_j=(d_{1j},d_{2j},\cdots,d_{nj})^{\mathrm{T}}$ 是关于 U 的布尔向量。决策矩阵 $\boldsymbol{M}_D$ 定义为：

$$\boldsymbol{M}_D=(d_{ij})_{n\times m}=[\boldsymbol{D}_1,\boldsymbol{D}_2,\cdots,\boldsymbol{D}_m] \tag{11}$$

定理 1 设一个 IIS=$(U,C\cup D,V,f)$，其中，$\boldsymbol{M}_D$ 为决策矩阵，$\boldsymbol{M}_L$ 为最近容差关系矩阵。

1）关于决策集 $\boldsymbol{M}_D=(d_{ij})_{n\times m}$ 的上近似集向量表示为：

$$\boldsymbol{\lambda}_{\overline{D}}=\bigcup_{i=1}^{m}\boldsymbol{\lambda}_{\overline{d}_i}=\oplus(\boldsymbol{M}_L\circledast(\boldsymbol{M}_L\odot\boldsymbol{M}_D)) \tag{12}$$

2）关于决策集 $\boldsymbol{M}_D=(d_{ij})_{n\times m}$ 的下近似集向量表示为：

$$\boldsymbol{\lambda}_{\underline{D}}=\bigcup_{i=1}^{m}\boldsymbol{\lambda}_{\underline{d}_i}=\oplus(\boldsymbol{M}_L\circledast(\boldsymbol{M}_L\otimes\boldsymbol{M}_D)) \tag{13}$$

约定 $\boldsymbol{\alpha}=(a_1,a_2,\cdots,a_n)$，$\boldsymbol{\beta}=(b_1,b_2,\cdots,b_n)$，则 $\boldsymbol{\alpha}\bigcup\boldsymbol{\beta}=(\max(a_1,b_1),\max(a_2,b_2),\cdots,\max(a_n,b_n))$。

证明 为讨论不失一般性，使用 X 代替 D，$\boldsymbol{M}_X$ 代替 $\boldsymbol{M}_D$。NTS=$\{\mathrm{NT}_1,\mathrm{NT}_2,\cdots,\mathrm{NT}_k\}$表示最近容差关系序列集，$U=\{x_1,x_2,\cdots,x_n\}$，$\boldsymbol{M}_L=(l_{ij})_{l\times n}$，$\boldsymbol{M}_X=(x_j)_{n\times 1}$，$\boldsymbol{\lambda}_{\overline{X}}=(u_i)_{1\times n}$，$\boldsymbol{M}_L\circledast(\boldsymbol{M}_L\odot\boldsymbol{M}_X)=(b_{ij})_{l\times n}$，$\boldsymbol{M}_L\odot\boldsymbol{M}_X=(a_i)_{l\times 1}=\bigvee_{k=1}^{n}(l_{ik}\wedge X_k)$。

若 $x_j\in\boldsymbol{\lambda}_{\overline{X}}$，则有 $u_j=1$，使得 $\bigvee_{i=1}^{l}b_{ij}=1$，那么 $\exists i\in\{1,2,\cdots,l\}$，使得 $a_{ij}=\bigvee_{k=1}^{n}(l_{ik}\wedge X_{kj})=1$ 且 $l_{ij}=1$，等价于，$\exists k\in\{1,2,\cdots,n\}$，使得 $X_k=1$ 且 $l_{ik}=1$，等价于，$\exists\,\mathrm{NT}_i\in\mathrm{NTS}$，使得 $x_k\in X$ 且 $x_k,x_j\in\mathrm{NT}_i$。因此，$\exists\,\mathrm{NT}_i\in\mathrm{NTS}$，使得$\mathrm{NT}_i\cap X\neq\varnothing$ 且 $x_j\in\mathrm{NT}_i$，等价于 $\exists\,\mathrm{NT}_i\in\mathrm{NTS}$，使得 $\mathrm{NT}_i\in\{\mathrm{NT}\in\mathrm{NTS}\mid\mathrm{NTS}\cap X\neq\varnothing\}$ 且 $x_j\in\mathrm{NT}_i$，等价于，$\exists\,\mathrm{NT}_i\in\mathrm{NTS}$，使得 $\mathrm{NT}_i\subseteq\bigcup\{\mathrm{NT}\in\mathrm{NTS}\mid\mathrm{NT}\cap X\neq\varnothing\}$ 且 $x_j\in\mathrm{NT}_i$。因此，$x_j\in\bigcup\{\mathrm{NT}\in\mathrm{NTS}\mid\mathrm{NT}\cap X\neq\varnothing\}$。

同理可证。

至此，完成了最近容差关系粗糙集的矩阵化。

为了说明第 2 章的向量化工作，通过以下示例来解释。

根据定义 5，数据矩阵：$\boldsymbol{U}_{u_{2,6}}=[u_2,u_3,u_5,u_6]$，$\boldsymbol{U}_{u_8}=[u_1,u_3,u_8,u_9]$，$\boldsymbol{U}_{u_4}=[u_4,u_7,u_{10}]$。局部关系矩阵将论域 U 不均等地划分成子矩阵，矩阵元素都与可靠元存在容差关系。根据定义 6，局部关系矩阵 $\boldsymbol{M}_R=(r_{ij})_{m\times m}$ 表示为：

$$\boldsymbol{M}_{u_{2,6}}=\begin{array}{c}\\u_2\\u_3\\u_5\\u_6\end{array}\begin{array}{c}\begin{array}{cccc}u_2&u_3&u_5&u_6\end{array}\\\begin{bmatrix}1&1&1&0\\0&1&0&0\\0&1&1&0\\0&1&1&1\end{bmatrix}\end{array}$$

$$\boldsymbol{M}_{u_8}=\begin{array}{c}\\u_1\\u_3\\u_8\\u_9\end{array}\begin{array}{c}\begin{array}{cccc}u_1&u_3&u_8&u_9\end{array}\\\begin{bmatrix}1&0&0&1\\0&1&0&0\\1&1&1&1\\0&0&0&1\end{bmatrix}\end{array}$$

$$\boldsymbol{M}_{u_4}=\begin{array}{c}\\u_4\\u_7\\u_{10}\end{array}\begin{array}{c}\begin{array}{ccc}u_4&u_7&u_{10}\end{array}\\\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix}\end{array}$$

根据推论 1 计算得到最近容差关系 $u_2\xrightarrow{N}u_5$，$u_6\xrightarrow{N}u_5$，$u_5\xrightarrow{N}u_3$，$u_8\xrightarrow{N}u_3$，$u_8\xrightarrow{N}u_1$，$u_1\xrightarrow{N}u_9$，$u_4\xrightarrow{N}u_7$，$u_7\xrightarrow{N}u_{10}$，则最近容差关系序列 $\mathrm{NT}_{u_{2,6},u_3}$ 表示为 $u_{2,6}\xrightarrow{N}u_5\xrightarrow{N}u_3$，$\mathrm{NT}_{u_8,u_3}$ 表示为 $u_8\xrightarrow{N}u_3$，NT_{u_8,u_9} 表示为 $u_8\xrightarrow{N}u_1\xrightarrow{N}u_9$，$\mathrm{NT}_{u_4,u_{10}}$ 表示为 $u_4\xrightarrow{N}u_7\xrightarrow{N}u_{10}$。根据定义 4，序列对应的布尔向量表示为：

$$\boldsymbol{G}(\mathrm{NT}_{u_{2,6},u_3})=[0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0]$$
$$\boldsymbol{G}(\mathrm{NT}_{u_8,u_3})=[0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0]$$
$$\boldsymbol{G}(\mathrm{NT}_{u_8,u_9})=[1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0]$$
$$\boldsymbol{G}(\mathrm{NT}_{u_4,u_{10}})=[0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1]$$

因此，最近容差关系矩阵 $\boldsymbol{M}_L$ 为：

$$\boldsymbol{M}_L=\begin{bmatrix}0&1&1&0&1&1&0&0&0&0\\0&0&1&0&0&0&0&1&0&0\\1&0&0&0&0&0&0&1&1&0\\0&0&0&1&0&0&1&0&0&1\end{bmatrix}$$

根据定理 1 得到上、下近似集向量为：

$$\boldsymbol{\lambda}_{\overline{D}}=[1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1]$$
$$\boldsymbol{\lambda}_{\underline{D}}=[1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1]$$

所得结果与第 1.3 节集合所表示的一致。

3 算法设计与复杂度分析

3.1 关系矩阵计算近似集算法

为了使用矩阵的方式计算近似集，本节设计了在 CPU 上基于传统的关系矩阵计算上、下近似集的布尔矩阵算法，并给出算法的时间复杂度和空间复杂度分析。

算法 1 传统关系矩阵计算近似集矩阵算法

输入 不完备信息系统 IIS=(U,A,V,f)

输出 决策集 D 的上、下近似集向量 $\boldsymbol{\lambda}_{\overline{D}}$ 和 $\boldsymbol{\lambda}_{\underline{D}}$

1. 构建决策矩阵 $\mathrm{M_D}=(\mathrm{d_{ij}})_{\mathrm{n\times m}}$

2. 构建关系矩阵

$$\mathrm{M_R}=(\mathrm{r_{ij}})_{\mathrm{n\times n}}=\mathrm{U^T}\circ\mathrm{U}=\begin{bmatrix}\mathrm{u_1}\circ\mathrm{u_1}&\cdots&\mathrm{u_1}\circ\mathrm{u_n}\\\vdots&&\vdots\\\mathrm{u_n}\circ\mathrm{u_1}&\cdots&\mathrm{u_n}\circ\mathrm{u_n}\end{bmatrix}$$

3. 构建最近容差关系矩阵 $\mathrm{M_L}=(\mathrm{l_{ij}})_{\mathrm{k\times n}}$

4. 计算上近似集 $\lambda_{\overline{\mathrm{D}}}=\oplus(\mathrm{M_L}\circledast(\mathrm{M_L}\odot\mathrm{M_D}))$

5. 计算下近似集 $\lambda_{\underline{\mathrm{D}}}=\oplus(\mathrm{M_L}\circledast(\mathrm{M_L}\otimes\mathrm{M_D}))$

6. 输出上近似集向量 $\lambda_{\overline{\mathrm{D}}}$ 和下近似集向量 $\lambda_{\underline{\mathrm{D}}}$

第 1 行，根据定义 12 来构建决策矩阵 $\boldsymbol{M}_D$，首先扫描决策列来获得决策值个数，再次扫描来构建决策集矩阵，所以时间复杂度为 $O(n)$。

第 2～3 行，根据定义 6 构建 $\boldsymbol{M}_R$ 矩阵，时间复杂度为 $O(n^2\times|A|)$，$|A|$ 表示属性值个数。根据定义 7 计算矩阵 $\boldsymbol{M}_L$，时间复杂度为 $O(n^2)$。

第 4～5 行，根据定义 8～定义 11，$\otimes$ 运算算子的时间复杂度为 $O(mnk)$，$\odot$ 运算算子的时间复杂度为 $O(mnk)$，$\circledast$ 运算算子的时间复杂度为 $O(m(n+k))$，$\oplus$ 运算算子的时间复杂度为 $O(mn)$。

因此，算法 1 的时间复杂度为：

$$O(n+n^2\times|A|+n^2+mnk+mnk+m(n+k)+mn)=O(n^2\times|A|)$$

空间复杂度包括 $\boldsymbol{M}_D$，$\boldsymbol{M}_R$，$\boldsymbol{M}_L$，$\boldsymbol{\lambda}_{\overline{D}}$，$\boldsymbol{\lambda}_{\underline{D}}$ 的存储空间总和，因此空间复杂度为：

$$O(mn+n^2+nk+n+n)=O(n(m+n+k))$$

3.2 CPU 分块计算近似集算法

在实际计算中，使用传统关系矩阵的方式计算上、下近似集算法需要消耗大量的内存，使得程序不能在实际环境中运行，这与算法 1 的时间复杂度和空间复杂度分析相一致。为了提高算法的计算效率和解决 CPU 内存容量有限的问题，在本节设计了计算近似集的布尔矩阵分块算法，通过将关系矩阵分解，同时将最近容差关系矩阵分批导入内存计算近似集。图 2 给出了计算近似集的矩阵分块算法流程图。

算法 2 计算近似集的矩阵分块算法(CPU)

输入 IIS=(U,A,V,f)，分块大小 S

输出 决策集 $\boldsymbol{D}$ 的上、下近似集向量 $\boldsymbol{\lambda}_{\overline{D}}$ 和 $\boldsymbol{\lambda}_{\underline{D}}$

1. 初始化近似集向量 $\lambda_{\overline{\mathrm{D}}}=(0)_{1\times\mathrm{n}}$，$\lambda_{\underline{\mathrm{D}}}=(0)_{1\times\mathrm{n}}$

2. 构建决策矩阵 $\mathrm{M_D}=(\mathrm{d_{ij}})_{\mathrm{n\times m}}$

3. 构建局部关系矩阵 $\mathrm{M_R}=\{\widetilde{\mathrm{R}}_1,\widetilde{\mathrm{R}}_2,\cdots,\widetilde{\mathrm{R}}_\mathrm{p}\}$

4. for $\widetilde{\mathrm{R}}_\mathrm{i}$ in $\mathrm{M_R}$：

5. 从每个局部容差关系矩阵中计算最近容差关系向量 $\mathrm{M}_{\widetilde{\mathrm{L}}_\mathrm{i}}=(\mathrm{l_{ij}})_{\mathrm{k\times n}}$

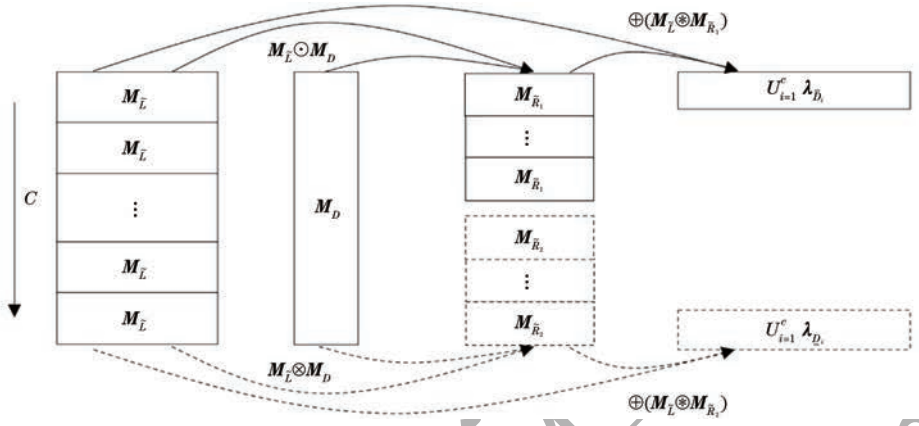

图 2 矩阵分块算法示意图

Fig. 2 Schematic diagram of the matrix-block algorithm

6. 合并结果 $M_L=M_L\cup M_{\tilde{L}_i}$

7. end for

8. 计算 CPU 最大装载行数 MAX_L

9. 计算迭代次数 $C=\lfloor k/MAX_L\rfloor+1$

10. for c in 0 to C−1：

11. 计算块的开始索引 $s=MAX_L*c$

12. 计算块的结束索引 $e=MAX_L*(c+1)$

13. 构建计算块 $M_{\tilde{L}[s:e]}=(B_{L_s},B_{L_{s+1}},\cdots,B_{L_e})^T$

14. 计算每次迭代的部分上近似集结果

$\tilde{\lambda}_{\overline{D}}=\oplus(M_{\tilde{L}[s:e]}(M_{\tilde{L}[s:e]}\odot M_D))$

15. 计算每次迭代的部分下近似集结果

$\tilde{\lambda}_{\underline{D}}=\oplus(M_{\tilde{L}[s:e]}(M_{\tilde{L}[s:e]}\otimes M_D))$

16. 合并每次迭代计算的结果

$\lambda_{\overline{D}}\vee\tilde{\lambda}_{\overline{D}}$，$\lambda_{\underline{D}}\vee\tilde{\lambda}_{\underline{D}}$

17. end for

18. 输出上近似集向量 $\lambda_{\overline{D}}$ 和下近似集向量 $\lambda_{\underline{D}}$

第 2 行，构建矩阵 $\boldsymbol{M}_D$ 的时间复杂度与算法 1 类似，时间复杂度为 $O(n)$。

第 3～7 行，根据定义 6，构造局部关系矩阵 $\boldsymbol{M}_R$ 和计算最近容差关系，时间复杂度为 $O\left(\frac{n^2\times|A|}{P}\right)$，$P$ 表示局部关系矩阵的个数。

第 10～17 行，计算决策矩阵 $\boldsymbol{M}_D$ 的近似集，首先根据输入的分块大小 S 来计算 $\boldsymbol{M}_L$ 矩阵一次可以被计算多少行，需要分多少块。通常，分块大小 S 与 $\boldsymbol{M}_L$ 矩阵大小不是倍数关系，易得前 $c-1$ 个块大小为 S，最后一个数据块大小为：$n-S(n/S-1)$。因此，需要消耗 n/S 时间计算所有数据块，对于每次迭代计算过程如下。

第 11～12 行，计算矩阵 $\boldsymbol{M}_L$ 的开始索引和结束索引，时间复杂度为 $O(1)$。

第 13 行，构建需要计算数据块的开始索引“start”和结束索引“end”，时间复杂度为 $O(e-s)$，经过 c 次迭代后时间复杂度为 $O(l)$。

第 14～17 行，计算近似集向量，时间复杂度与算法 1 相同，每次迭代的时间复杂度为 $O((e_c-s_c)(mn+mn+m)+mn+mn)$，$\sum_{c=0}^{c-1}e_c-s_c=k$，$e_c$ 和 s_c 是第 c 次迭代的开始索引和结束索引。因此，时间复杂度为 $O(nmk)$。

因此，整个程序的时间复杂度为：

$$O(n+n^2\times|A|+n^2+k+mnk+mnk+m(n+k)+mn)=O(n^2\times|A|)$$

空间复杂度相对算法 1 增加了 $M_{\tilde{p}[s:e]}$ 的存储空间，因此算法 2 的空间复杂度为：

$$O(mn+n^2+nk+n+n+(e-s)n)=O(n(m+n+k))。$$

3.3 GPU 分块计算近似集算法

第 3.2 节介绍了基于布尔矩阵分块算法计算近似集，以及算法的时间复杂度分析和空间复杂度分析，得出计算最密集部分是近似集的计算。因此，测试 9 个数据集在 CPU 下 3 个关键矩阵的计算时间比例，如图 3 所示，实验结果验证第 3.2 节中的时间复杂度分析是正确的。为了将计算密集的近似集部分迁移到 GPU 端来加速，设计了 GPU 加速近似集计算的布尔矩阵分块算法。决策矩阵 $\boldsymbol{M}_D$ 和关系矩阵 $\boldsymbol{M}_R$ 的计算仍然在 CPU 端计算。

算法 3 计算近似集的矩阵分块算法(GPU)

输入 IIS=$(U,C\cup D,V,f)$，分块大小 S

输出 决策集 D 的上、下近似集向量 $\boldsymbol{\lambda}_{\overline{D}}$ 和 $\boldsymbol{\lambda}_{\underline{D}}$

1. 初始化近似集向量 $\lambda_{\overline{D}}=(0)_{1\times n}$，$\lambda_{\underline{D}}=(0)_{1\times n}$

2. 构建决策矩阵[CPU]$M_D=(d_{ij})_{n\times m}$

3. 加载到 GPU[CPU→GPU] M_D，$\lambda_{\overline{D}}$，$\lambda_{\underline{D}}$

4. 构建局部关系矩阵 $M_R=\{\tilde{R}_1,\tilde{R}_2,\cdots,\tilde{R}_p\}$

5. for $\tilde{R}_i$ in M_R：

6. CPU 中计算每个局部容差关系矩阵中存在的最近容差关系向量[CPU]$M_{\tilde{L}_i}=(l_{ij})_{k\times n}$

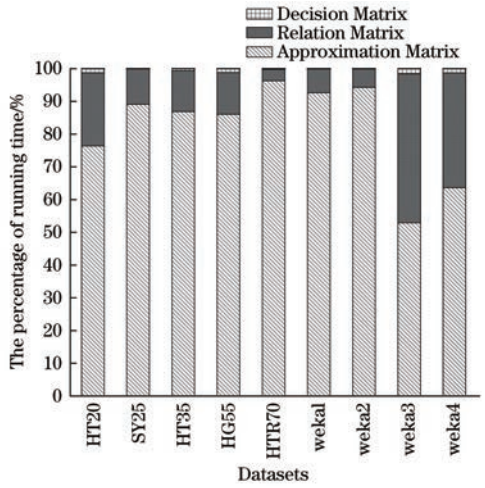

图 3 计算时间百分比

Fig. 3 The percentage of computing time

7. 结果合并 $M_L = M_L \cup M_{\tilde{L}_i}$

8. end for

9. 计算 CPU 最大装载行数 MAX_L

10. 计算迭代次数 $C = \lfloor k/MAX_L \rfloor + 1$

11. for c←0 to C−1

12. $s = MAX_L * c$ //计算块的开始索引

13. $e = MAX_L * (c+1)$ //计算块的结束索引

14. 构建计算块并传输到 GPU

[CPU→GPU] $M_{\tilde{L}[s:e]} = (B_{L_s}, B_{L_{s+1}}, \cdots, B_{L_e})^T$

15. 在 CUDA 核心中计算部分上近似集结果

[CUDA Kernel] $\tilde{\lambda}_{\overline{D}} = \oplus(M_{\tilde{L}[s:e]} \circledast (M_{\tilde{L}[s:e]} \odot M_D))$

16. 在 CUDA 核心中计算部分下近似集结果

[CUDA Kernel] $\tilde{\lambda}_{\underline{D}} = \oplus(M_{\tilde{L}[s:e]} \circledast (M_{\tilde{L}[s:e]} \otimes M_D))$

17. 合并每次迭代计算的结果

[CUDA Kernel] $\lambda_{\overline{D}} \vee \tilde{\lambda}_{\overline{D}}, \lambda_{\underline{D}} \vee \tilde{\lambda}_{\underline{D}}$

18. end for

19. 输出上、下近似集向量[GPU→CPU] $\lambda_{\overline{D}}$ 和 $\lambda_{\underline{D}}$

设 $C(\cdot)$ 表示 CPU 的计算时间，$G(\cdot)$ 表示 GPU 的计算时间，$CG(\cdot)$ 表示 CPU 与 GPU 的通信时间。算法 3 是在算法 2 的基础上将计算密集的上、下近似集部分迁移到 GPU 端加速计算，而决策矩阵 $\boldsymbol{M}_D$、局部关系矩阵 $\boldsymbol{M}_R$ 和最近容差关系序列矩阵 $\boldsymbol{M}_L$ 在 CPU 端计算。因此 CPU 端的时间复杂度为 $C(n^2 \times |A|)$。GPU 端只包含近似集的计算时间，类似于算法 1 中近似集部分的时间复杂度分析。因此，GPU 端的时间复杂度为 $G(nmk/P)$，P 是 GPU 的并行度，与 GPU 的内核架构有关。通信时间包括决策矩阵 $\boldsymbol{M}_D$ 和最近容差关系序列矩阵 $\boldsymbol{M}_L$ 与设备的通信时间，以及上近似集向量 $\boldsymbol{\lambda}_{\overline{D}}$ 和下近似集向量 $\boldsymbol{\lambda}_{\underline{D}}$ 与设备之间数据传递的时间。因此，以设备之间通信的时间复杂度为：

$$CG(nm+nk+n+n)=CG(n(m+k))$$

因此，总的时间复杂度为：

$$C(n^2\times|A|)+G\left(\frac{nmk}{P}\right)+CG(n(m+k))$$

设 HOST(•)表示 CPU 的内存使用，DEVICE(•)表示 GPU 的显存使用。算法 3 中 CPU 端包括决策矩阵 $\boldsymbol{M}_D$、局部关系矩阵 $\boldsymbol{M}_R$ 和最近容差关系序列矩阵 $\boldsymbol{M}_L$ 的存储。因此，CPU 端的空间复杂度为：$\mathrm{HOST}(nm+n^2+nk)=\mathrm{HOST}(n(m+n+k))$。另外，为了减少设备之间的数据通信次数，算法 3 中第 3 行，决策矩阵 $\boldsymbol{M}_D$、上近似集向量 $\boldsymbol{\lambda}_{\overline{D}}$ 和下近似集向量 $\boldsymbol{\lambda}_{\underline{D}}$ 被一次性地传输到显存中，算法 3 中的第 14～16 行，需要存储 $\boldsymbol{M}_{\tilde{L}}$ 矩阵，因此空间复杂度为：

$$\mathrm{DEVICE}(nm+n+n+n(e-s))=$$
$$\mathrm{DEVICE}(n+nm)$$

因此，总空间复杂度为：

$$\mathrm{HOST}(n(n+m+k))+\mathrm{DEVICE}(n+nm)$$

算法 1 是使用传统的关系矩阵计算近似集，算法 2 是在算法 1 的基础上，将关系矩阵分解为局部关系矩阵求解最近容差关系，并将最近容差序列分批次导入内存计算。算法 3 是在算法 2 的基础上把计算密集的近似集部分迁移到 GPU 端加速。图 4 给出了 3 种算法之间的关系。

4 实验分析

4.1 数据集和实验环境

表 2 给出了实验的所有数据集，前 5 个数据集来源于 UCI 标准数据集，后 4 个数据集由 Weka 数据生成器生成，其中 dataset-X 表示从 dataset 数据集中抽取 X 万条数据。

算法 1、矩阵分块算法 2 和算法 3 均采用 Python 进行编码，算法 3 在 CUDA11.1 框架[28]和 CuPy-CUDA111 10.3.1 框架[29]下运行。主机具有 32 GB 内存，CPU 的设备型号为 Inter Core i3-9100F(3.6 GHz×4)，GPU 的设备型号为 NVIDIA GeForce GTX1660，拥有 1 408 个 CUDA 核心和 6 GB RAM。

4.2 传统的关系矩阵算法实验

使用矩阵的方式计算近似集往往涉及到关系矩阵的计算，而传统关系矩阵的构建需要对象之间比较才能得出容差关系，这种比较往往导致时间复杂度和空间复杂度较高。因此，传统的关系矩阵计算效率不高。

在本节中，基于算法 1 测试了使用传统的关系矩阵计算近似集的效率。为了使数据集能够在实际

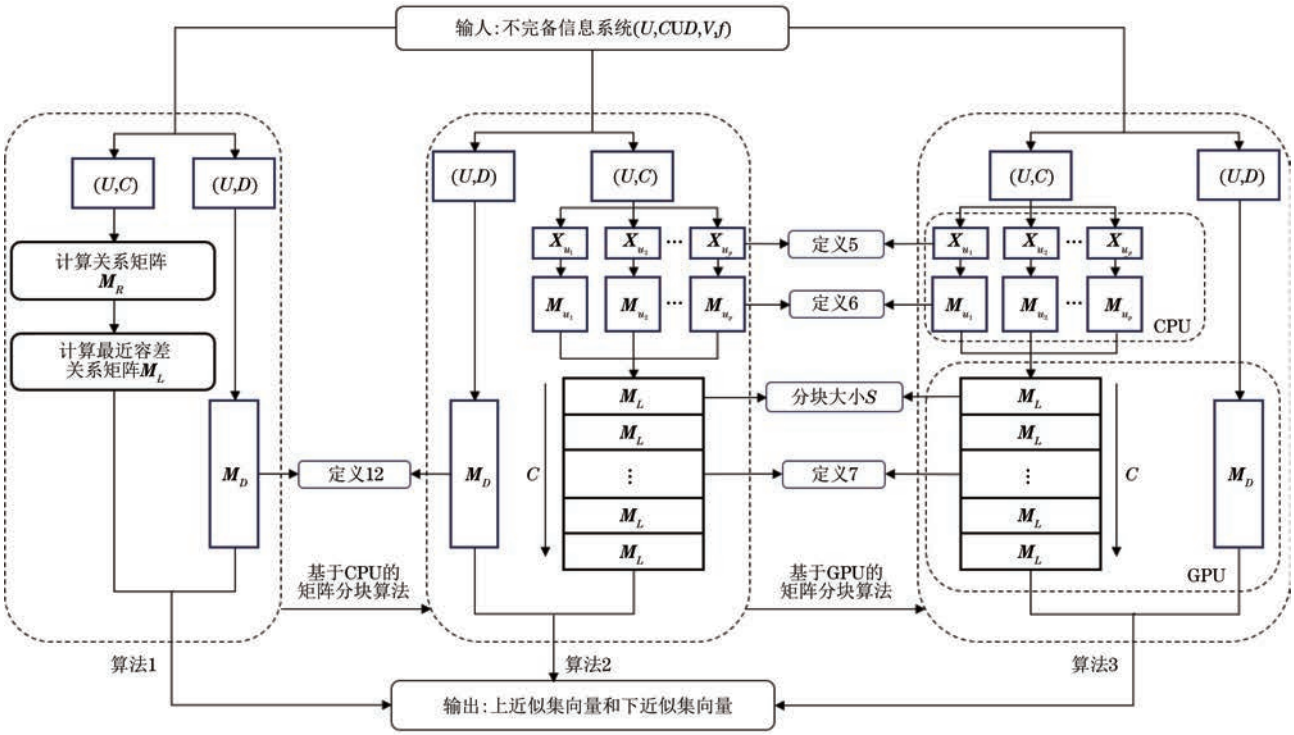

图 4　3 种算法关系图

Fig. 4　Relationship diagram of three algorithms

表 2　UCI 数据集和自定义数据集描述

Table 2　The description of UCI dataset and user-defined dataset

来源	数据集	简写	样本数/个	属性		类别数/个	M_L 行数
				属性值数/个	属性取值		
UCI	Hepmass_test-20	HT20	200 000	27	1,2	2	26 906
UCI	Susy-25	SY25	250 000	18	1,2	2	567 228
UCI	Hepmass_test-35	HT35	350 000	27	1,2	2	61 565
UCI	Higgs-55	HG55	550 000	28	1,2	2	53 858
UCI	Hepmass_train-70	HTR70	700 000	27	1,2	2	231 005
UD	Weka500k20a2c	Weka1	500 000	20	1,2	2	42 245
UD	Weka500k20a4c	Weka2	500 000	20	1,2	4	41 972
UD	Weka500k40a2c	Weka3	500 000	40	1,2	2	21 770
UD	Weka500k40a4c	Weka4	500 000	40	1,2	4	19 028

环境下运行,表 3 中的 5 个数据集被划分为不同的比例。使用传统关系矩阵的计算效率如表 3 所示。其中“—”表示在当前数据集比例下的近似集在 1 h 内无法计算出结果。

表 3　使用传统关系矩阵的计算效率

Table 3　The computational efficiency of using traditional relational matrix

数据集	计算时间/s					
	数据量的 1/256	数据量的 1/128	数据量的 1/64	数据量的 1/32	数据量的 1/16	数据量的 1/8
HT20	1.34	6.04	46.93	373.34	2 798.7	—
SY25	2.93	16.45	201.12	1 678.27	—	—
HT35	4.40	33.44	256.65	1 892.76	—	—
HG55	13.37	131.69	884.04	—	—	—
HTR70	33.96	259.14	1 892.21	—	—	—

从表 3 可以看出，传统的关系矩阵计算近似集存在较高的时间复杂度，使得数据的计算时间呈几何倍数增长。在计算近似集时出现了“内存不足”的问题。原因如下：例如，数据集 SY25 的样本数为 250 000 个，根据第 3.1 节算法 1 的空间复杂度分析，构建关系矩阵 $\boldsymbol{M}_R$ 需要 $250\ 000^2$ Btye(约 58.2 GB)的存储空间。构建最近容差关系矩阵 $\boldsymbol{M}_L$ 需要 250 000×567 228(约 13.2 GB)的存储空间，内存大小无法满足存储要求。此外，随着数据集中样本数量的增加，计算时间呈指数级增长，这与 3.1 节中算法 1 的时间复杂度分析一致，这种时间和空间的复杂度是不可接受的。因此，为了提高计算效率，设计了矩阵分块算法 2 和算法 3 来计算近似集。

4.3 矩阵分块算法的参数选择实验

为了解决传统关系矩阵时间复杂度过高的问题，以及获得合适的分块大小，基于算法 2 和算法 3 对表 2 中 5 个 UCI 数据集进行测试。在分块算法中，所有矩阵的存储方式均采用 1 Byte 的布尔存储。

实验结果表明，分块算法在 CPU 端和 GPU 端的计算时间都随着分块大小的增加而明显减少，最终趋于平缓，在一定范围内波动。但是，相应的内存和显存消耗急剧上升，这与算法 2 和算法 3 的复杂度分析相一致。原因如下：

1)在同一个数据集下，GPU 的显存消耗明显高于 CPU 内存。其原因为在矩阵分块算法中，局部关系矩阵 $\boldsymbol{M}_R$ 和决策矩阵 $\boldsymbol{M}_D$ 存在大量繁琐的计算，被放在 CPU 端来计算，而计算最密集的近似集部分被迁移到 GPU 端计算，导致显存的消耗高于内存，同时也体现了矩阵分块算法对于解决空间利用问题的可行性。

2)分块大小是影响计算时间的关键因素。分块大小增加，CPU 和 GPU 对于计算近似集的时间减少。一方面，矩阵的计算块太小，不能充分发挥 CPU 和 GPU 的计算能力，使得循环次数增加，最终导致计算时间增加；另一方面，分块大小到达某一值，使得循环次数减少，但对于每次计算块时间也增加，两者共同导致计算时间相对稳定。

为了平衡时间与空间，实验中分块大小的选择采用临界点原则。图 5 所示为时间和内存与分块的关系。图 6 所示为分块大小与处理时间和显存的关系。

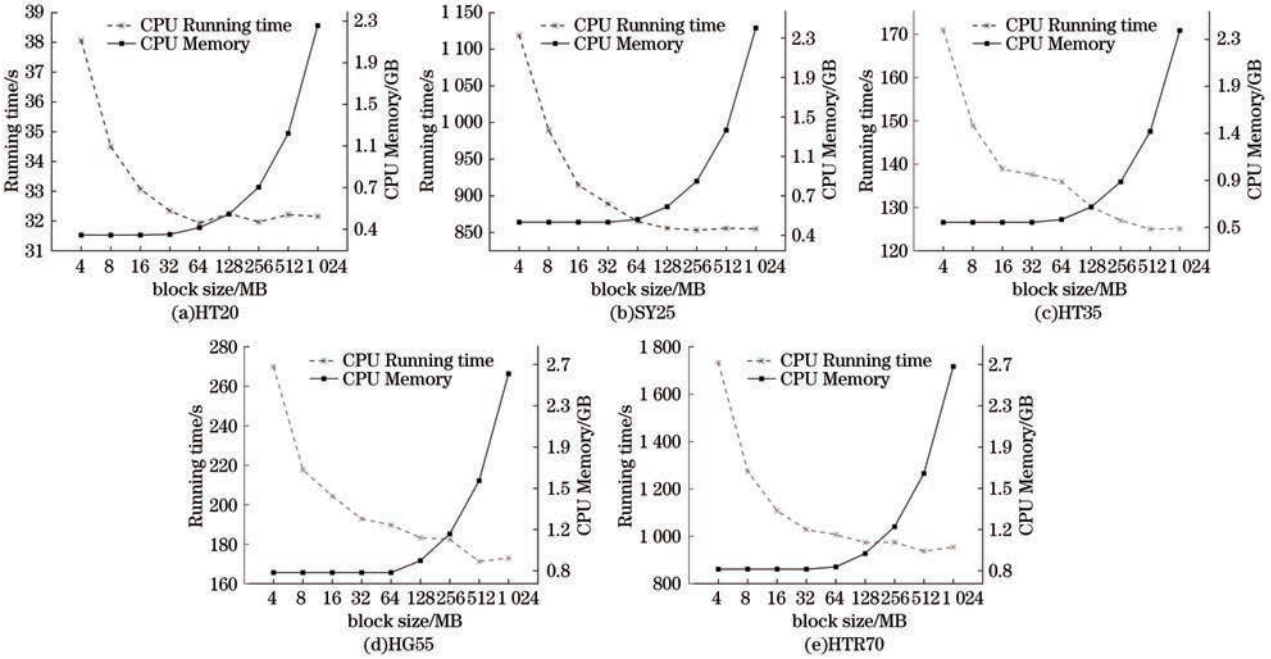

图 5 分块大小与处理时间和内存的关系

Fig. 5 The relationship between block size, processing time and memory

4.4 计算时间的对比实验

为了测试 GPU 对近似集计算的加速效果，本节基于算法 2 和算法 3，在不同的分块大小下，对表 2 中 9 个数据集下的近似集计算时间进行测试，同时，对比了分布式环境下近似集的计算时间，分布式环境具有 6 个节点，程序使用 Scala 编程，在 Spark3.1 集群中运行，每个节点拥有 60 GB 内存和 16 个核心，Hadoop 版本为 3.1.1。

在不同硬件体系下的矩阵分块算法的计算时间如图 7 所示。从图 7 可以看出，GPU 能够有效地加速近似集的计算过程，相比 CPU 实现 16.69 倍的加速效果，相比分布式算法实现 3.89 倍的加速效果，如表 4 所示。同时，近似集的计算时间也受其他因素的影响。

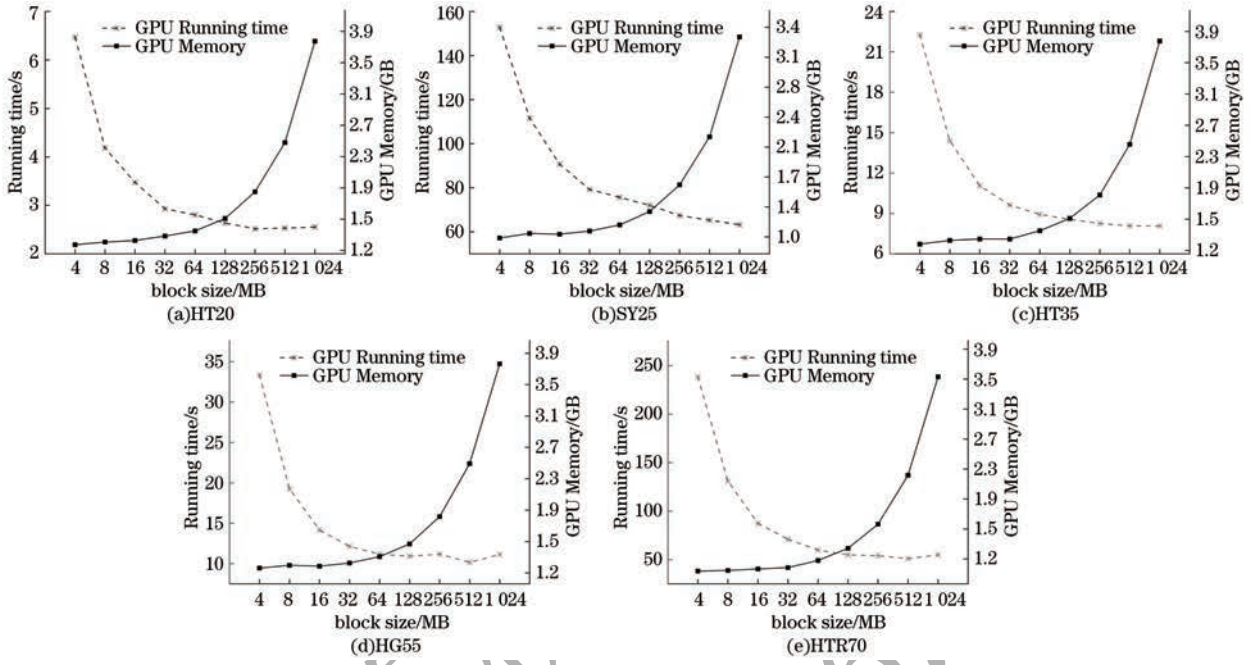

图 6 分块大小与处理时间和显存的关系

Fig. 6 The relationship between block size, processing time and video memory

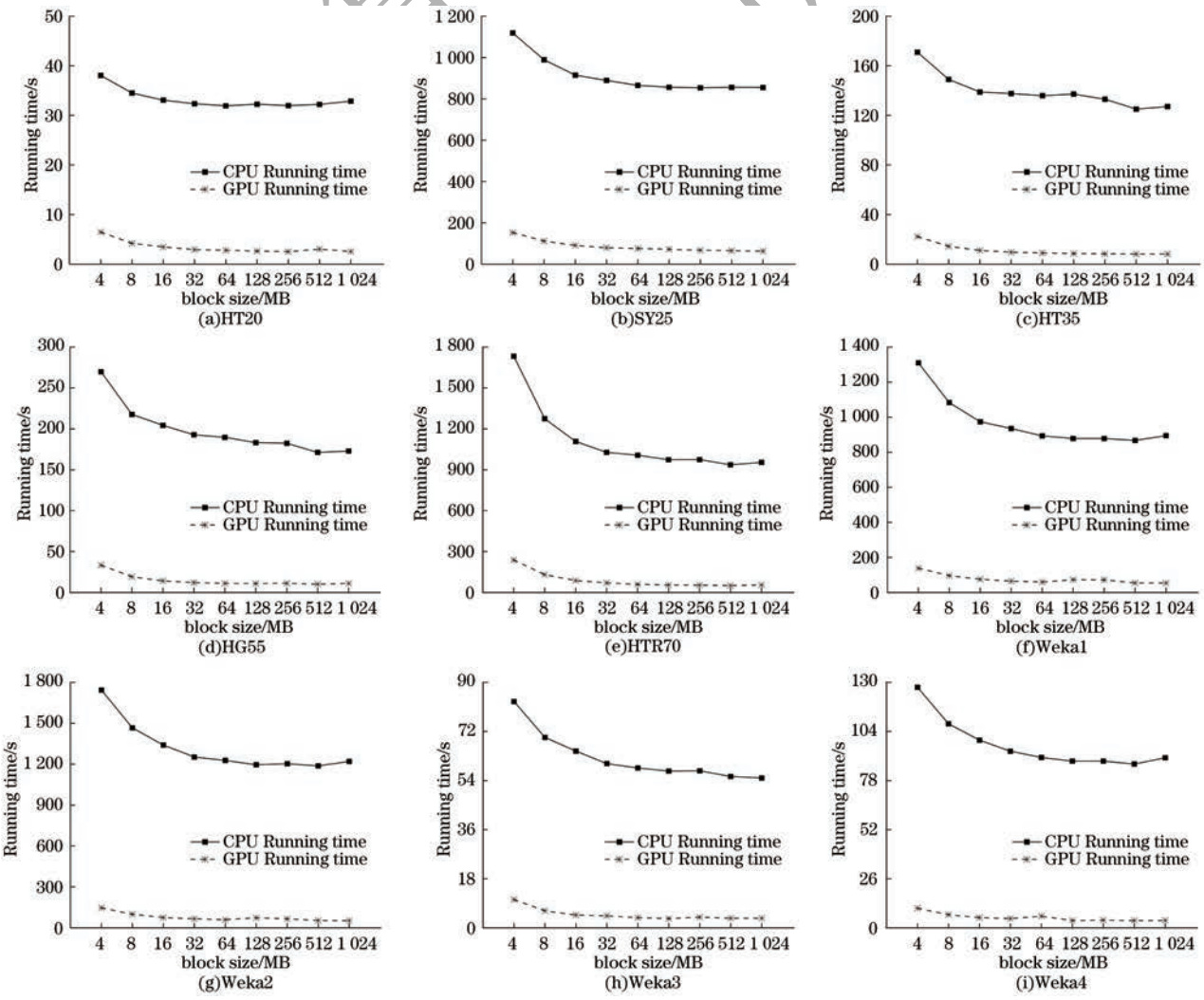

图 7 矩阵分块算法在 CPU 和 GPU 上的计算时间

Fig. 7 Computation time of matrix block algorithm on CPU and GPU

表 4 近似集的计算时间及加速比

Table 4 The computing time and speedup of approximate set

数据集	CPU 计算时间/s	GPU 计算时间/s	分布式计算时间/s	加速比(CPU/GPU)	加速比(分布式/GPU)
HT20	32.84	2.55	11.73	12.87	4.60
SY25	855.58	65.72	127.87	13.01	1.95
HT35	125.02	8.06	11.68	15.51	1.45
HG55	171.22	11.13	13.21	15.38	1.17
HTR70	936.87	54.92	14.32	17.05	1.19
Weka1	867.17	54.32	423.11	15.69	7.79
Weka2	1 188.18	52.59	426.20	22.59	8.10
Weka3	55.49	3.51	15.80	15.80	4.50
Weka4	86.75	3.88	16.59	22.35	4.28
AVG	479.90	28.52	117.83	16.69	3.89

1)分块大小对计算时间的影响。随着分块大小的增加,近似集的计算时间缩短。这是因为分块越小,一次计算的最近容差关系数据块越小,内存与显存的数据传输次数也会增加,最终导致计算次数增加。而随着分块大小的增加,计算的数据块将会增大,同时,程序的迭代次数将会减少,内存和显存之间的通信次数也会减少,最终,计算时间稳定在一定的范围。此时,CPU 和 GPU 的计算能力和迭代计算的次数达到了平衡。

2)数据量对计算时间的影响。在同一个数据集下,对比 HT20 和 HT35 数据集可以看出,数据量大小增加,加速比(CPU/GPU)也会提高,这是因为样本数增加,最近容差关系矩阵的列数增多,计算量相应增加,而 GPU 的计算能力远远高于 CPU,导致加速比提升。

3)数据集的属性值个数对计算时间的影响。SY25 和其它数据集相比,虽然 SY25 的数据量少,但是近似集计算时间很长。其原因为 SY25 相比其它数据集仅有 18 个属性值,对象之间存在容差关系的可能性更大,进而最近容差关系序列较多,导致最近容差关系矩阵的行数增多,以此带来更多的计算量。另外,数据集 Weka3 和 Weka4 的计算时间非常短,因为数据集具有更多的属性值,对象之间存在容差关系的可能性更低,进而最近容差关系序列较少,导致最近容差关系矩阵的行数减少,计算量减少。而 HG55 数据集和 HTR70 数据集样本数较大,HG55 的计算时间很少,一方面,受数据集本身的影响,另一方面,HG55 的属性值个数较多,导致容差关系序列减少,进而导致计算时间减少。

4)类别个数对计算时间的影响。Weka2 和 Weka4 有 4 个分类,加速比值较为接近,相比其它数据集有较高的加速比,这是因为类别的数量导致决策矩阵增大,计算近似集时会带来更多的计算量,导致加速比提升。因此,分块矩阵算法对于高维度和多类别的数据集具有更高的加速比。

5)在分布式环境下,9 个数据集的计算时间均高于 GPU,但是低于 CPU 的计算时间,GPU 的平均加速比相比于分布式环境提高了 3.89 倍,这也说明 GPU 矩阵分块算法的有效性。另外,Weka1 和 Weka2 在分布式环境下的加速比较高,原因在于 Weka1 和 Weka2 的属性值较少,对象间存在最近容差关系的可能性更大,产生更多的最近容差关系序列,导致计算时间增加,加速比提升,反之,Weka3 和 Weka4 的加速比(分布式/GPU)较低。在相同数据集下的 HT20 和 HT35 加速比差别较大,其原因为分布式环境能够处理大量数据,当数据增加较少的情况下,GPU 计算时间的成本要大于分布式环境。

4.5 最近容差关系与非对称容差关系的对比实验

基于算法 2 和算法 3,测试了在非对称容差关系下近似集的计算时间及非对称容差类个数($\boldsymbol{M}_L$ 行数),结果如表 5 所示,提升率表示最近容差关系相比非对称容差关系在近似集计算时间的提升。

从表 5 可以看出,相比非对称容差关系,最近容差关系在 CPU 和 GPU 计算时间均有所提升。因为最近容差关系消除了图 1 中虚线所表示的传递性,减少了容差类,通过只保留最近容差关系来计算近似集。因此,这种最近容差关系会得到更少的非对称容差类,相应地减少了 M_L 行数,进而导致计算时间减少。数据集 SY25、Weka1 和 Weka2 容差类减少最多,其原因为数据集的属性值较少,对象间存在的容差关系可能性更大,导致容差类增多。最近容差关系将会消除更多的冗余类。因此,数据集 SY25、Weka1 和 Weka2 计算时间的提升率较大,其他数据集提升较小。

表 5 最近容差关系与非对称容差关系的对比

Table 5 Comparison of nearest tolerance relation with asymmetric tolerance relation

数据集	CPU		GPU		M_L 行数	容差类减少数
	计算时间/s	提升率/%	计算时间/s	提升率/%		
HT20	33.7	2.6	3.27	2.2	28 392	1 486
SY25	1 735.32	50.7	128.38	48.8	1 156 778	589 550
HT35	137.30	8.9	8.79	8.3	66 639	5 074
HG55	183.01	6.4	11.77	5.4	55 102	1 244
HTR70	1 009.17	7.2	57.54	4.6	246 079	15 074
Weka1	1 523.06	43.1	89.05	39.0	504 998	462 753
Weka2	2 059.71	42.3	90.60	42.0	508 870	466 898
Weka3	105.25	47.3	3.59	2.2	19 287	2 483
Weka4	136.86	36.6	3.94	1.5	22 231	3 203

5 结束语

非对称容差关系粗糙集的计算多采用非对称容差类计算，而这种关系存在冗余的容差类，导致重复计算。为了解决这一问题，本文提出一种最近容差关系。另外，矩阵计算近似集往往是基于关系矩阵，这种方式的计算具有较高的时间复杂度，使得计算效率不高。本文设计一种矩阵分块算法来提高计算效率，并使用 GPU 加速。首先将计算过程矩阵化，并给出上近似集和下近似集的布尔矩阵表示方法。在计算中，使用一种分块矩阵代替传统的关系矩阵来提高计算效率。此外，设计基于 CPU 和 GPU 的矩阵分块算法计算近似集，通过实验确定合适分块大小，然后根据分块大小将近似集的计算部分迁移到 GPU 加速。实验结果表明，在基于最近容差关系计算近似集中，消除了存在的冗余容差类，所提算法在 GPU 上具有较高的加速比。

本文的研究成果有助于将 GPU 矩阵计算粗糙集的方式扩展到其它粗糙集模型中，实现快速计算近似集的目的，进而提高数据挖掘的效率。后续将最近容差关系的概念融入到特征选择中，扩大适用范围，发挥实际应用价值。

参考文献

[1] PAWLAK Z. Rough sets [J]. International Journal of Computer & Information Sciences, 1982, 11(5): 341-356.

[2] WANG G Q, LI T R, ZHANG P F, et al. Double-local rough sets for efficient data mining[J]. Information Sciences, 2021, 571: 475-498.

[3] 赵洁, 叶文浩, 梁周扬, 等. 基于不一致近邻的模糊粗糙集特征选择[J]. 计算机工程, 2024, 50(1): 110-119.
ZHAO J, YE W H, LIANG Z Y, et al. Fuzzy rough set feature selection based on inconsistent nearest neighbors[J]. Computer Engineering, 2024, 50(1): 110-119. (in Chinese)

[4] YAO W, ZHANG G, ZHOU C J. Real-valued hemimetric-based fuzzy rough sets and an application to contour extraction of digital surfaces[J]. Fuzzy Sets and Systems, 2023, 459: 201-219.

[5] HU Q H, YU D R, LIU J F, et al. Neighborhood rough set based heterogeneous feature subset selection[J]. Information Sciences: An International Journal, 2008, 178(18): 3577-3594.

[6] ZHANG J B, LI T R, CHEN H M. Composite rough sets for dynamic data mining[J]. Information Sciences, 2014, 257: 81-100.

[7] GUAN Y Y, WANG H K. Set-valued information systems [J]. Information Sciences, 2006, 176(17): 2507-2525.

[8] KRYSZKIEWICZM M. Rough set approach to incomplete information systems[J]. Information Sciences, 1998, 112(1-4): 39-49.

[9] STEFANOWSKI J, TSOUKIAS A, On the extension of rough sets under incomplete information[C]//Proceedings of International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing. Berlin, Germany: Springer, 1999: 73-81.

[10] 袁静珍, 田祥宏, 马晓. 基于多粒度决策粗糙集的云计算产品分类方法[J]. 控制工程, 2021, 28(1): 55-61.
YUAN J Z, TIAN X H, MA X. Cloud computing product classification method based on multi-granularity decision rough set[J]. Control Engineering of China, 2021, 28(1): 55-61. (in Chinese)

[11] WU Z J, WANG H, CHEN N, et al, Semi-monolayer covering rough set on set-valued information systems and its efficient computation [J]. Information Journal of Approximate Reasoning, 2021, 130: 83-106.

[12] 孙妍, 米据生, 冯涛, 等. 变精度极大相容块粗糙集模型及其属性约简[J]. 计算机科学与探索, 2020, 14(5): 892-900.
SUN Y, MI J S, FENG T, et al. Maximum consistent block based variable precision rough set model and attribute reduction [J]. Journal of Frontiers of Computer Science and Technology, 2020, 14(5): 892-900. (in Chinese)

[13] HAMED A, SOBHY A, NASSAR H. Distributed approach for computing rough set approximations of big incomplete information systems[J]. Information Sciences, 2021, 547: 427-449.

[14] ROHMAT S R, HAIRULNIZAM M, SHAHREEN K, et al. A relative tolerance relation of rough set for incomplete information system[C]//Proceedings of the 3rd International Conference on Soft Computing and Data Mining. Berlin, Germany: Springer, 2018: 72-81.

[15] DERIS M M, HAMID M A, NORAINI I, et al. Data reduction using similarity class and enhanced tolerance relation for complete and incomplete information systems[C]//Proceedings of the 10th International Conference on Information and Communication

Systems. Washington D. C., USA: IEEE Press, 2019: 134-139.
[16] 吴正江, 吕成功, 王梦松. 融合 GPU 的拟单层覆盖近似集计算方法[J]. 计算机工程, 2024, 50(5): 71-82.
WU Z J, LÜ C G, WANG M S. Calculation method for semi-monolayer covering approximation sets fushing GPU [J]. Computer Engineering, 2024, 50(5): 71-82. (in Chinese)
[17] XIA H, CHEN Z Z, WU Y M, et al. Attribute reduction method based on improved granular ball neighborhood rough set[C]//Proceedings of the 7th International Conference on Cloud Computing and Big Data Analytics. Washington D. C., USA: IEEE Press, 2022: 13-16.
[18] 危前进, 魏继鹏, 古天龙, 等. 粗糙集多目标并行属性约简算法[J]. 软件学报, 2022, 33(7): 2599-2617
WEI Q J, WEI J P, GU T L, et al. Multi-objective parallel attribute reduction algorithm in rough set [J]. Journal of Software, 2022, 33(7): 2599-2617. (in Chinese)
[19] ZHANG J B, WONG J S, PAN Y, et al. A parallel matrix-based method for computing approximations in incomplete information systems[J]. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(2): 326-339.
[20] FARYAL N, USMAN Q, MUHAMMAD S R. A parallel rule-based approach to compute rough approximations of dominance based rough set theory [J]. Engineering Applications of Artificial Intelligence, 2022, 115:105285.
[21] RAZA M S, QAMAR U. A parallel approach to calculate lower and upper approximations in dominance based rough set theory [J]. Applied Soft Computing Journal, 2019, 84: 105699.
[22] 倪鹏, 刘阳明, 赵素云, 等. 动态模糊粗糙特征选取算法[J].计算机科学与探索, 2020, 14(2): 236-243.
NI P, LIU Y M, ZHAO S Y, et al. Dynamic fuzzy rough feature selection algorithm [J]. Journal of Frontiers of Computer Science and Technology, 2020, 14(2): 236-243. (in Chinese)
[23] ZHANG J B, ZHU Y, PAN Y, et al. Efficient parallel Boolean matrix based algorithms for computing composite rough set approximations[J]. Information Sciences, 2016, 329: 287-302.
[24] WU Z J, CHEN N, GAO Y. Semi-monolayer cover rough set: concept, property and granular algorithm [J]. Information Sciences, 2018, 456: 97-112.
[25] JING S Y, LI G L, ZENG K, et al. Efficient parallel algorithm for computing rough set approximation on GPU [J]. Soft Computing, 2018, 22(22): 7553-7569.
[26] GAO Y, LÜ C W, WU Z J. Attribute reduction of boolean matrix in neighborhood rough set model [J]. International Journal of Computational Intelligence Systems, 2020, 13(1): 1473-1482.
[27] HU Y M, LI T R, HU J, et al. Parallel attribute reduction algorithms based on CUDA[C]//Proceedings of International FLINS Conference. Singapore: [s. n], 2018:1-10.
[28] NVIDIA. CUDA toolkit documentation V11. 1. 0[EB/OL]. [2024-09-05]. https://docs. nvidia. com/cuda/archive/11. 1. 0/.
[29] CuPy. CuPy API reference[EB/OL]. [2024-09-05]. https://pypi. org/ project/cupy-cuda111/10. 3. 1.

文字编辑 薛晋栋
栏目编辑 赖玉玲