Rete算法:研究现状与挑战

顾小东 高 阳
(南京大学计算机软件新技术国家重点实验室 南京 210093)

摘 要 产生式规则推理系统因其可理解性、易增删、易修改等特点而被广泛用于各种智能领域,但其规则匹配效率极其低下,不适合大规模推理。Rete算法通过规则条件共享和保存临时匹配结果大大加速了产生式推理,使其成为效率最高的产生式推理算法之一。但随着数据规模的日益增大、业务信息的频繁变更以及不完整数据和模糊逻辑的广泛出现,Rete算法也面临前所未有的挑战。基于这些背景,对Rete算法的原理、研究现状与面临的问题进行综述,指出了Rete算法的常用改进方法。介绍常见的改进方法,并对其进行分析和比较,最后总结了该算法面临的挑战,指出了未来的研究方向。

关键词 Rete算法,产生式推理系统,规则引擎
中图法分类号 TP182 **文献标识码** A

Rete Algorithm:Current Issues and Future Challenge

GU Xiao-dong GAO Yang
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)

Abstract The production system is widely used in AI. But it's not practicable in large amount of data because of its bad matching efficiency. Rete algorithm, which speeds up the match efficiency by sharing condition elements and reserving temporary matching results, has become one of the most widely used reasoning algorithms for production systems. This paper gave a survey about the principles, state of art and facing problems of Rete algorithm. Many prominent improvements of Rete algorithmare were analyzed and compared. Finally, it concludes the challenges that Rete faces and the direction of future research.

Keywords Rete algorithm, Production reasoning system, Rule engine

1 引言

随着系统逻辑的不断复杂以及环境的日渐开放和动态变化,业务逻辑也由封装代码的形式逐渐转变为额外的业务规则的形式,早期的产生式推理系统应运而生。产生式推理系统是根据用户定义的规则(产生式)和当前系统获取的事实(Facts)进行规则匹配并进行冲突消解,产生的推理结果供用户或运行系统使用的一种推理机制。如图1所示,一个产生式系统通常包括一组产生式(规则),每个产生式包括一组条件(Conditions)和一组动作(Actions),当规则的所有条件都满足时,规则的动作被加入到议程(Agenda)中等待执行。系统包括的所有产生式的集合称为产生式内存(Production-Memory)或规则库(Rule Base)以及工作内存(Working Memory)。工作内存是一个包含数据元素的全局数据库,每个数据元素(称为工作内存单元 Working Memory Element,简称WME)都可以被规则条件引用,被规则动作创建、修改、移除。产生式内存是一种长期存储的表示如何执行一项任务的知识集合,而工作内存一般包括任务实例执行时的临时信息或结果。运行时系统根据当前工作内存的事实与规则集的规则进行匹配,得到被激活的规则执行,同时可能产生新的事实。不断重复此过程,直到得到完整的推理过程与结果。

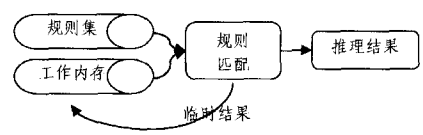


图1 产生式推理系统示意图

产生式系统的一个最大缺点是其匹配效率低下。如果按照事实和每个规则逐个匹配,匹配的时间复杂度会随规则和事实集合的增长呈指数增长。Forgy[1]证明了产生式系统90%以上的时间被用在匹配过程中。如何提高推理效率,降低存储空间,进行高效的推理,成为产生式推理系统首要解决的关键问题。在这样的背景下,各种对产生式推理系统的改进方法也应运而生。

最早对产生式系统进行优化的是静态索引方法(Index Structures),即对事实对应的规则建立索引,当再次对事实断言时可以快速找到规则。后来有学者提出增量匹配[11]的概

到稿日期:2012-01-11 返修日期:2012-04-05 本文受国家自然科学基金(61035003,61175042,61021062),国家973计划(2009CB320702),江苏省自然科学基金重点项目(江苏973计划BK2011005),教育部新世纪人才支持计划(NCET-10-0476)资助。
顾小东(1987-),男,硕士,主要研究方向为人工智能,E-mail:guxiaodong1987@126.com;高 阳(1972-),男,教授,主要研究方向为人工智能、机器学习、云计算。

念，即在推理过程中保存匹配的状态。在有新匹配时只关注事实中变化的部分。Rete 算法正是一种增量匹配的推理算法。它由 Forgy 在 1979 年提出，通过规则的模式共享以及缓存匹配状态来加速规则的匹配，但 Rete 算法也面临状态重复保存等问题，不利于大规模数据、快速可变数据和逻辑的处理。本文在介绍 Rete 算法及其研究现状后，分析 Rete 算法面临的问题以及挑战。

本文第 2 节详细介绍 Rete 算法的概念、步骤等；第 3 节分析 Rete 算法的研究现状，对不同的改进进行分析归类；第 4 节分析 Rete 算法目前面临的问题与挑战，并对未来的研究作了一个展望；最后总结全文。

2 Rete 算法

2.1 基本概念

为了解决产生式推理引擎效率低下的问题，Forgy 在 1979 年提出 Rete 算法[1]作为产生式系统的高效的模式匹配算法。Rete 算法的初衷是：利用规则之间各个域的公用部分减少规则存储，同时保存匹配过程的临时结果以加快匹配速度。为了达到这种效果，算法将规则拆分，其中每个条件单元作为基本单位（节点）连接成一个数据辨别网络，然后将事实经过网络筛选并传播，最终所有条件都有事实匹配的规则被激活。

网络共有 5 类节点：Root 节点、Type 节点、α 节点（也称单输入节点）、β 节点（也称双输入节点）、Terminal 节点等，图 2 中显示了各个节点的表示。根节点（RootNode）代表整个 Rete 网络的入口，它可以让所有的事实通过，并传递给 ObjectType 节点。作为根节点的后继，ObjectType 节点用于选择事实的类型，将符合本节点类型的事实向后继的 α 节点传播。α 节点（也称 1-input Node）主要进行同对象类型内属性的约束（Intra-conditions）或常量测试（Literal Restriction），比如“name==Zhang”，“age>15”等等。β 节点（也称 2-input Node）主要根据不同对象之间的约束（Inter-conditions）如“p. name==c. friend”，“p. age>cat. age”等进行连接操作。β 节点又分为 Join 节点、Not 节点等。Join 节点包括两种输入，左部输入事实列表，称为元组（Tuple），右部输入一个事实对象，对象与元组在 Join 节点按照类型间约束进行 Join 操作，将符合的事实加入元组中继续传入下一个 β 节点。Terminal 节点是规则的末尾节点，它代表一个规则匹配结束，当事实或元组传递到 Terminal 节点时表示该 Terminal 节点对应的规则被激活。图 2 是一个具体的 Rete 网络的例子。它包含以下两个规则：

R_1：if
c:Coffee(type==“instant”)
    p:Person(money>Coffee. price)
then
p. buy(c)

R_2：if
c:Coffee(type==“instant”)
    p:Person(money<Coffee. price)
then
p. return(c)

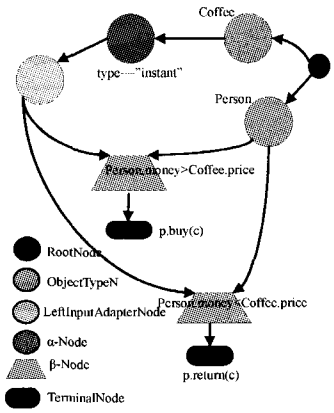


图 2 一个 Rete 网络的例子

2.2 算法步骤

Rete 算法的步骤主要分为两个部分：规则网络的建立和事实的匹配。其中，网络建立过程算法如下：

(1)创建根。

(2)取出一个规则 r。

a. 取出一个模式 p，检查参数类型，如果是新类型，则加入一个类型节点；

b. 检查模式 p 的条件约束，对于单类型约束，检查对应的 α 节点是否已存在，如果存在则记录下节点位置，否则将该约束作为一个 α 节点加入链的后继，所有 α 处理完后连接 α 内存；

c. 检查模式 p 的域，若为多类型约束，则创建相应的 β 节点，其左输入为前一 β 节点（第一个 β 节点左部输入为空），右输入为当前链的 α 内存；

d. 重复 b—c，直到模式 p 的所有约束处理完毕；

e. 重复 a—d，直到所有的模式处理完毕，创建 Terminal 节点，每个模式链的末尾连到 Terminal 节点；

f. 将动作（Then 部分）封装成叶节点（Action 节点）作为输出节点。

(3)重复(2)直到所有规则处理完毕。

运行时的匹配过程如下：

(1)从工作内存中取一 WME 放入根节点进行匹配。

(2)遍历每个 α 节点（含 ObjectType 节点），如果 α 节点约束条件与该 WME 一致，则将该 WME 存在该 α 节点的匹配内存中，并向其后继节点传播。

(3)对 α 节点的后继结点继续(2)的过程，直到 α 内存所有通过匹配的事实保存在 α 内存中。

(4)对每个 β 节点进行匹配，如果单个事实进入 β 节点左部，则转换成一个元素的元组存在节点左侧内存中。如果是一个元组进入左部，则将其存在左内存中。如果一个事实进入右侧，则将其与左内存中的元组按照节点约束进行匹配，符合条件则将该事实对象与左部元组合并，并传递到下一节点。

(5)重复(4)直到所有 β 处理完毕，元组对象进入到 Terminal 节点。对应的规则被触活，将规则后件加入议程（Agenda）。

(6)对 Agenda 里的规则进行冲突消解，选择合适的规则执行。

2.3 算法特点

Rete 算法通过共享规则节点和缓存匹配结果，获得产生式推理系统在时间和空间上的性能提升。其优点可以总结如下[4]：

• 通过节点共享，减少或消除了特定类型的规则或事实冗余。

• 进行连接操作时存储部分的匹配结果，这反过来也使得产生式系统在每次有事实变化或增删的时候都可以避免对

整个事实集的重评估，只需评估工作内存中变化的事实。

·在事实从工作内存撤销时，Rete 允许记录单元有效地从网络中移除。

3 研究现状

Rete 算法自从 1979 年提出以来，已经经历过各种改进与推广。除了对自身规则网络结构的优化外，对一些功能扩展如模糊推理、事件处理、并行化等也有很多研究。

3.1 结构优化

结构优化亦即通过在原始 Rete 算法的规则网络上增加或修改一些结构，来提高网络的灵活性、高效性等。Rete 从提出至今，已经经历过不同的演化[13]。对 Rete 本身网络结构的优化一直是研究的重要课题。主要的结构优化可以总结如下。

3.1.1 混合逻辑符的处理

逻辑操作符(operators)是指诸如 and、or、not 等的逻辑符号。最初的 Rete 算法只默认处理条件之间为 and 操作符的规则。Drools[2] 中可以支持 and 和 or 操作符的混合，但只是将规则从 or 操作符处切分为多个规则。这种方法会增加很多的切分开销。文献[5]在这种拆分之前将规则前件进行了一定的预处理，即对规则左部构建语法树，同时对树进行一定变换，把 or 连接提到顶端，并对树进行一定约减。最终从顶端的 or 节点开始拆分规则，这样减少了拆分的个数，也降低了规则网络的复杂度。

文献[3,19]对 Rete 规则的语法结构进行了扩展，即根据规则的 and、or 等逻辑操作将规则构建成一个抽象语法树(Abstract Syntax Tree)，并在语法树的基础上建立 Rete 网络。逻辑操作符本身作为一个 Rete 节点参与事实匹配。这种做法减少了额外的规则拆分开销，也给规则匹配带来了便利。

3.1.2 规则前件的重排序[11,13,14,21]

规则前件顺序是指规则条件中的各个约束的排列顺序，它决定了条件连接操作的执行顺序，影响中间结果的大小，是决定规则匹配的效率的关键因素。另外，好的连接顺序可以增加节点共享，进而减少临时事实元组[13]。例如 2.1 节例举的规则 R_1 中，如果将条件“c:Coffee(type==‘instant’)”放在条件“p:Person (money >coffee. price)”之后，则规则 R_1 与 R_2 的 α 节点无法在 Rete 网络中共享。

总结起来，对规则前件的重排序主要有以下方法：

1. 限制性强的条件优先：限制性强的前件置前可以尽早地过滤更多的事实，从而让网络的临时事实数量减少。

2. 不稳定的条件置后：在匹配时频繁地变更匹配事实的节点位置可以减少后继的增删次数，从而提高效率。

3. 变量分组：有相同变量的条件元素应该分成一组，这样避免了笛卡尔连接效应[11]，而且因连接操作受变量一致性绑定的约束而使其产生更少的中间结果。

但是这些排序方法都在一种或局部场景下发挥其优点，不能有效地结合。文献[14]针对这种问题提出了一种成本模型(cost model)的方法，即对产生式系统中规则网络的节点增加参数，统计每个节点经过的事实数，测试成功的事实数、内存平均大小等数据，并建立一种成本模型(cost model)，根据这些统计参数计算相应的成本，最后选择成本最低的排序方式。这种做法解决了各种排序原则结合的问题。但是它是在规则运行前进行训练再排序，属于一种静态网络(compile-time optimization)，不能在运行时结合实际运行效果进行动态调整。TREAT 算法是对 Rete 的一种改进方法，支持在线的调整方案称为种子排序(seed ordering)[15]，关于条件排序问题特别是在线的顺序调整问题还有待于进一步研究。

3.1.3 索引方法

索引方法是指对 Rete 网络的节点建立当前节点对后继的索引，在事实断言时可以通过索引快速找到对应的后继节点而无需逐个查找。Drools[2] 在 Rete 的面向对象版本 Rete-OO 算法中对 ObjectType 节点增加后继 α 节点的索引，以事实的属性为 key，α 节点为 value，这样在事实通过类型节点验证后可以迅速找到对应的 α 节点进行断言。同样，对 β 节点也可以建立索引，β 节点的索引主要是针对节点左右内存的查询。当一个事实传递到 β 节点的右内存中时，需要与该节点的左内存进行连接操作，即遍历左侧内存中的事实元组，找到符合节点约束的事实进行连接。该过程的遍历查找效率较低，很多学者对此问题进行了索引化的改进。例如文献[26]将 β 内存分成若干单元，每个单元分配一个 id；对右侧的事实用哈希函数求索引，该索引就是某个单元的位置，通过索引快速找到相应单元进行匹配，如果不在该分区，则将该对象组成一个新的单元加入左内存。

3.2 功能扩展

Rete 算法最初用于优化产生式的匹配，是在确定性的一阶布尔逻辑下进行规则匹配。然而当前各种其他逻辑的广泛应用、数据的缺失、模糊逻辑等问题对传统规则引擎的功能提出了新的要求。为了既能表示这些扩展逻辑又能利用 Rete 高效的匹配算法，近年来设计出各种 Rete 算法的功能扩展，典型的有三值布尔逻辑、F-逻辑、处理事件推理等等。

3.2.1 处理其他逻辑

Rete 最初只用于处理一阶布尔逻辑，目前有很多 Rete 的扩展被用来处理其他的逻辑，如 F-逻辑(F-logic)、三值布尔逻辑(3-Valued Boolean)等。Florian 等人研究在演绎数据库中用 Rete 进行面向集合的自底向上的规则评估[10]。在 F-逻辑规则评估引擎 Florid 中用 Rete 作为一种可行的评估技术。他们的研究证明，在许多规则共享子目标的情况下可以用 Rete 方法获取执行时间的提升量。三值布尔逻辑广泛用于缺损数据的推理。Sottara 采用泛化的逻辑以及参数化的配置来扩展 Rete 网络，通过对网络的参数配置使 Rete 网络能够处理三值布尔逻辑[3]。当然，功能上的逻辑扩展更多来源于具体的数据特点及处理需求。3.3 节将针对各种特殊数据类型对逻辑扩展作进一步介绍。

3.2.2 带时间信息的事件处理

Rete 通过事实来表达当前的状态，但是很多应用包含一些事件流，这些事件流中时间在事件的并发执行中起到关键作用。然而 Rete 算法不提供这种对时间敏感的事实的处理[28]。许多关于 Rete 算法引入事件处理的研究也应运而生[7-9]。事件一般包括 3 种类型：警告(alarms)、警告确认(confirmations of alarms，表示该警告依然存在)和警告解除(cancellations of alarms)[8]。文献[28]引入含时间戳的事件以及事件间时间间隔约束(temporal constraints)的概念，使得 Rete 算法可以同时处理事实和事件。

3.3 特殊数据的推理

3.3.1 瑕疵数据与不确定性推理

产生式系统在许多场合，特别是对问题领域已经有充分理解的情况下表现得很成功[22]，对这些领域的知识可以用一些基于逻辑的语言表示出来。传统的逻辑中，一个问题的判断结果非真即假。但现实中许多问题本身是不精确的，定义模糊或有歧义，我们称之为瑕疵数据(Imperfect Data)，例如规则"如果年级大则容易患中风"。产生式推理系统将不能精确表达"年级大"及"容易"这样的概念，相应的推理也不能得到精确的结果。这种场合下，传统的产生式推理(包括 Rete 算法)变得很脆弱。因为产生式推理系统的语义表达能力相当有限，所以必须对 Rete 进行一定的扩展，以处理瑕疵数据和不确定性逻辑[22]。通常这一类瑕疵主要包括以下几点：

1. 不确定性(Uncertainly)：由于缺少对一个事实或事件的足够认识，推理不能给出确信的结果，而会给出各种可能性的其他选择。通常，结果的不确定性可以被统计并且可以用概率来衡量。

2. 不精确性(Imprecision)：由于对知识的定义不具体，导致了规则的歧义、含糊或近似。

3. 不一致性(Inconsistency)：规则集中含有冲突的信息，但其不能精确地描述和区分。

处理不确定性问题通常使用概率方法，例如将"容易患中风"表述成"患中风的概率 70%"。不精确问题常用模糊集(Fuzzy Set)的概念来描述，例如将"张三年纪大"表述为"张三对老年人的隶属度为 0.8，对中年人的隶属度 0.2"等。经过模糊化的描述后，可以用模糊逻辑进一步对这些规则进行推理。而不一致性问题通常采用的方法是进行规则的冲突消解。例如同时有矛盾的规则进入了冲突集，用冲突检测机制(通常的做法是对规则引入偏好度)选取用户更偏好的规则执行。

DavideSottara 等人[3,19]将 Rete 处理的规则逻辑由布尔逻辑泛化到一般的模糊逻辑，主要泛化过程包括：引入不确定性谓词、泛化操作符、泛化演绎推理。在事实传播过程中也引入了更多的评估机制，对 RETE 网络增加了相应的配置参数，通过不同的参数配置将传统的处理确定性推理的 Rete 算法推广到不确定性推理，包括各种模糊逻辑、缺损数据如三值布尔逻辑、贝叶斯推理等。这种扩展大大提高了 Rete 算法处理模糊缺失数据的能力。

3.3.2 快速变化数据与机器学习

除了数据有瑕疵，对于变化剧烈的数据也成为 Rete 算法亟需处理的问题。Rete 算法通过保存临时匹配结果来记录当前状态，网络中的每个节点都记录了通过该节点的事实。在数据变化剧烈的情况下，这种临时事实将剧烈改变，带来很大的计算开销，也让 Rete 失去了这种增量匹配的优势。

现有的一种解决方式是从数据中寻找规则逻辑。文献[24]通过决策树学习算法从数据中学习规则。这种方法本质上并不是对 Rete 算法的改进，但是在处理快速变化数据这样的问题上，这种采用机器学习从数据中获取规则的方法对改进 Rete 仍有很大的意义。有关快速变化的数据处理问题还有待于未来更多的研究。

3.4 并行化

Rete 算法从提出至今，伴随着数据规模的增长，其效率提升问题一直是研究的重点。多核多处理机面世后，将推理过程分配到不同机器上并行处理成为一种常见的效率提升方法。早在 20 世纪 90 年代，就有学者对一般规则引擎的并行化推理进行了研究[27]。Rete 的并行化包括匹配的并行化、执行的并行化。早期的并行化研究关注于规则的分组和并行匹配。Stolfo 在 DADO 机上实现了 4 种并行匹配算法及 1 种并行执行算法[29]。Fu-Chiung Cheng 等人[6]提出一种并行化方法。通过分析规则间的依赖关系，建立规则依赖矩阵以及执行矩阵，为规则匹配和规则激活提供了并行化方法。

近年来，随着云计算的研究与应用，规则推理引擎在云平台上如何处理大规模数据也相应地成为研究重点。特别是 Google 提出的 MapReduce 模型[18]问世后，很多学者对于传统的人工智能方法在其上的并行化实现上做了很多工作。张琦等人[30]在 MapReduce 平台上针对大规模规则和数据设计出了 Rete 算法的并行化实现。

4 研究挑战与展望

Rete 算法的节点共享和状态缓存大大提高了匹配效率，然而也有很多缺点[11]。

1. 存在状态重复保存的问题，比如匹配过模式 1 和模式 2 的事实要同时保存在模式 1 和模式 2 的节点缓存中，将占用较多空间并影响匹配效率。

2. 处理逻辑有限，仅有一阶布尔逻辑。

3. 处理大规模数据和快速变化的数据时效率较低。

尽管在过去的几十年间，对 Rete 及一般的产生式推理引擎有过大量研究和改进，但是我们实际面临的数据和逻辑依然以出乎意料的速度增长。这既推动了对 Rete 推理方法的革新，也为 Rete 今后的研究提出了更大的问题和挑战。

4.1 处理大规模数据或大规模逻辑规则

处理海量数据无疑是推理系统乃至整个计算机领域面临的重大问题。随着算法的不断演变、数据量与业务逻辑的不断增多，以及计算平台日新月异的革新，特别是近年来随着网格计算、云计算的发展，对算法的海量数据处理能力不断提出了新的问题和挑战。Rete 算法作为规则推理引擎的一个常用算法，在海量数据处理能力上的发展一直是关注的焦点。

从算法的本身改进来讲，可以在匹配过程中用一些启发式方法，也可以引入一些随机算法、近似算法的技术来提高速度。在一些大规模应用如服务发现、服务选择中，对算法的精度要求并不很高，而是关注于最快地找到一个可行解。这类方法将能有效提高算法的性能。

从数据平台和编程模型的角度，这种传统的规则推理引擎也应该更多地从以逻辑为中心转向以数据为中心，借助数据的特点对算法的模式进一步改进以使其适应于分布式平台或云平台进行并行化处理。3.4 节介绍了 MapReduce 模型下的并行化，但面临日新月异的分布式、云计算平台，算法还需要更多的改进与调整。比如如何对规则之间的依赖性关系进行分析以适应规则拆分，对规则进行层次化分组以提高并行度等等。这一类问题必将成为今后研究的一大挑战。

4.2 处理快速变化的数据

除了数据规模的增长外，数据应用的反应性需求也是日渐重要的问题。Rete 的一个主要缺点就是不适合处理快速变化的数据和规则。主要表现在：

1. 数据变化引起节点保存的临时事实频繁变化，这将让Rete失去增量匹配[11]的优势。

2. 数据的变化使得对规则网络的种种优化方法如索引、条件排序等失去效果。

3. 第3节介绍了目前有些学者通过机器学习从数据中获取规则的方法[24]，其取得了很好的效果。但是关于Rete算法本身如何进行改进以应对这种结构和功能上的自适应，还没有相关研究。

我们相信，随着机器学习技术的广泛使用，可以将逻辑处理更多地关注到数据中，即通过数据分析来应对变化，例如在服务选择中，预先用机器学习技术挖掘各类服务的特点、分布。在推理过程中对不同分布的服务信息按照其特点进行推理、选择。自适应的技术同样可以用于推理的优化。例如给Rete网络若干自适应参数，在推理过程中对数据进行统计分析，并利用数据特征来对规则网络或匹配过程进行动态自适应。通过这种对数据的分析和挖掘以及自适应网络调整来提高推理的效率和准确性。

4.3 处理模糊逻辑或缺失数据

贝叶斯网络等统计推理技术由于其在处理不确定性问题及模糊逻辑方面的出色表现已经在工程上得到广泛应用，相比这些基于统计的方法，规则引擎依然有它的长处：易理解、易修改、计算量小等。然而它的缺点也逐渐明显：不能处理缺失的数据及模糊的逻辑。如何发挥两者的长处，即让Rete这样的确定性推理算法处理模糊缺失数据的问题近年来也相应地得到研究者的重要关注[3]并一度成为研究热点。3.4节说明了近年来在这个方面的研究趋势。在未来，这种对Rete的随机化、经验化进行改进以适应不确定性的推理问题依然需要更深入的研究。

目前研究的关注点在于逻辑的泛化，包括增加参数等方法，未来的研究可以结合更多的模糊推理技术如贝叶斯网络、概率图推理等，使Rete算法既能发挥产生式推理引擎易理解、易修改的优点，又具备更强的表达能力和逻辑处理能力。

结束语 规则引擎是人工智能的一个重要分支，它的出现极大地便利了知识表示、数据推理、专家系统等。Rete算法的提出极大地提高了规则引擎的推理效率，并为其他相关领域的研究提供了新技术和新思路，研究前景广阔。近年来，虽然仍有很多研究人员致力于该方面的研究，发表了很多新的研究成果，但随着业务逻辑的日益复杂、数据的日益复杂庞大及新的计算平台如分布式和云计算平台的日益普遍，目前的方法将面临诸多问题：(1)处理海量数据与规则；(2)处理快速变化的数据；(3)模糊逻辑和缺失数据的推理。本文对产生式推理系统中一个具有代表性的算法Rete作了一个比较全面的综述，分析了Rete算法的各种改进方法、研究现状和待解决的问题。对这类问题的研究与解决必将进一步推动数据推理乃至人工智能领域的新发展。

参 考 文 献

[1] Forg C L. Rete: a fast algorithm for the many pattern/many object pattern match problem[J]. Artificial intelligence, 1982, 19: 17-37

[2] Apache Drools Project. Apache Drools Expert User Guide[OL]. http://docs.jboss.org/drools/release/5.3.0.Final/droolsjbpm-integration-docs/html_single/index.html, 2008-09-07

[3] Sottara D, Mello P, Proctor M. A configurable Rete-OO engine for reasoning with different types of imperfect information[J]. IEEE Transactions on knowledge and data engineering, 2010, 22(11): 1535-1548

[4] Wikipedia[OL]. http://en.wikipedia.org/wiki/Rete_algori-thm, 2009-09-08

[5] Xiao Ding, Tong Yi, Yang Hai-tao, et al. The Improvement for Rete Algorithm[C]// Proc of The International Conference on Information Science and Engineering (ICISE 2009). NJ: IEEE, 2009: 5222-5226

[6] Cheng Fu-chiung, Chen Huei-huang, Perng J-H. Parallel Execution on Production Systems[C]// Proc of the IEEE Second Symposium on Parallel and Distributed Processing. NJ: IEEE, 1990: 463-470

[7] Walzer K, Breddin T, Groch M. Relative temporal constraints in the Rete algorithm for complex event detection[C]// Proc of the Second International Conference on Distributed Eventbased Systems. New York: ACM, 2008: 147-155

[8] Zhou Dong-dai, Fu Yi-fan, Zhong Shao-chun, et al. The Rete Algorithm Improvement and Implementation[C]// Proc of the 2008 International Conference on Information Management, Innovation Management and Industrial Engineering. NJ: IEEE, 2008: 426-429

[9] Walzer K, Groch M, Breddin T. Time to the Rescue-Supporting Temporal Reasoning in the Rete Algorithm for Complex Event Processing[C]// Proc of the 19th international conference on Database and Expert Systems Applications. Berlin: Springer, 2008: 635-642

[10] Florian S, Nour S, Georg L. Adapting the Rete-algorithm to evaluate F-Logic rules[C]// LNCS 4824: Proc of the International Symposium on Rule Interchange and Applications (RuleML'07). Berlin: Springer, 2007: 166-173

[11] Lagun E. Evaluation and Implementation of match algorithms for rule-based multi-agent systems using the ecample of Jadex [D]. Hamburg: University of Hamburg, 2010

[12] Boyer J, Mili H. Agile Business Rule Development Process, Architecture, and JRulesExamples[M]. Berlin: Springer, 2011: 161-174

[13] Scals D J. Efficient Matching Algorithms for the SOARlOPS Production System[OR]. ftp://reports.stanford.edu/pub/cstr.old/reports/cs/tr/86/1124/CS-TR-86-1124.pdf, 1986

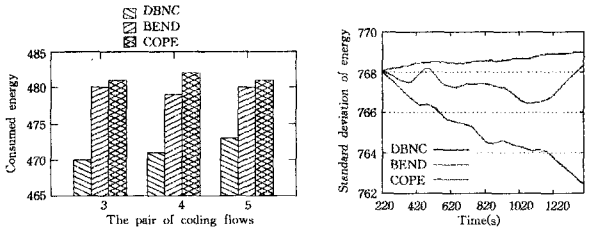
[14] Ishida T. An optimization algorithm for production systems[J]. IEEE Transactions on Knowledge and Data Engineering, 1994, 6(4): 549-558

[15] Brownston L, Farrell R, Kant E, et al. Programming Expert Systems in OPS5: An Introduction to Rule Bused Programming [M]. MA: Addison-Wesley, 1985

[16] Tamasmeszaros, Vadasz B. An extension to the Rete match algorithm: supporting both forward and backward chaining[OL]. http://www.mit.bme.hu/~ meszaros/me/pubs/tempus94.ps.gz, 2012-02-03

[17] Rangel P, Junior J G C, Ramirez M R, et al. Context reasoning through a multiple logic framework[C]// Proc of the Sixth International Conference on Intelligent Environments. NJ: IEEE, 2010: 116-121

(下转第33页)



(a) 能量最低节点的平均能耗　(b) 4 对可编码流时节点能量的标准差值

图 5　节点能耗的比较(实验 2)

表 3　4 对可编码流时能量最低节点消耗一定能量的平均时间

单位能量 \ 协议	DBNC	BEND	COPE
100	255.3	250.6	249.7
200	508.9	501.4	499.2
300	764.1	752.1	750.3
400	1016.9	1003.4	999

结束语　为了延长异构无线传感器网络的生命周期，同时保障数据的可靠高效传输，本文提出了一种基于区分服务的网络编码协议(DBNC)。该协议周期性地计算节点剩余能量的均值和标准差，对节点进行动态分类。在此基础上，将服务质量需求映射到网络编码协议中。一方面，引入能量感知编码包转发机制，以高效地利用节点的能量；另一方面，设计多优先级数据包调度策略，目的是创造更多的编码机会。仿真结果表明，DBNC 不仅保障了数据的可靠传输，而且有效地减少了低能量节点的能耗，从而延长了网络的生命周期。DBNC 与 BEND 和 COPE 协议相比具有更好的性能，而且 DBNC 对异构无线传感器网络环境具有自适应的特点。下一步的工作是研究多媒体传感器网络基于区分服务的网络编码协议。

参 考 文 献

[1] Akyildiz I, Su W, Sankarasubramaniam Y, et al. A survey on sensor networks[J]. IEEE Communications Magazine, 2002, 40(8): 102-114

[2] Duarte-Melo E J, Liu M. Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks[C]// Proceedings of IEEE GLOBECOM. New York, NY, USA, March 2002: 21-25

[3] Smaragdakis G, Matta I, Bestavros A. SEP: A stable election protocol for clustered heterogeneous wireless sensor networks [C]// Proceedings of SANPA. Boston, MA, USA, August 2004: 1-11

[4] Keshavarz-Haddad A, Riedi R. Bounds on the benefit of network coding: throughput and energy saving in wireless networks[C]// Proceedings of IEEE INFOCOM. Phoenix, AZ, USA, April 2008: 376-384

[5] Katti S, Rahul H, Hu W, et al. XORs in the air: practical wireless network coding[C]// Proceedings of ACM SIGCOMM. Pisa, Italy, September 2006: 243-254

[6] 陈贵海，李宏兴，等. 多跳无线网络中基于网络编码的多路径路由[J]. 软件学报，2010，21(8)：1908-1919

[7] Matsuda T, Noguchi T, Takine T. Survey of Network Coding and Its Applications[J]. IEICE Transactions on Communications, 2011, E94. B(3): 698-717

[8] 夏卓群，陈志刚，等. 无线 Mesh 网中网络编码的研究进展[J]. 计算机工程与应用，2010，46(12)：1-4

[9] Ahlswede R, Cai N, Li S R, et al. Network Information Flow [J]. IEEE Transactions on Information Theory, 2000, 46(4): 1204-1206

[10] Zhang J, Chen Y P, Marsic I. MAC-layer Proactive Mixing for Network Coding in Multi-hop Wireless Networks[J]. Computer Networks, 2010, 54(2): 196-207

(上接第 12 页)

[18] Da Dean J, Dean J G S, Ghemawat S. MapReduce: Simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113

[19] Sottara D, Mello P, Proctor M. Adding Uncertainty to a Rete-OO Inference Engine[C]// LNCS 5321: Proc. Int'l Symp. Rule Representation, Interchange and Reasoning on the Web (RuleML2008). Berlin: Springer, 2008: 104-118

[20] Kim M, Kim M. Fast Service Selection using Rete Network in Dynamic Environment[C]// Proc of World Conference on Services 2009. NJ: IEEE, 2009: 85-92

[21] Özacar T, Öztürk Ö, Osman ünalir M. Optimizing a Rete-based Inference Engine using a Hybrid Heuristic and Pyramid based Indexes on Ontological Data[J]. Journal of Computers, 2007, 2(4): 41-48

[22] Luger G F, Chakrabarti C. Knowledge-Based Probabilistic Reasoning fromExpert Systems to Graphical Models [OR]. http://www. cs. unm. edu/~ luger/23-Luger-Chakrabarti. pdf, 2010-06-09

[23] Bouaud J. TREE: The heuristic driven join strategy of a RETE-like matcher[C]// Proc of Thirteenth International Joint Conference on Artificial Intelligence. New York: ACM, 1993: 496-503

[24] Oguz G, Proctor M. Decision Tree Learning for Drools[OR]. http://infoscience. epfl. ch/record/126292/files/oguz-thesis _ final. pdf? version=1, 2008-04-07

[25] Nickles M, Sottara D. Approaches to Uncertain or Imprecise Rules-A Survey[C]// LNCS 5858: Proc of RuleML2009. Berlin: Springer, 2009: 323-336

[26] Xiao Ding, Zhong Xiao-an. Improving Rete Algorithm to Enhance Performance of Rule Engine Systems[C]// Proc of 2010 International Conference on Computer Design and Appliations (ICCDA 2010). NJ: IEEE, 2010: 572-575

[27] Stolfo S J. Five Parallel Algorithms for Production System Execution on the DADO machine[C]// Proc. of National conference of A. I. AAAI, 1984: 300-307

[28] Berstel B. Extending the RETE Algorithm for Event Management[C]// Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02). Washington: IEEE Computer Society, 2002: 49-52

[29] Stolfo S T, Shaw D E. DADO: A Tree-structured Machine for Production Systems[C]// Proc. of National conference of A. I. AAAI, 1982: 242-246

[30] Cao Bin, Yin Jian-wei, Zhang Qi, et al. A MapReduce-based architecture for rule matching in production system[C]// Proc of the 2nd IEEE International Conference on Cloud Computing Technology and Science. NJ: IEEE, 2010: 790-796