学号：201510223120

上海海事大学本科毕业设计（论文）

多维关联分析算法在港口仿真数据挖掘上的应用

学院：物流工程学院
专业：物流工程
班级：工物 152
姓名：俞李韵
指导教师：添玉
完成日期：2019 年 5 月 28 日

承 诺 书

本人郑重承诺：所呈交的《多维关联分析算法在港口仿真数据挖掘上的应用》是在导师的指导下，严格按照学校和学院的有关规定由本人独立完成。文中所引用的观点和参考资料均已标注并加以注释。论文研究过程中不存在抄袭他人研究成果和伪造相关数据等行为。如若出现任何侵犯他人知识产权等问题，本人愿意承担相关法律责任。

承诺人（签名）：_____

日期：　　年　　月　　日

摘 要

本文主要探讨的是挖掘算法中的关联规则分析，从原理、属性、引理等方面探讨分析了两种主流的关联规则分析方法：Apriori 算法和 FP-Growth 算法。选取更加适合此目标的 FP-Growth 算法，并在此基础上，推出了一个 Upgrade-FP-Growth 改进算法，主要面向具有“输入与输出”两块数据组特点的港口仿真数据。将其运用在多维港口数据分析中，选取一部分数据为代表，对其进行算例演示与分析，对于港口仿真系统的设计自查和修改提供了理论依据和改进方向。

关键词：数据挖掘；关联规则；多维

Abstract

This paper mainly discusses the analysis of association rules in mining algorithms. From the aspects of principle, attribute and lemma, it analyzes two mainstream association rules analysis methods: Apriori algorithm and FP-Growth algorithm. The FP-Growth algorithm, which is more suitable for this goal, is selected. Based on this, an upgrade-FP-Growth algorithm is proposed, which is mainly for port simulation data with two data sets of “input and output”. It is applied in multi-dimensional port data analysis, and some data is selected as representative. It is used to demonstrate and analyze the example, which provides a theoretical basis and improvement direction for the design and modification of the port simulation system.

Key words: Data mining; association rules; multi-dimension

目录

1 引言

1.1 研究背景

随着大数据时代的到来，各行各业对于数据挖掘的需求与日俱增，现在发展得最迅猛的方面就要数计算机业、商业、医学等领域。但是在港口行业中的发展进度就有些不尽人意了。但是数据的激增意味着数据的存储和加工处理的时间控制就成了一个大难题。同时，从另外一个角度来看就意味着新的发展机遇。作为一门交叉性学科，数据挖掘会涉及到很多领域，同时也会推动各个领域的进一步发展。

港口管理者也渐渐意识到对于港口发展的重要性，以及对于港口数据的储存与分析的运用。合理地对港口数据地分析可以对港口的高运作效率有更加好的改进空间。虽然最直接也是最有效的方式就是购买或者更新港口的各种装卸设备。但是由于集装箱码头的装卸设备一般都比较昂贵，所以一般不能通过无限制购进设备来满足箱量需求和提高码头 服务水平。其实自新中国成立以来，中国沿海港口的发展大体可划分为恢复发展建设期、快速发展建设期、高速高等级发展建设期、平稳发展建设期 4 个阶段。恢复发展建设期（1949—1979 年）：我国港口建设以扩建、改造老码头为主。对外贸易逐年扩大，我国港口建设了一些深水原油码头，扩建、新建了一批万吨级以上散杂货和客运码头。快速发展建设期（1980—1999 年）：加大了对沿海港口建设的投入，并研究提出了沿海主枢纽港布局规划，指导建设了一批专业化码头。高速高等级发展建设期（2000—2010 年）：随着我国加入世贸组织，促进经济的高速发展和临港工业的兴起，这一时期沿海港口建设高速发展，高等级码头及航道建设提速明显，港口吞吐量、深水泊位数均增长迅猛，沿海港口吞吐量年均增速达 16.25%，2010 年达 54.8 亿吨；万吨级以上泊位数增至 1343 个。这一时期，依靠科技创新，我国筑港技术显著提高，复杂环境下的深水航道、离岸深水港建设等技术已居世界先进水平。 发展至今，最热门的港口研究方向主要是港口的自动化研究。包括自动调度，自动响应的港口装卸设备等，尽可能地最大化港口码头的装卸效率。

根据德勤在 2015 年推出的《全球港口和航运业 2030 展望》中，未来港口建设会朝着物流数字化的方向发展，而现在信息科技正在飞速改变现有物流的组成方式。由于物流活动正日趋复杂，信息流数字化的需求与日俱增。同时，数据分析及数据交换正在成为港口一项新的相对优势。从亚洲的视角来看，未来预计到 2030，亚洲占全球 GDP 的

份额将会上升至 40%以上。且国际贸易路线将继续改变亚洲大陆，亚洲将会主导国际航运业。

1.2 研究目的及意义

从现代数字化发展的进程来看，社会、移动通信、分析学及云技术方面的飞速进步会将港口的数据分析计算带入下一个阶段。而港口计算机系统演变为能感知、监视、控制人类及物理环境的互联系统。

现阶段对于港口的研究一般集中在港口的各种装卸设备的单个或联合调度研究以及数据挖掘的遗传算法来对数据的趋势做预测分析。但是遗传算法做出的预测模型需要十分大的数据样本量，并且港口数据十分复杂及多样化，所以在研究初期很难得出一个可用的预测模型算法。关联算法分析可以在初期对像港口数据这样具有多维高复杂度属性的数据提供基本的可行分析模型。

本文研究了最基本的 Apriori 算法以及常用的 FP-Growth 算法，分析两者利弊。根据性能更优的 FP-Growth 算法，提出了一个改进算法：Upgrade-FP-Growth 算法。此算法主要面向具有“输入与输出”两块数据类特点的港口仿真数据。在一定数量的维数下，Upgrade-FP-Growth 算法时间和空间两个方面可以表现出较优的性能。

1.3 国内外研究现状

数据挖掘现如今由于人工智能的大火而随之迅速发展，但在港口分析领域依旧缺少进步。这主要是由于其数据的特殊性，其中包括多维、多层、数据量大以及情况多样等特点。由此，若要将逐渐成熟的数据挖掘技术运用于物流港口领域还需要将算法进行改变。而在这几种算法中，要数优化和关联分析最为重要。本论文对于现有的优化关联规则算法进行改进，使此算法更加适合运用于港口数据的挖掘。

对于关联算法最基础也是最著名的 Apriori 算法 ，即对数据先计算支持度产生候选频繁项集，再通过计算可信度确定此层级的频繁项集，以此循环直到层级结束。经过一段时间的文献阅读以及整理，目前 Apriori 算法有两大发展方向 ：1、解决处理数据量过大的大内存占用问题。此方向可参考《FP-growth 算法的实现方法研究》 中的分而治之法，分而治之法（divide and conquer）的原理是将一个大问题分成若干个小问题，逐个击破。其主要分为三个步骤：分——将问题分解为规模更小的子问题；治——将这些规模更小的子问题逐个击破；合——将已解决的子问题合并，最终得出“母”问题的解。2、有目标性地挖掘数据，即有候选项集的挖掘。此方向可参考文献《Finding Interesting Associations

without Support Pruning》采用了最小哈希的方法，摒弃了 Apriori 的剪枝步骤，在提前给定的项集内逐个接进频繁项集的范围中。

同时，关联规则的分类也影响着算法的改进，其分为多层和多维两种特征。由于港口物流的数据以多层为主要特点，所以本文主要致于解决多维数据挖掘的算法改进。多维关联规则主要分为两种：维间关联规则和混合关联规则。例如 维间关联规则是指不允许维同时出现在 X 和 Y 里面，而混合关联规则是指允许维在 X 和 Y 中重复出现。解决多维的数据分析，采用 FP-Growth 的算法框架较好，通过调整最小支持度的数据设定，从而分别挖掘出数据的维间关联规则和混合关联规则。文献《一个高效的多维关联规则挖掘算法》即为一篇典型的多维数据挖掘算法的改进方向。其将维谓词集索引树(DPI 树)的原理引入多维关联规则的并发挖掘，通过构建一个特殊节点链的指针表，可实现超大规模数据库的并发、多维挖掘。对实现港口系统信息自动化及其它数据挖掘应用领域都具有极其重要的指导意义。

1.4 论文主要内容及结构

主要内容：

本文分析了港口仿真数据的特点:多维、高复杂性以及具有“输入与输出”两块数据类。基于 FP-Growth 算法推出了一种针对此特点的数据的新改进算法：Upgrade-FP-Growth 算法，并将此算法应用于港口仿真数据的分析中。

论文结构如图所示：

第一章绪论部分。首先阐述了本文选题的背景、研究目的与意义，最后介绍了国内外的研究现状。

第二章为两种基本算法的理论介绍与分析。首先对关联规则的定义、基本算法进行简要地介绍；其次分别对 Apriori 算法与 FP-Growth 算法的原理、构造流程以及伪代码进行了简要的说明，包括 FP-Growth 采用的数据存储结构 FP-Tree 的结构过程；最后将 Apriori 算法与 FP-Growth 算法从时间复杂度等角度进行分析比对，阐述其优劣点。

第三章为 Upgrade-FP-Growth 算法的架构介绍与分析。对于研究内容进行了问题分析，并介绍基于 FP-Growth 算法的改进算法：Upgrade-FP-Growth 算法，包括其原理、伪代码、时间复杂度分析以及其采用的数值标记法。

第四章为算例演示。在庞大的港口仿真数据中，选取一部分，对其进行算例演示与分析。

第五章为总结与展望。对于全文做了总结，指出了一些不足之处，并对今后的工作做出了展望。

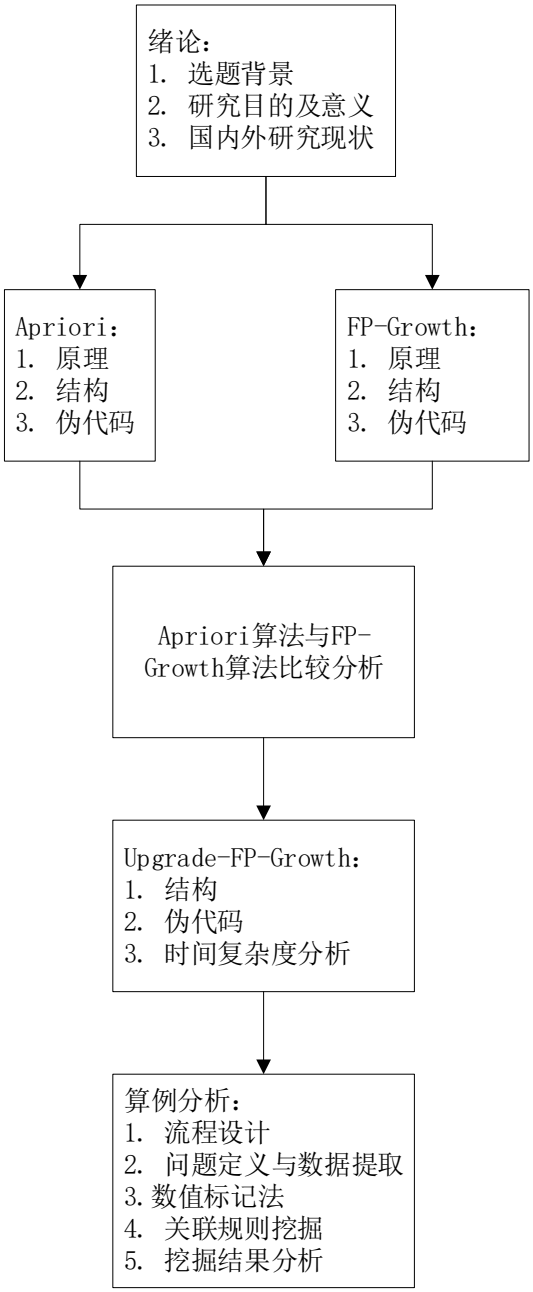

图 1.1 论文结构图

2 关联规则分析算法

2.1 关联规则基本介绍

关联规则是挖掘数据库中两个或者多个数据项存在的强关联联系。在 1993 年由 Agrawal 首先提出关联规则分析的概念，以后学者们以 Apriori 算法为基础进行了大量的研究。主要的方向有：1.改进 Apriori 算法的效率，即尽可能地缩小时间复杂度。2. 研究数值型字段地预处理。3. 减少内存的占用等。

关联规则的基本定义有以下几点[4]：

定义 1（项集和交易集）：设 $I=\{i_1,i_2,\cdots,i_k\}$ 为数据项(item)的集合，具有二进制的属性；并设 $D=\{T_1,T_2,\cdots,T_K\}$ 为交易(transaction) T 的集合，交易 T 是数据项的集合，每个 t 是一个二进制矢量集，并 $T\subseteq I$ 。

定义 2（支持度）：支持度为 $support(A\Rightarrow B)=P(A\cup B)$ 。其中 $A\subset I,B\subset I$ ，且 $A\cap B=\varnothing$ 。支持度是一种十分重要的度量，因为支持度十分低的规则可能只是偶然出现。

定义 3（置信度）：置信度为 $confidence(A\Rightarrow B)=P(B|A)=support(A\Rightarrow B)/support(A)$ 。其中 $A\subset I,B\subset I$ ，且 $A\cap B=\varnothing$ 。置信度度量是通过规则进行推理具有可靠性，对于给定规则，置信度越高，B 包含在 A 的事务中出现的可能性就越大。置信度也可以估计 B 在给定 A 下的条件概率。

定义 4（强关联规则）：其目标是从上一步发现的频繁项集中提取所有高置信度的规则，称为强规则。

2.2 有候选项集的产生方法：关联规则经典算法 Apriori

Apriori 是有有候选项集的产生算法的最经典的基础算法[5]。根据关联规则的定义，遍历全部的支持度与置信度，产生候选项集，并且再次遍历产生候选项集产生新的候选项集。Apriori 所用的结构称为格结构（lattice structure）将所有可能的候选项集的集合都存在这个结构中。从空间储存的方面来说，如果此数据库包含 k 个数据项。那么 Apriori 大约可能产生 2^k-1 个频繁项集，不包括空集在内。因为在很多现实的真实数据库中 k 的值可能十分庞大，那么由此可知需要产生的候选数据项项集可能是呈指数规模的。而根据 Apriori 的操作流程，发现频繁项集需要确定格结构中每个候选项集(candidate itemset)的支持度计

数，并将每个候选项集与每个事务进行比较。如果候选项集包含在事务中，则候选项集的支持度计数增加。Apriori 频繁集算法需要进行 $O(NMw)$ 次比较，其中 N 是事务数，$M=2^k-1$ 是候选项集数，而 w 是事务的最大宽度。

定理 1 先验原理　如果一个项集是频繁的，则它的所有子集一定也是频繁的。

例如，假定{c,d,e}是频繁项集，则它的子集{c,d},{c,e},{d,e},{c},{d}和{e}一定是频繁的。

定理 2　单调性　令 J=2^I 是 I 的幂集，度量 f 是单调的(或向上封闭的)，如果 $\forall A,B\in J:(A\subseteq B)\rightarrow f(A)\leq f(B)$，则表明如果 A 是 B 的子集，则 $f(A)$ 一定不会超过 $f(B)$。并且，f 是反单调的（向下封闭）。如果 $\forall A,B\in J:(A\subseteq B)\rightarrow f(B)\leq f(A)$ 表示如果 A 是 B 的子集，则 $f(B)$ 一定不超过 $f(B)$。

Apriori 算法的伪代码[6]如下:

算法 2.1　Apriori 产生频繁项集

1: $k=1$
2: $F_k=\{i\mid i\in I\wedge\sigma(\{i\})\geq N\times minsup\}$　{发现所有频繁项1-项}
3: **repeat**
4: $k=k+1$
5: $C_k=apriori-gen(F_{k-1})$　{产生候选集}
6: **for** 每个事务 $t\in T$ **do**
7: $C_t=subset(C_k,t)$　{识别属于t的所有候选}
8: **for** 每个候选集 $c\in C_t$ **do**
9: $\sigma(c)=\sigma(c)+1$　{支持度计数增值}
10: **end for**
11: **end for**
12: $F_k=\{c\mid c\in C_k\wedge\sigma(c)\geq N\times minsup\}$　{提取频繁k-项集}
13: **until** $F_k=\varnothing$
14: $Result=\cup F_k$

算法 2.2　Apriori 产生规则

1: **for** 每一个频繁 k-项集 f_k，$k\geq 2$ **do**
2: $H_1=\{i\mid i\in f_k\}$　{规则的1-项后件}
3: **call** $ap-genrules(f_k,H_1)$
4: **end for**

算法 2.3　$ap-genrules(f_k,H_m)$ 定义

```
1:   k = | f_k |        {频繁项集的大小}
2:   m = | H_m |        {规则后件的大小}
3:   if k > m + 1 then
4:      H_{m+1} = apriori - gen(H_m)
5:      for 每个 h_{m+1} ∈ H_{m+1} do
6:         conf = σ(f_k) / σ(f_k - h_{m+1})
7:         if conf > minconf then
8:            output:规则 (f_k - h_{m+1}) → h_{m+1}
9:         else
10:           从 H_{m+1} delete h_{m+1}
11:        end if
12:     end for
13:   call ap - genrules(f_k, H_{m+1})
14:   end if
```

Apriori 流程图如图 2.1 所示：

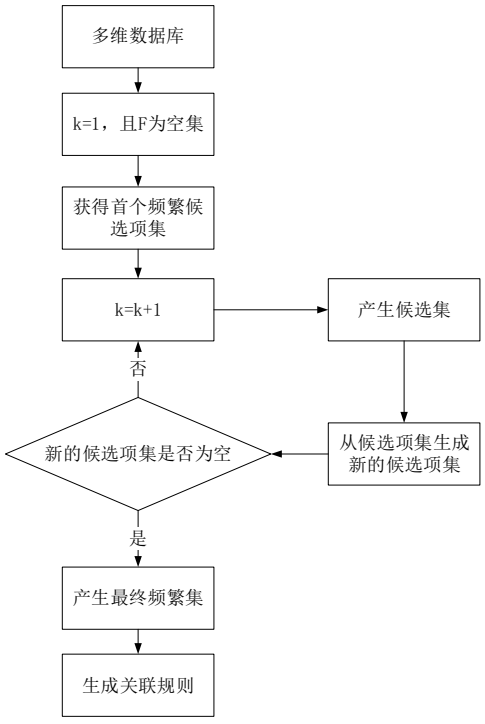

图 2.1 Apriori 算法流程

Apriori 算法通过逐步筛选候选集合取得了很好的效果，虽然可能需要巨大的存储空间来保存它。然而在拥有丰富的频繁模式、或是支持度设置得十分地小，Apriori 算法表现的效果就有些不尽人意了。主要突出的问题有以下两点：

(1) 产生的候选项集过于庞大，十分消耗内存。例如现在手上有 10 的 4 次方个频繁集 1-项，那么这个 Apriori 算法将会产生超过 10 的 7 次方长度的候选项并逐一计算、检测支持度。那么换句话说，为了挖掘 10 的 2 次方个数据项，算法将生成超过 2 的 100 次方，约等于 10 的 30 次方的候选集。并且生成庞大的候选项集这个状况是无法避免的。

(2) 重复多次扫描数据库并逐一比对频繁模式，十分浪费资源。而特别是在挖掘多维频繁模式时，状况更加严重。

2.3 无候选项集的产生方法：FP-Growth 频繁集算法

不同于 Apriori 算法下的“产生再测试”的流程，FP-Growth 采用一种名为 FP-Tree 的紧凑数据结构组织数据[7]，并直接从次树中分析提取频繁项集。避免了 Apriori 类算法的巨大候选集和重复扫描的弊端，FP-Growth 在时间和空间复杂度上有了很大的提升。FP-Growth 主要有以下几点：

(1) 首先采用了一种全新、紧凑的数据结构：频繁模式树(FP-Tree)，一种扩展前缀树来储存生成频繁模式的重要数据信息。每个频繁项路径只有一条，且频繁计数的项越高越接近树根。使加枝的步骤更加地有效率，并能更好地压缩数据。

(2) 其二，一个基于FP-Tree的分段增长挖掘方法仅仅建立一个有条件的频繁模式树，并递归地挖掘这棵树。通过从有条件 FP-Tree 中生成的并联新后缀来增长模式。因为交易集中的频繁项是安排在 FP-Tree 的相应路径，模式增长确保了结果的完整性，不会出现数据关系的遗漏。FP-Growth 不像 Apriori 使用“产生再测试”的流程规定，而只是受制于“测试”的限制。此算法的主要操作是累计计数和前缀路径计数调整，这些操作比生成候选项集和模式匹配比 Apriori 类算法更加地有效率。

(3) 其三，FP-Growth 的挖掘算法是基于划分和分而治之的方法，而不是像 Apriori 类算法由下至上生成频繁项集的结合，这大大地减少了时间和空间复杂度。所以它将原本寻找长长地频繁模式地文题转换成了寻找短频繁模式，并串联上了后缀。这些技术都为大程度地减少搜索花费。

后一章将会介绍 FP-Tree 的结构和架构它的方法，第三章将会介绍基于 FP-Tree 数据结构的关联规则挖掘算法 FP-Growth。

2.3.1 FP-Tree 结构与特点

令 $I=\{i_1,i_2,\cdots,i_k\}$ 为数据项(item)的集合， $D=\{T_1,T_2,\cdots,T_K\}$ 为交易(transaction) T 的集合，其中 $T_i,(i=1,2,\cdots,k)$ 为 item 的集合。根据给定的交易集合 D 和支持度阈值，这个找所有频繁模式的问题称作频繁模式挖掘问题。

FP-Tree，全称为 frequent pattern tree，其数据结构有以下几个定义：

(1) FP-Tree 包括一个标记为“null”的根，并连着一集合的数据 item 的后缀子树作为“null”根的子树和一个频繁项的头表。

(2) 每个后缀子树的节点包含三块内容：项名(item-name)、计数(count)和节点链接(node-link)。记录了代表的项的名称；计数代表了这个节点的路径所代表的事务处理的数目；节点链接指向树的下一个节点，若已为最后一个节点则节点链接指向空。因为不同的事务会有若干个相同的项，路径会有重叠。显而易见路径重叠的越多，数据压缩的效果越好。

(3) 频繁项集头表的每条数据包括两块内容：项名(item-name)和节点连接(node-link)的起点。该起点指向 FP-Tree 中的第一个带项名的节点。

基于以上这三个定义，可以进一步讨论 FP-Tree 的架构算法。

2.3.2 FP-Tree 架构算法

算法 2.4：

输入：交易集合 D，最小支持度阈值

输出：频繁模式树 FP-Tree

方法：

(1) 扫描一次交易集合 D，收集频繁数据项集合 F 和对应的支持计数。并将 F 按支持计数降序排序成新的集合 L。

(2) 新建了 FP-Tree 的根 T，标签记为“null”。遍历每个 $T_i,(i=1,2,\cdots,k)$，循环做以下几步操作。在交易集内的数据项筛选并重新排序成新的序列 L。记拥有排序后数据项的交易集为 $[p|P]$，其中 p 是第一个元素，P 是剩下的元素集合。调用函数 $insert\_tree([p|P],T)$。

函数 $insert\_tree([p|P],T)$ 有如下的定义。如果 T 有一个孩子为 N，并满足 $N.item-name=p.item-name$ 那么 N 的计数增量为 1；否则就新建一个节点 N，设其计数为 1，并将新建的节点 N 向上父链接连到 T，其节点链接则通过节点链接结构连到具有同样项名的节点。如果 P 不为空，则递归地调用函数 $insert\_tree(P,N)$。

例，如下表的交易集合表 2.1 与相对应的 FP-Tree 的架构过程图 2.2：

表 2.1 FP-Tree 示例交易项集

TID	交易项
TID001	a, b, e
TID002	b, d
TID003	b, c
TID004	a, b, d
TID005	a, c
TID006	b, c
TID007	a, c
TID008	a, b, c, e
TID009	a, b, c
TID010	a, b, c
TID011	a, b, d
TID012	a, c, d, e
TID013	b, d

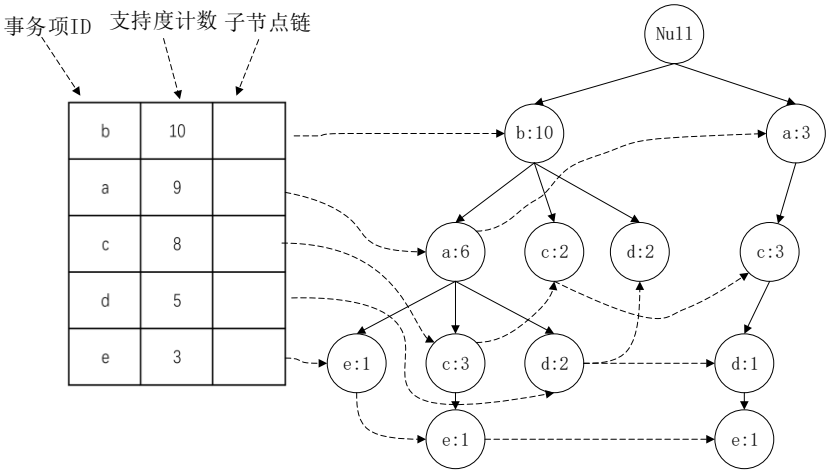

图 2.2 FP-Tree 架构过程示例图

通过架构 FP-Tree 数据结构地过程可知，只需要扫描两次交易集合数据库。第一次是为了收集频繁项集，第二次是为了架构 FP-Tree。将某个交易集合插入到树中，需要的时间复杂度为$O(|Trans|)$，其中$|Trans|$是该交易集合中数据项个数。

2.3.3 FP-Growth 挖掘算法基本属性

FP-Tree 的压缩结构为后续的挖掘挖掘步骤提供了很好的保障。然而它依旧不能自动地保证后续操作有十分高的效率。因为如果直接将 FP-Tree 拿来挖掘并将每个候选模式都一一读取检验的话，还是会遇到生成候选集并合并它们的问题，和 Apriori 就没有区别了。在本章节将介绍 FP-Growth 的算法原理，包括如何调取压缩数据结构的数据和生成一种高效的挖掘方法来挖掘整个频繁模式集合。

属性 2.1（节点链接属性）：任意数据项$i_n,(n=1,2,\cdots,k)$，所有包括i_n的频繁模式都可以通过i_n的节点链接被获取，并从是 FP-Tree 的头表中以i_n为第一个的路径开始。

这个属性主要是基于 FP-Tree 的架构过程。其多应用于通过 FP-Tree 中i_n的节点链接，遍历所有和i_n相关的模式信息。

属性 2.2（后缀路径属性）：为了计算在路径P中的某个节点i_n的频繁模式，只需要将路径P下的该节点i_n后缀子路径进行计算，并将后缀路径中每个节点的频繁计数更新至与节点频繁计数一致。

引理 2.1（分段式增长）：使α作为数据库中的一个数据项，B是数据项α的条件模式基，并且β是B中的另外一个数据项。则$\alpha \cup \beta$在数据库中的支持度计数等于β在数据库中的支持度计数。

从该引理，可以得出以下一个重要的推论。

推论 2.1（模式增长）：令α为数据库中的一个频繁数据项，B是数据项α的条件模式基，并设β为B中的另外一个数据项。那么，根据数据库中的数据，$\alpha \cup \beta$是频繁的。当且仅当，β在B中是频繁的。

引理 2.2（单条 FP-Tree 路径模式生成）：假设 FP-Tree$T$中有一路径P。T的全部频繁模式可以通过列举路径P下的所有子路径集合以及各个子路径对应的支持度计数作为该子路径下的各个数据项的最小支持度计数来生成。

基于以上的属性和引理，称得出的使用 FP-Tree 的频繁模式挖掘为 FP-Growth 算法。

2.3.4 FP-Growth 挖掘算法伪代码

定义 FP-Growth 函数的伪代码[8]如下。

输入：根据算法 3.4 生成的 FP-Tree，以及最小支持度阈值

输出：全部的频繁模式

方法：调用函数$FP-Growth(FP-Tree,\alpha)$

算法 2.5　$FP-Growth(Tree,\alpha)$

```
1:  if Tree 包括一条单路径 P then
2:    for 路径 P 中每个节点组合 do    {节点组合记为 β}
3:      生成模式 β∪α，并 support_count(β∪α) = min_support(β中的节点)
4:    end for
5:  else
6:    for 每个树的最底层中的数据项 i_n do
7:      生成模式 β = i_n ∪ α，并 support_count(i_n ∪ α) = support(i_n)
8:      生成 β 的条件模式基
9:      生成 β 的条件 FP-Tree: Tree_β
10:     if Tree_β ≠ 0 then
11:       call FP-Growth(Tree_β, β)
12:     end if
13:   end for
14:  end if
```

FP-Growth 算法流程图如图 2.3 所示：

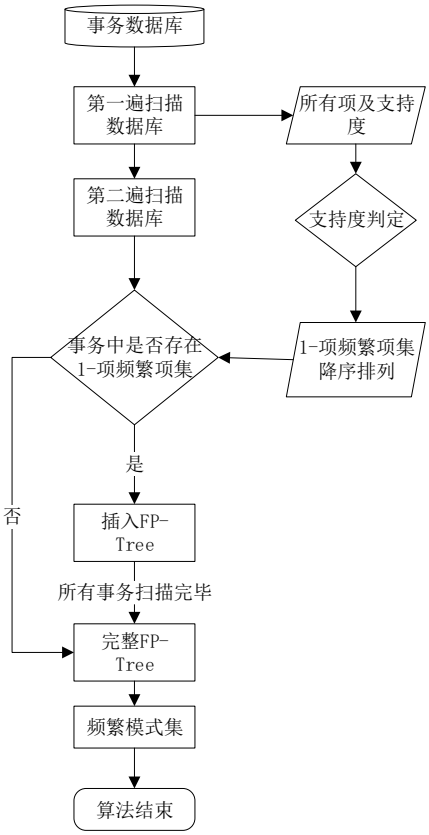

图 2.3 FP-Growth 算法流程图

FP-Growth 挖掘过程仅扫描 FP-Tree 一次，即可对应每个不同的频繁数据项 i_n 生成一个小型的模式基 B_{i_n}，且每个都包含 i_n 的后缀路径。频繁模式的挖掘即是接下来递归地在小型模式基 B_{i_n} 中为 B_{i_n} 建立一个条件 FP-Tree。通过算法 3.4 可知，FP-Tree 一般是比数据库的尺寸要小的。相似地，条件 FP-Tree“FP-Tree$|i_n$”是根据模式基 B_{i_n} 架构的，其规模尺寸应该小于或等于 B_{i_n} 的尺寸。并且，模式基 B_{i_n} 通常比其原始 FP-Tree 小很多，因为它仅包括一个数据项 i_n 相连的转换后的后缀路径。因此，每个子序列的挖掘过程是应用于一个尺寸通常更小的模式基和条件 FP-Tree 的集合。挖掘操作中包括绝大部分的后缀计数调整，计数和模式分段式串联。这比生成并测试庞大数量的候选模式要来得快。所以此算法是更加有效率的。

同时，从算法的思想和操作流程可见，FP-Growth 挖掘过程是一个“分而治之(divide-and-conquer)”的过程[9]，且扫描次数的需求通常会骤减。如果从数据库建立一颗 FP-Tree 的缩减因子是 20 到 100 之间，那么对应地从每个已经十分小的条件频繁模式基中架构相应的条件 FP-Tree 的过程会有上百次的操作减少。

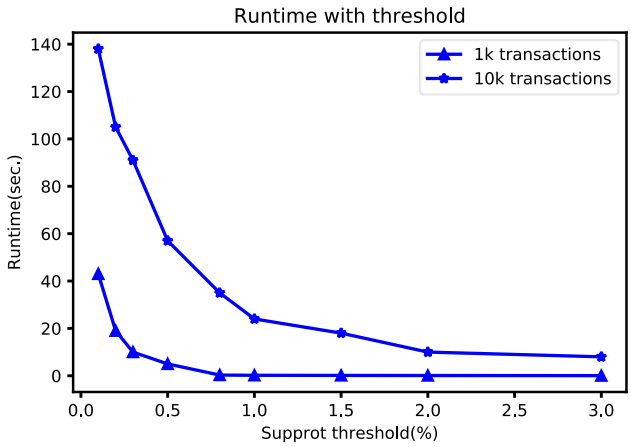

图 2.4 FP-Growth 支持度与运行时间走势图

如图 2.4，其分别显示了拥有一千和一万的交易集数量在不同的支持度阈值设置下，FP-Growth 算法所需要运行的时间。

值得注意的是，甚至一个数据库中可能会生成指数级数量的频繁模式，FP-Tree 的尺寸通常还是很小，且它的尺寸绝不会按照指数级增长。

2.3.5 FP-Growth 时间复杂度分析

假如某条件模式基含有 n 项，则按照单个条件模式基的长度递增排序 $T_1(n)=\theta(n*log(n))$。此时总匹配次数是：$n-1+n-2+...+1=n*(n-1)/2=\theta(n^2-n)=\theta(n^2)$，则总的时间复杂度为：$T(n)=n*log(n)+n*(n-1)/2=\theta(n^2+n*log(n)-n)$。当 n 大于时 4 时，该值小于 n^2。在实际中 n 一般会大于 4。即 $T(n)=\theta(n^2)$

2.4 两种算法优劣比较

2.4.1 FT-Tree 的数据保存完整性

经过证明，Apriori 类的算法共同拥有一个瓶颈就是会生成大量的候选集，并逐一检测。但是 FP-Tree 提供了一种压缩数据库的方法，唯一需要注意的就是 FP-Tree 是否完整地保存了数据，并且保证数据的压缩率。再建立 FP-Tree 的时候，可以总结出以下几点引理。

引理 2.3　给定一个交易集和支持度阈值，它对应的 FP-Tree 包括了关于频繁模式挖掘所需要的所有信息。

基于 FP-Tree 的架构过程每一个交易事务项都画成 FP-Tree 中的一条路径，并且每个交易集中的频繁数据项的信息都完整地储存在 FP-Tree 里面。同时，FP-Tree 里面的一条路径可能代表着不同交易集中的频繁数据项，但是并不会产生歧义，因为这条路径每个交易集合必须从每个数据项的后缀子树的根开始。

引理 2.4　不需要考虑根节点，FP-Tree 宽度的大小和数据库中所有相关的频繁数据项息息相关，而其深度的大小与所有交易集合中的频繁项的最大数量息息相关。

基于 FP-Tree 的架构过程，对于数据库中的每一个交易集合在 FP-Tree 中一条路径是对应频繁项的后缀字数，所以能确保路径上的节点完完全全就是交易集中的频繁项。由于任何交易集合中不存在一个频繁项能够在树中创建超过一个节点，FP-Tree 的树根是唯一一个不由插入频繁项而产生的，并且每个节点包含一个节点链接和一个计数信息。树的尺寸在上一个引理已经介绍过。任意节点的带此节点开头的频繁数据项集列表的后缀子树的高度是任意交易集的频繁项的最大个数数量，因此这棵树的高度和数据库中任意交易集的频繁项的最大个数数量息息相关，前提是如果不考虑树根带来的其他层数。

引理 2.5　其实展现了 FP-Tree 的重要的优点：FP-Tree 的尺寸和它的来源数据库的尺寸息息相关，因为每个交易集会最多为 FP-Tree 贡献一条路径，路径的长度和交易集中的频繁项的个数相同。由于各个交易集之间经常会有重复的频繁项，所以树的尺寸一般比原始数据库的尺寸要小。不像 Apriori 类算法会生成呈指数级别增长的候选集个数是最坏的情况，没有任何情况下 FP-Tree 会生成呈指数型增长的节点个数。

FP-Tree 是一种高度压缩的数据结构，用于储存频繁模式挖掘所需的所有信息，由于某个单路径为“$i_1 \to i_2 \to \cdots \to i_n$”在 i_1 的后缀子树中，频繁项集的最大个数的情况就是在“$i_1 \to i_2 \to \cdots \to i_n$”中。并且候选集是边生成边挖掘的。

频繁项集中的项是按照降序排序的，大多数频繁次数多的是更加接近 FP-Tree 的根并且更有可能被共享。这表明 FP-Tree 的结构通常十分地紧凑，一颗很小地 FP-Tree 是通过压缩一些十分大型地数据库。例如，在 MaxMiner[10] 中用过地数据库 Connet-4 拥有 67,557 个交易集，每个交易集有 43 个频繁项。当支持度阈值设定为 50%，总生成地频繁项集个数为 2,219,609，而 FP-Tree 的总节点数为 13,449。这表明尽管有成百上千的频繁模式组合，但是此 FP-Tree 的压缩缩减率依旧达到了 165.04。当然，如果交易集过短的话，压缩率就不一定那么表现优秀了。由此得出结论，虽然 FP-Tree 比 Apriori 类算法表现优秀，但是它不是适用于任意条件的数据库。

2.4.2 Apriori 算法和 FP-Growth 算法

FP-Growth 算法比 Apriori 算法的可扩展性要强的多。这是因为随着支持度阈值的下降，频繁项集的总个数同时也是总长度激增。Apriori 生成并测试的候选项集会变得十分的庞大，并且需要搜索并处理大量的模式配对，通常这个过程十分地耗时耗存储空间。

其次，随着支持度阈值地增大，每个项集的平均运行时间也随之增长。虽然频繁项集的数量呈指数性增长，但是 FP-Growth 算法的每个项集的平均运行时间的增长幅度十分地保守。这就是为什么 FP-Growth 算法在有支持度阈值地情况下，也可以有很强地扩展性。

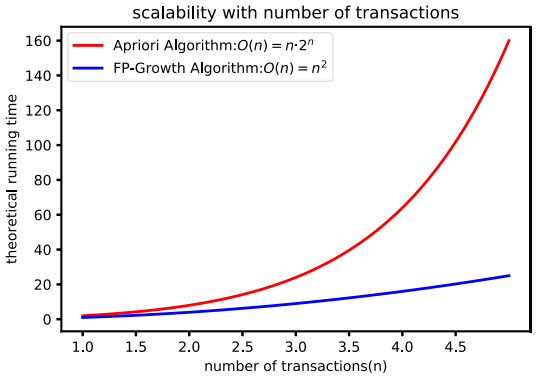

图 2.5 交易集合数与理论运行时间走势图

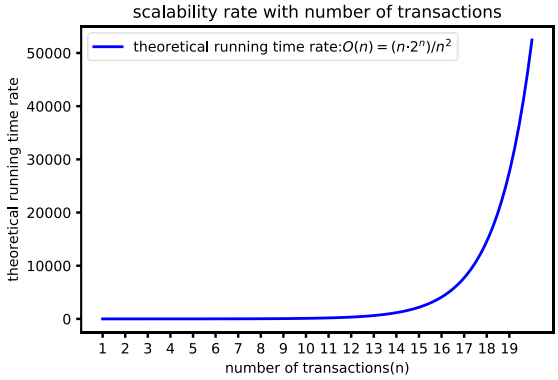

图 2.6 交易集合与理论运行时间比率走势图

两个 FP-Growth 和 Apriori 表现出线性的扩展性，随着交易集合的数量的增长，FP-Growth 比 Apriori 有更优秀的延展性，并且 FP-Growth 的延展性值与 Apriori 的延展性值之间的差距越来越大。总的来说，FP-Growth 比 Apriori 快上一个数量级，且当最小的支持度阈值缩减时，这个差距将会越来越宽。

3 改进算法

港口仿真数据具有以下几个特点：维数多，复杂度高以及可分为“输入与输出”两大数据类。针对最后一项特点，本算法在数据处理的过程中使用了数值标记法，可以在挖掘的关联规则结果中有明确的对应的维。本章将介绍基于 FP-Growth 算法，针对港口仿真数据的特点设计的改进算法 Upgrade-FP-Growth 算法，包括伪代码、算法流程图以及时间复杂度分析。

3.1 数值标记法

由于将输入项与输出项分别归为一起，调用 算法会导致生成关联规则时，会出现数据混淆。不能够十分清楚地区分生成地数据是在哪一个数据项集中的，但是又不需要附加一个庞大的计算程序，因为这会很大程度上地增长计算时间。同时也不能为每个数据增加一个属性项，存储量会按数据项集的数量成比例增长，大幅度影响了空间复杂度。于是决定采用最简单的标记法。将输出项集的各个数据项按顺序分别乘以 10 的倍数，倍数依次增加，以示区分。例在输出项中，qc 数量，agv 数量，armg 数量和任务数量等。

这样在挖掘结束后，只需要根据数据的次方数来区分数据项。不仅减少了数据处理和整理的时间，而且有效地减少了程序的长度和时间复杂度以及空间复杂度。对于不同分类的多维数据架构采用先“输出数据项列表挖掘，再输入数据项挖掘”索引的架构，先对输入项的数据进行挖掘，并在此基础上，循环地对每个关联规则所对应的输入项进行挖掘过滤。

3.2 Upgrade-FP-Growth 算法

通过上一章节的分析，可以得出 FP-Growth 算法在性能上比 Apriori 算法表现更优。所以改进的算法，是部分采用了 FP-Growth 算法。改进后的 Upgrade-FP-Growth 算法伪代码如下表所示：

算法 3.1 *Upgrade – FP – Growth*

1: **import** 所有所需 python 数据包

2: **for** excel 的每行数据 **do**

3: 每个数据项的 list 导入该行对应的数据
4: **end**
5: **for** 输入类数据的 list 中的每条数据项 **do**
6: **if** 该条输入类数据不在 $trans\_combine$ 里面 **then**
7: 将该条输入类的数据导入 $trans\_combine$ 里面
8: 将其对应的输出类数据导入到 $trans\_indi$ 里面
9: **else**
10: 遍历 $trans\_combine$，找到该条数据对应的位置
11: 将其对应的输出类数据导入到 $trans\_indi$ 中，遍历得出的下标 list 里面
12: **end if**
13: **end**
14: **for** $trans\_indi$ 中的每一个下标 **do**
15: **for** $trans\_indi$ 该下标对应 list 的每一个下标 **do**
16: 将 $trans\_indi$ 该下标对应 list 的对应下标数据项进行数值标记法处理
17: **end**
18: **end**
19: $call\ \ FP-Growth$
20: **for** 第一次挖掘得出的 rules 字典的每一个 key **do**
21: **if** 该条 rules 只有 key，没有对应的内容项 **then**
22: 筛选无用的 rules
23: **else**
24: **for** 每个 key **do**
25: 导入 key 的值到 list_i 中
26: **end**
27: **for** 交易集合的每条数据 **do**
28: 判断是否是 trans_output 中数据的子集
29: **if** 不是子集 **then**
30: 导入数据到 trans_output 里面
31: **end if**
32: **end**
33: **end if**
34: $call\ \ FP-Growth$

35: 遍历第二次挖掘的 rules

36: 筛选无用的信息

37: **print** 最终的关联规则

38: **end**

在此算法中，首先是将后续程序所要用到的数据包导入到程序中，同时也包括 FP-Growth 算法的定义类和函数的 python 文件。接着就是读取筛选后的 excel 文件，将其数据一条条地读取，赋值进程序新定义的列表里面。接下来就是将读取的数据维列表按照“输入和输出”两个数据类进行合并。因为基于港口仿真数据的特殊性，“输入”数据类的维中数据项多有重复，所以在合并之前先遍历之前已赋值的“输入项”列表，判断是否此输入项在此之前已赋值过相同的数据。如果赋值过，则返回之前赋值的位置下标，将此条数据的“输出项”赋值到对应的“输出项”列表位置中；如果没有赋值过，则在“输入”和“输出”两个列表中分别在最后赋值上此条数据。这样有效地减少数据冗余的情况，节省了存储空间的资源。

当数据存储的步骤完成之后，接下来遍历先前的“输入项”和“输出项”列表，遍历列表中的每一个内列表，对每个内列表中的每一个数据项进行数据处理和数值标记。具体此状况下，数值标记法该如何运用，是要基于人工对于整理数据情况的把控。对不同的数据质量和数据之间的差异做出具有针对性的数值标记方案。通常在同等级数的情况下，每个数据项依次乘以 10 的次方的倍数。以此以示区分。

数值标记之后，接下来就是第一次调用 FP-Growth 算法，进行第一次的挖掘。每组一条“输入项”对应一条交易集合数据，并且每条“输出项”交易集合所对应的“输入项”的各个数据维的数不完全相同。待第一次挖掘的关联规则输出之后，对每条关联规则，在最初的“输入输出”数据列表遍历，找出所有对应该关联规则的数据条。并汇总二次调用 FP-Growth 算法。

对于二次挖掘算出的关联规则进行筛选，删除无用的规则，同时合并部分有交集的关联规则。整理好数据后，程序输出最终的关联规则。程序结束。

此算法的 python 程序可见附件。

且 Upgrade-FP-Growth 算法的流程图如图 3.1 所示：

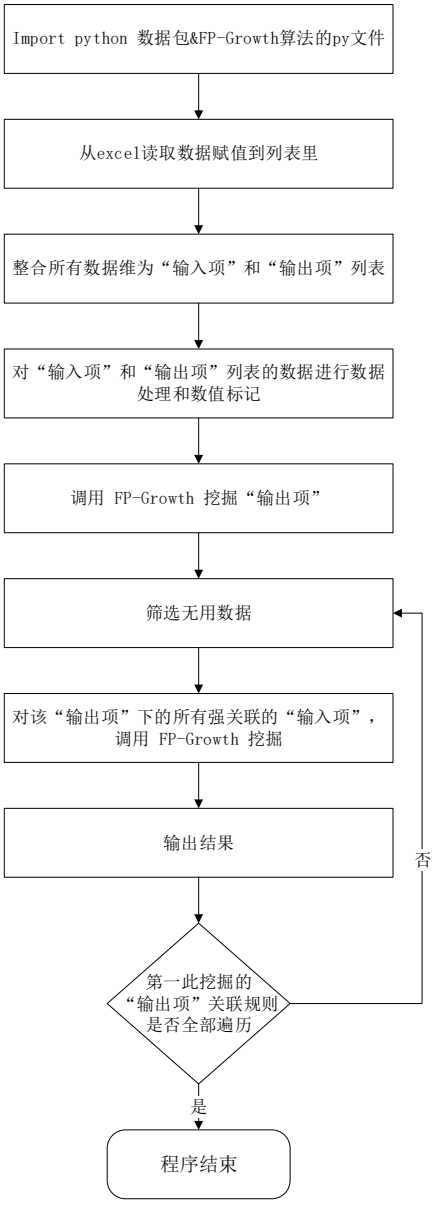

图 3.1 Upgrade-FP-Growth 算法流程图

算法分析：

假设经过数据预处理之后，Upgrade-FP-Growth 需要处理 n 条的数据交易集。由此前的 FP-Growth 算法的时间复杂度 $T(n)=\theta(n^2)$。不妨假设第一次调用 FP-Growth 时生成的关联规则集合，经过了合并处理，一条交易集中最多包含一条关联规则的数据。那么，能够得到 $\mathrm{O}\left(n^2\right)$ 个关联规则，而每个关联规则需要再次合并。因为一个关联规则会出现其对应的输入数据项是有重复出现的现象。那么由此可得后期调用 FP-Growth 将需要 $\mathrm{O}\left(\left(n^2\right)^2\right)=\mathrm{O}\left(n^4\right)$ 的时间复杂度。所以汇总起来 Upgrade-FP-Growth 的时间复杂度为 $T(n)=\theta(n^2)+\mathrm{O}\left(n^4\right)=\mathrm{O}\left(n^4\right)$。[11] 当 n 小于 17 时， *Apriori* 算法是较 Upgrade-FP-Growth 更加快的。但是一般交易集合的个数是远远大于 17，所以长远来看，Upgrade-FP-

Growth 较 Apriori 更加快，且其时间复杂度可以维持在$O(n^4)$，而 Apriori 是以指数倍来增长的。

虽然 Upgrade-FP-Growth 的时间复杂度是依靠幂函数的增长幅度来增长的。并不难看出，为了减小其时间复杂度，最主要的方向就是尽量减少数据的维度。而这部分操作的实现主要是要依赖数据的预处理。

4 算例分析

4.1 算例流程设计

由于关联规则挖掘所需的港口仿真数据的数据量过于庞大，不便以图表的形式展示，所以本章选取了一小部分数据作为样板来进行算例分析。主要流程步骤有以下几个阶段：

（1） 问题定义与数据提取

根据问题的定义锁定目标的范围与目的，从而选取对应的数据段，进行数据收集。

（2） 数值标记法

将收集到的数据进行预处理，利用上一章介绍的数值标记法进行二次加工处理。

（3） Upgrade-FP-Growth 算法挖掘

对处理后的数据进行数据转换，多次调整支持度和置信度参数，并利用上一章介绍的 Upgrade-FP-Growth 算法进行关联规则的挖掘。

（4） 分析结果汇总

根据挖掘出的关联规则进行分析，并以文字的形式做出汇总。

整个算例的流程设计如图 4.1 所示：

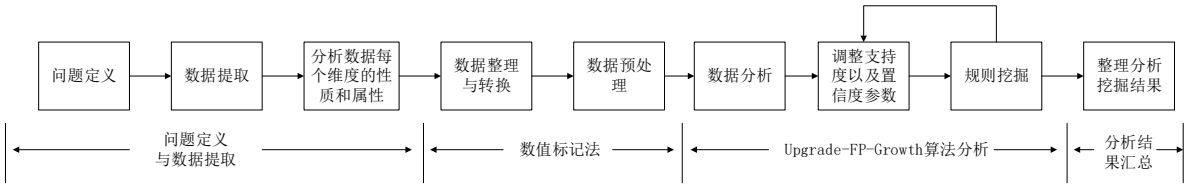

图 4.1 算例流程设计图

4.2 问题定义与数据提取

4.2.1 问题定义

为了在复杂度高的情况下，对数据有初步的分析和了解，最终得到有利于港口仿真系统改进的依据。

4.2.2 目标与范围

目标：算例分析的样张选择根据对仿真输入与输出两大数据类进行挖掘，为港口仿真系统的该进提供依据与建议。

范围：针对港口仿真系统中的相关信息进行数据提取。

4.2.3 数据提取

由于算例分析的样张只起到演示作用，所以根据以上给定的范围在港口仿真数据中进行信息的筛选,选取其中 142 条数据。每条数据包含 8 个重要属性，其中 5 条为输入类数据：agv 数量、armg 数量、任务数量、指派策略选择、key，另外三条为输出类数据：timecost、平均 AGV 到达待选任务接箱点的空驶距离、平均 AGV 指派待选任务所需的重驶距离。如表 4.1 所示。

表 4.1 初始数据表

agv 数量	armg 数量	任务数量	指派策略选择	key	平均 AGV 到达待选任务接箱点的空驶距离
2	3	10	0	0	0.471014493
2	3	10	0	1	0.81884058
2	3	25	0	0	0.471014493
2	3	25	0	1	0.857487923
2	3	30	0	0	0.81884058
2	3	30	0	1	0.81884058
2	3	30	0	0	0.471014493
2	3	30	0	1	0.548309179
2	3	50	0	0	0.471014493
2	3	50	0	1	0.857487923
2	3	50	0	0	0.096618357
2	3	50	0	1	0.548309179
2	3	50	0	0	0.096618357
2	3	50	0	1	0.548309179
2	3	50	1	0	0.471014493
2	3	50	1	1	0.096618357
2	3	40	1	0	0.096618357
2	3	40	1	1	0.548309179
2	3	30	1	0	0.096618357
……	……	……	……	……	……
5	4	30	9	1	0.548309179

其中每一项的数据项的定义与数据类型以及指派策略数据项的每个编号对应的指派内容，如表 4.2 与表 4.3 所示。

表 4.2 数据项类型及含义表

数据项集	数据类型	代表意义
agv 数量	文本	此次仿真运作的自动导引小车(agv)数量
armg 数量	文本	此次仿真运作的场桥(armg)数量
任务数量	文本	此次仿真总箱数
指派策略选择	文本	9 大指派策略的采用
key	文本	指定初始作业的岸桥号
timecost	文本	此次仿真运作的总时间
平均 AGV 到达待选任务接箱点的空驶距离	文本	此次仿真运作的九大指标数值之一：平均 AGV 到达待选任务接箱点的空驶距离
平均 AGV 指派待选任务所需的重驶距离	文本	此次仿真运作的九大指标数值之一：平均 AGV 指派待选任务所需的重驶距离

表 4.3 指派策略编号表

编号	指派策略
0	紧急任务指派(due time)
1	相对到达时间差大优先指派(arrival time difference of AGV)
2	最早交箱时刻指派(crane handover time)
3	当前 AGV 到达待选任务接箱点的空驶距离优先指派(empty travel distance)
4	当前 AGV 所选待选任务的重驶距离优先指派(loaded travel distance)
5	岸桥平均延迟大优先指派(average delay of qc)
6	装船箱优先指派(handling type)
7	所在 ARMG 的相对剩余工作量指派(relative remaining workload of armg)
8	最好在一个箱区装卸，双循环的可能性指派(dual cycle likelihood of agv)
9	随机指派(random dispatching)

4.3 数值标记法

4.3.1 数据整理与转换

从采用的仿真系统导出的数据文件中可见数据的属性为字段，无法在后续的程序运行中直接采用处理。所以需要先将数据的属性改为数值。并再剔除数据表中的一些明显有问题的数据，例如，在“平均 AGV 到达待选任务接箱点的空驶距离”数据维中，存在一些数值为 0 的情况。

4.3.2 数据预处理

根据上一章节介绍的数值标记法，将数据的维分为输入维和输出维两大类。分别对于输入维和输出维中的维度进行数值标记。在输入项中，由于有两个维同时出现有数值“0”的存在，所以先将“key”维全部加一并乘以 10 的 2 次方处理；“指派策略选择”和“任务数量”的数值维持不动；由于“agv 数量”和“armg 数量”在此数据库中都为单位数，所以就简单地“agv 数量”乘以 10 的 3 次方，“armg 数量”乘以 10 的 4 次方。在输出项中，“平均 AGV 到达待选任务接箱点的空驶距离”乘以 10 的 1 次方再取整。

处理过后的数据效果如表 4.4 所示：

表 4.4 数据项表（数据处理后）

agv 数量	armg 数量	task 数量	指派策略	key	指标一
2000	30000	10	0	100	[5]
2000	30000	10	0	200	[8]
2000	30000	25	0	100	[5]
2000	30000	25	0	200	[9]
2000	30000	30	0	100	[8, 5]
2000	30000	30	0	200	[8, 5]
2000	30000	50	0	100	[5, 1, 1]
2000	30000	50	0	200	[9, 5, 5]
2000	30000	50	1	100	[5]
2000	30000	50	1	200	[1]
2000	30000	40	1	100	[1]
2000	30000	40	1	200	[5]
2000	30000	30	1	100	[1]
2000	30000	30	1	200	[8]
2000	30000	25	1	100	[5, 8]
2000	30000	25	1	200	[5, 1]
5000	20000	10	1	100	[9]
5000	20000	10	1	200	[5]
5000	30000	10	1	100	[9]
5000	30000	10	1	200	[9]
2000	30000	10	2	100	[1, 9]
……	……	……	……	……	……
2000	30000	25	2	100	[1]

4.4 Upgrade-FP-Growth 算法挖掘

4.4.1 技术简介

操作系统：Windows 10

开发环境：Python 3.7.1

开发工具：Spyder

编程语言：Python

4.4.2 关联规则挖掘

如上文中提到的，将 excel 中的数据等信息导入到程序中后，遍历每一个数据，将其改为整数的形式。此算例的 Upgrade-FP-Growth 算法的程序可见附件。

初次实验，为了大致了解此段数据的情况。设定第一次挖掘的参数设定为：支持度为 0.5%，置信度为 0.5%；第二次挖掘的参数设定为：支持度 1%，置信度为 0.5%时。第一次关联规则得出了 2 条关联规则。第二次挖掘时，两条关联规则所得出的二次关联规则数量分别为 30 条与 145 条。

为了找出较为适合的置信度和指出都参数，经过多次实验，记录实验结果，如表 4.5 所示：

表 4.5 算例调参表

第一次挖掘		第二次挖掘		第一次关联规则数量	第二次关联规则数量	
支持度(%)	置信度(%)	支持度(%)	置信度(%)			
0.5	0.5	1	0.5	2	30	145
0.5	0.5	1	1	2	30	136
0.5	0.5	1	1.5	2	0	0
0.5	0.5	1.5	1	2	0	86
1	0.5	1.5	1	2	0	86
1	1.5	1.5	1	0	0	0
1.5	1	1.5	1	0	0	0
1	1	3	1	2	0	45
1	1	6	1	2	0	25
1	1	20	1	2	0	2

由表格可见，经过多次调参。通过逐个控制变量的方法，确定四个参数的最大极限值。第三、六、七条数据分别确定了第二次挖掘的置信度最大参数为 1%，第一次的置信度和支持度参数也为 1%。第二次挖掘的最大支持度参数也确定为 20%。

以第二次挖掘的支持度参数为例。

当第一次挖掘的参数设定为：支持度为 1%，置信度为 1%；第二次挖掘的参数设定为：支持度 6%，置信度为 1%时，以下为程序输出结果：

一次关联规则分析: {(0,): ((1,), 1.0), (1, 1): ((5,), 1.0)}

二次关联规则分析: {}

二次关联规则分析: {(50,): ((2000, 30000), 1.0), (50, 2000): ((30000,), 1.0), (50, 30000): ((2000,), 1.0), (50, 200): ((1, 2000, 30000), 1.0), (50, 200, 2000): ((1, 30000), 1.0), (50, 200, 30000): ((1, 2000), 1.0), (1,): ((200, 2000, 30000), 1.0), (1, 50): ((200, 2000, 30000), 1.0), (1, 200): ((2000, 30000), 1.0), (1, 30000): ((200, 2000), 1.0), (1, 2000): ((200, 30000), 1.0), (1, 50, 200): ((2000, 30000), 1.0), (1, 50, 30000): ((200, 2000), 1.0), (1, 50, 2000): ((200, 30000), 1.0), (1, 200, 2000): ((30000,), 1.0), (1, 200, 30000): ((2000,), 1.0), (1, 2000, 30000): ((200,), 1.0), (1, 50, 200, 2000): ((30000,), 1.0), (1, 50, 200, 30000): ((2000,), 1.0), (1, 50, 2000, 30000): ((200,), 1.0), (50, 200, 2000, 30000): ((1,), 1.0), (200, 2000): ((30000,), 1.0), (200, 30000): ((2000,), 1.0), (2000,): ((30000,), 1.0), (30000,): ((2000,), 1.0)}

这里不妨将四个参数调到最大。第一次挖掘的参数设定为：支持度为 1%，置信度为 1%；第二次挖掘的参数设定为：支持度 20%，置信度为 1%时，以下为程序输出：

一次关联规则分析: {(0,): ((1,), 1.0), (1, 1): ((5,), 1.0)}

二次关联规则分析: {}

二次关联规则分析: {(2000,): ((30000,), 1.0), (30000,): ((2000,), 1.0)}

由此结果可得，在此数据片段中，可以得出输出数据 1, 5 与输入数据 2000, 30000 有强关联的结论。

4.5 分析结果汇总

由上一小章的挖掘可得出以下结果：

输出数据 1, 1, 5 分别与输入数据 1, 50, 200, 2000, 30000 有强关联。根据前期采用数值标记法的数据预处理过程，可知在此提取的数据库中，当输入数据中指派策略选择 1，task 数量为 50，key 为 1，agv 数量为 2，armg 数量为 3 的情况下，输出维数据：平均 AGV 到达待选任务接箱点的空驶距离出现 0.1 或 0.5 的情况较多。

虽然对小型数据量进行挖掘，能够对 Upgrade-FP-Growth 算法有较清晰的认识，但这也导致了数据结果的局限性。例如在采用的数据库中 agv 数量和 armg 数量分别有较多的“2”和“3”的数据，所以导致挖掘结果朝着 agv 取“2”和 armg 取“3”倾斜。但是同时从此算例分析的挖掘结果可看出，对于“平均 AGV 到达待选任务接箱点的空驶距离”

这一输出项，“指派策略选择”的 10 项指派策略分别产生了不同的效果。这也说明在此输出项下，港口仿真系统设置这 10 个指派策略并没有冗余。

在四个参数都调整到最大时，只得到一条最强的关联规则：输出数据 1, 5 与输入数据 2000, 30000 有强关联的结论。这条关联规则的含义是在港口仿真系统中，如果设定 agv 数量为 2、armg 为 3 的情况下，很有可能得到 0.1 或者 0.5 左右的“平均 AGV 到达待选任务接箱点的空驶距离”数值，并且出现的比例为 2：1。由于选取的数据片段中有一定部分的 agv 和 armg 数量数值为“2”和“3”。所以，不能得出此数值组合对其他平行的数值之间的关联。但是能很明显地看出，agv 数量和 armg 数量是对数据项“平均 AGV 到达待选任务接箱点的空驶距离”有一定数值上的影响的。

经过上面的分析，可以清楚地认识到关联规则虽然对于港口仿真不能像调度研究或者其他优化数学模型直接给出优化的方案，但是对于港口仿真系统的设计自查和修改提供了理论依据和改进方向。

5 总结

5.1 研究总结

就目前而言，计算机领域数据关联规则的算法已逐渐趋于成熟，但是由于港口仿真数据的特殊特征，如多层、多维、数据量巨大等。本文以多维港口仿真数据作为研究对象，分析了两种不同的挖掘关联规则分析的算法思路，并进行了优劣比较。在 FP-Growth 算法的基础上，进行改进成了 Upgrade-FP-Growth 多维关联规则分析算法，并将此改进算法运用在多维港口仿真数据上。综上所述，本文主要涉及以下几个方面：

(1) 分析了港口以及港口数据发展的现状，引出未来数据挖掘在拥有海量数据的领域应用的重要性。

(2) 对两种关联规则 Apriori 算法与 FP-Growth 算法进行了理论介绍与分析，并进行比对，阐述其优劣点。

(3) 主要介绍了基于 FP-Growth 算法的改进算法：Upgrade-FP-Growth 算法，包括其架构、伪代码、时间复杂度分析以及其采用的数值标记法。

(4) 将 Upgrade-FP-Growth 算法运用在多维港口数据分析中，选取一部分数据为代表，对其进行算例演示与分析。对于港口仿真系统的设计自查和修改提供了理论依据和改进方向。

5.2 研究展望

然而，显然此算法依旧有些许的不成熟之处。首先，Upgrade-FP-Growth 算法时间复杂度的最大难题还是数据的维数控制，并且维数的数值还是处在指数的位置上，由此导致 Upgrade-FP-Growth 算法在时间复杂度上地表现不及预期的优秀；其次，数据项的数值区分还有改进的空间。本文采用了最简单的数值标记法，对后期的数据处理都一定程度上地减轻了负担，但是在前期需要均衡分析每个维度的数值情况做出相应的处理方案。并且维度越大，需要的人工精力就越多。如果维数达到一定量大地时候，程序需要花一定量地时间来分析各数值是 10 的几次方。所以数值标记法只适用于维度并不是十分庞大的情况。虽然 FP-Tree 在压缩数据上十分有优势，但是维度过于庞大的话，其的表现会大打折扣。所以压缩输入的数据项维数是一个很需要改进、同时也很有改进前景的方向。

参考文献

[1] Agrawal R.Imieliński T.Swami, A. Mining Association Rules Between Sets of Items in large Databases: Proceedings of ACM SIGMOD Conference on Management of Data, 1993:207-216.

[2]蔡伟杰,张晓辉,朱建秋,朱扬勇.关联规则挖掘综述[J].计算机工程,2001(05):31-33+49.

[3]王新宇,杜孝平,谢昆青.FP-growth 算法的实现方法研究[J].计算机工程与应用,2004(09):174-176.

[4]Pang-Ning Tan, Michael Steinbach, Vipin Kumar. Introduction to Data Mining. 范明,范宏建,译. 2 版. 北京:人民邮电出版社, 2011：202-205.

[5]陆楠,王喆,周春光.基于 FP-tree 频集模式的 FP-Growth 算法对关联规则挖掘的影响[J].吉林大学学报(理学版),2003(02):180-185.

[6]单明辉. 改进的关联规则算法在采购数据挖掘中的应用[D].上海交通大学,2008.

[7] Jianyong Wang, Jiawei Han. An Efficient Algorithm for Mining Top-K Frequent Closed Itemsets. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 2005,17:652-664.

[8]穆云婷,谢文阁.基于 FP-Growth 算法的多维关联规则挖掘方法[J].辽宁工业大学学报(自然科学版),2010,30(02):81-83.

[9]陈俊波. 频繁闭合项集挖掘算法及应用研究[D].浙江大学,2009.

[10] Roberto J. Bayardo Jr. Efficiently Mining Long Patterns from Databases: Proceedings of ACM SIGMOD Conference on Management of Data, 1998:85-93.

[11]Thomas H.Cormen,Charles E.Leiserson,Ronald L.Rivest,Clifford Stein,殷建平,徐云,王刚,刘晓光,苏明,邹恒明,王宏志.算法导论(原书第 3 版)[J].计算机教育,2013(12):18.

致 谢

本学位论文是在我的指导老师添玉老师悉心指导之下完成的。在此要感谢所有给予指导、 帮助并且提出建设性意见的老师与同学，衷心感谢添玉老师在我写论文迷茫之时给予的辅导， 给我指明了方向，定期监督辅佐毕业设计的完成，从论文选题开始要求题目切合实际、有研究的意义，添玉老师全程都给与了我巨大的帮助。在论文即将完成之际，在此谨向添玉老师致以诚挚的谢意与崇高的敬意！

附 录

```
#Upgrade-FP-Growth Alogorithm
# -*- coding: utf-8 -*-
"""
@author: liyun yu
"""
"""
导入数据包
"""
import xlwt
import xlrd
import pyfpgrowth
"""
从 excel 中读取数据
"""
wb = xlrd.open_workbook(r'C:\Users\liyun\Desktop\毕设\数据\programming\New Data.xls')
sheet1 = wb.sheet_by_index(0)
originalrow=sheet1.nrows
"""
数值标记法
"""
agvnum=[]
armgnum=[]
task=[]
policy=[]
key=[]
indicator_one=[]
for rows in range(1,originalrow):#excel 列第一个为 0
    agvnum.append(int(sheet1.cell(rows,1).value*1000))
    armgnum.append(int(sheet1.cell(rows,2).value*10000))
    task.append(int(sheet1.cell(rows,3).value))
    policy.append(int(sheet1.cell(rows,4).value))
    key.append(int(sheet1.cell(rows,5).value+1)*100)
    indicator_one.append(round(float(sheet1.cell(rows,16).value)*10))
"""
输出处理后的 transaction list 到新的 excel 表格中
"""
book = xlwt.Workbook()
sheet = book.add_sheet('test', cell_overwrite_ok=True)
colume_name=['agv 数量','armg 数量','task 数量','指派策略','key','indicator 1 list']
row = 0
for item in range(len(colume_name)):
```

```
            sheet.write(row, item, colume_name[item])
        trans_combine=[]
        trans_indi_one=[]
        i=0
        location=0
        row=0
        for rowNum in range(1,originalrow):
            if [agvnum[rowNum-1],armgnum[rowNum-1],task[rowNum-1],policy[rowNum-1],key[rowNum-1]] not in
trans_combine:
                trans_combine.append([agvnum[rowNum-1],armgnum[rowNum-1],task[rowNum-1],policy[rowNum-
1],key[rowNum-1]])
                trans_indi_one.insert(i,[indicator_one[rowNum-1]])
                i=i+1
            else:
                location=trans_combine.index([agvnum[rowNum-1],armgnum[rowNum-1],task[rowNum-1],policy[rowNum-
1],key[rowNum-1]])
                trans_indi_one[location].append(indicator_one[rowNum-1])
        for row in range(1,i+1):
            sheet.write(row,0,trans_combine[row-1][0])#list 的第一个序号为 0
            sheet.write(row,1,trans_combine[row-1][1])
            sheet.write(row,2,trans_combine[row-1][2])
            sheet.write(row,3,trans_combine[row-1][3])
            sheet.write(row,4,trans_combine[row-1][4])
            sheet.write(row,5,str(trans_indi_one[row-1]))
        book.save(r'C:\Users\liyun\Desktop\毕设\数据\programming\test.xls')
        """
        Upgrade-FP-Growth
        """
        transactions=trans_indi_one
        print(transactions)
        patterns = pyfpgrowth.find_frequent_patterns(transactions,1)
        rules = pyfpgrowth.generate_association_rules(patterns,1)
        print("一次关联规则分析:",rules)
        for i in rules.keys():
            if len(rules[i][0])==0:
                pass
            else:
                list_i=[]
                trans_output=[]
                for t in range(0,len(i)):
                    list_i.append(i[t])
                for transaction_items in transactions:
                    Intersection = [k for k in i if k in transaction_items]
                    if Intersection==list_i:
```

```
            trans_output.append(trans_combine[transactions.index(transaction_items)])#找到原数据中的 output
数据
        input_patterns = pyfpgrowth.find_frequent_patterns(trans_output, 6)
        input_rules = pyfpgrowth.generate_association_rules(input_patterns, 1)
        print("二次关联规则分析:",input_rules)
        for n in input_rules.keys():
            list_n=[]
            trans_input=[]
            if len(input_rules[n][0])==0:
                if input_rules[n][0]==1:
                    print("输出数据",i,"和",rules[i][0],"与输入数据",n,"有强关联")
                else:
                    pass
            else:
                medium_list=[]
                medium_intersection=[]
                for m in range(0,len(n)):
                    medium_list=[n[m],input_rules[n][0][0]]
                    medium_intersection = [k for k in medium_list if k in list_n]
                    if medium_intersection == medium_list:
                        pass
                    else:
                        list_n.append(medium_list)
                        if input_rules[n][1] == 1:
                            print("输出数据",i,"和",rules[i][0],"与输入数据",medium_list,"有强关联")
                        else:
                            pass
```