NAIST-IS-DD0961021

**博士論文**

# 時系列パターンマイニングアルゴリズムの高速化手法に関する研究

松原 裕貴

2012年2月2日

奈良先端科学技術大学院大学
情報科学研究科 情報処理学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

松原 裕貴

審査委員：

加藤 博一 教授　（主指導教員）
中島 康彦 教授　（副指導教員）
宮崎 純 准教授　（副指導教員）
有次 正義 教授　（熊本大学）

# 時系列パターンマイニングアルゴリズムの高速化手法に関する研究*

松原 裕貴

## 内容梗概

本論文では，既存の時系列パターンマイニングアルゴリズム PAID の CPU キャッシュの利用効率に着目し，時間的局所性を向上させる CC-PAID と，データアクセスの効率化により計算量と CPU のキャッシュミスの削減を図った CCDR-PAID を提案した．

CC-PAID は，CPU キャッシュに取り込まれた特定のデータ構造を複数の時系列パターンで一度に処理を行う Common-Prefix-At-A-Time といった手法を提案し，PAID に適用したものである．これに対し，CCDR-PAID は CC-PAID を改良することにより，特定のデータ構造を再構築し，複数の処理で有効的にそれを利用することにより，不要なデータへのアクセスと処理の回避を行う．

PAID，CC-PAID，CCDR-PAID の処理時間と CPU のキャッシュミスと実行命令数等の測定を行い，CC-PAID と CCDR-PAID の有効性を確認した．結果として，PAID に対して CC-PAID は最大で約 55%，処理時間が短縮され，CC-PAID に対して CCDR-PAID は最大で約 22%，処理時間が短縮された．また，PAID に対して CCDR-PAID は最大で約 64%，処理時間の短縮が確認された．更に，並列処理による評価も行い，スレッド数 1 に対して 6 の時，PAID，CC-PAID，CCDR-PAID はそれぞれ 4 倍以上の速度向上となった．

**キーワード**

*奈良先端科学技術大学院大学 情報科学研究科 情報処理学専攻 博士論文，NAIST-IS-DD0961021, 2012 年 2 月 2 日.

時系列パターンマイニング, CPU キャッシュの利用効率, アルゴリズム, 並列処理,
性能評価

# A Study on Fast Sequential Pattern Mining Algorithms*

Yuki Matsubara

**Abstract**

In this dissertation, we focused mainly on improving the CPU cache utilization of an existing sequential pattern mining algorithm, called PAID. We have proposed two new algorithms: CC-PAID, which improves the temporal locality of PAID, and CCDR-PAID, which is designed as a data access optimization method to reduce CPU cache misses and the number of executed instructions of CC-PAID.

In CC-PAID, a common-prefix-at-a-time method is introduced to be applied to PAID, so as to process the data of multiple sequential patterns fetched in the CPU cache at once. On the other hand, CCDR-PAID reconstructs a particular data structure and uses it to reduce the CPU cache misses and the number of instructions executed, by avoiding nonessential data accesses.

We measured not only the processing time of PAID, CC-PAID and CCDR-PAID to validate our proposed algorithms, but also the effects when they are executed in parallel. As a result, the processing time of CC-PAID was reduced up to 55% to PAID, and CCDR-PAID was shortened up to 22% to CC-PAID. Furthermore, when the number of threads was changed from one to six, PAID, CC-PAID and CCDR-PAID could run more than four times faster.

*Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0961021, February 2, 2012.

**Keywords:**

sequential pattern mining, CPU cache utilization, algorithm, parallel processing, performance evaluation

# 目 次

# 図 目 次

# 表 目 次

# 第1章　序論

## 1. 研究背景

時系列パターンマイニング [3] とは，時系列データにより構成されたデータベースから時系列パターンと呼ばれる出現順序を維持した状態の特定の閾値を満たす部分列を発見する手法であり，大規模なデータベースから利用者にとって有用な情報を発見する手段として用いられる．応用分野としては，時間や順序といった情報が付加されたデータを含むデータベースを対象とし，データ間の時間や順序関係が重要となる分野（データ解析，行動予測，情報推薦，センサネットワーク等）への応用が考えられ，データベースの規模や閾値は様々なものとなる．

代表的な例としては，マーケティングへの応用が挙げられる．ある百貨店の一年間の購買履歴のデータベースを対象に時系列パターンマイニングを利用し，得られた時系列パターンの中から，今まで知られていなかった有用な時系列パターンとして，例えば「ワイングラスを購入した人の６割が３週間以内にジャケットを購入する」といったことがわかったとする．この結果を元に，顧客の行動を予測した商品の品揃や価格等の調整といったマーケティング戦略が可能となる．

一般に，時系列パターンマイニングにおける処理コストの増加要因として最小支持度とデータベースの規模が挙げられる．最小支持度とは，時系列パターンの抽出基準として用いられる閾値であり，この閾値が小さいほど多くの時系列パターンの抽出処理が発生する．また，時系列パターンの発見処理では多くのアルゴリズムにおいて，データベース全体に対するスキャンとデータのカウント処理が行われており，これらが処理時間の大半を占めるとされている．このため，小さな最小支持度と大規模なデータベースでは処理時間が膨大になるといった傾向があり，解決すべき重要な問題として，過去の研究においてアルゴリズムの提案

[3][15][21][4][12][16][19][18]，並列化による処理時間の改善 [14][20][24] が取り組まれてきた．

しかしながら，情報爆発時代の今日においては処理対象となる多くのデータベースはより大規模になると想定され，それに伴い処理時間も大幅に増加することが考えられる．このため，実際の利用において処理時間に制約がある場合は時系列パターンマイニングの処理を適用できない可能性もある．また，処理時間を減らすために最小支持度を大きく設定することは効果的だが，この場合は有用とされる時系列パターンを取りこぼすといった問題も生じる．更に，抽出されたパターンを利用して他の処理を行い有用な情報を発見するといったケースも考えられ，その場合，更に多くの時間が必要であるため，時系列パターンマイニングの処理時間の改善は今後も重要な課題の一つと考えられる．

近年では，処理時間を短縮するためのアルゴリズムの提案や並列化といったアプローチ以外に，相関ルールマイニング [2] の分野では，時間的局所性と空間的局所性等を向上させ，CPU キャッシュの利用効率の改善による処理速度の向上が試みられている [8]．

時間的局所性と空間的局所性は以下の 2 つを意味する [26]．

- 時間的局所性：アクセスされた命令またはデータは近い将来再びアクセスされる可能性が高い．
- 空間的局所性：アクセスされた命令またはデータの近くのものもアクセスされる可能性が高い．

これらの局所性は CPU キャッシュの利用効率に関係し，本論文で我々が提案する手法においても重要な概念である．

時系列パターンマイニングは相関ルールマイニングとよく似た性質を持っており，アルゴリズムの特性上，特定のデータ構造に対して再帰的なアクセスが発生する傾向がある．このため，CPU のキャッシュサイズを超えるようなデータ構造へのアクセスが生じる場合，キャッシュミスによるデータアクセスのレイテンシが発生する可能性がある．

また，CPU キャッシュのサイズは年々増加しているものの CPU のマルチコア化により，1 コアあたりの CPU キャッシュのサイズを考えると非常に小さい．更

に，CPU の速度向上に対するメインメモリのアクセス速度の差があるといったメモリの壁 [17] の問題が存在し，メインメモリのアクセス速度も向上はしているが CPU のマルチコア化が進む今日においては，その差が埋まることは考えにくい．

このため，大規模なデータベースを対象とした時系列パターンマイニングの処理では，CPU のキャッシュミスにより生じるアクセスレイテンシはボトルネックとなる可能性があり，CPU のキャッシュミスの軽減に着目した CPU キャッシュの利用効率の改善による処理時間の短縮手法の提案は有効的なアプローチの一つであると考えられる．また，データベースの規模が大きい場合，計算量も多くなることが予測されるため，計算量の削減も重要なアプローチと考えられる．更に，近年では CPU のマルチコア化が進んでおり，並列処理による処理時間の短縮も重要と考えられる．

## 2. 提案手法の概要

本論文で提案される手法は，Yang らにより提案された時系列パターンマイニングアルゴリズム PAID [18] の CPU キャッシュの利用効率の向上と計算量及び CPU のキャッシュミスの削減を行うものである．

PAID アルゴリズムを改良対象に採用した理由として，第一に処理速度が挙げられる．PAID は SPADE [21]，PrefixSpan [12]，LAPIN [19] といった他の時系列パターンマイニングアルゴリズムよりも処理速度が速いことが性能評価により証明されている．第二に，Yang らは PAID の提案の前に LAPIN アルゴリズム [19] の提案を行っており，PAID ではその LAPIN で利用される item-last-position list と呼ばれるデータ構造を用いている．item-last-position list は時系列データ上に繰り返し出現するアイテムの最終出現位置をまとめたものであり，これを支持度と呼ばれる部分列の出現割合の調査に用いることにより，時系列データ上に重複して出現するアイテムのスキャンを避け，効率的に支持度を調べることができる．このため，DNA シーケンス [6] のような同じアイテムが繰り返し出現するようなデータベースを効率的に処理可能と考えられている．

近年では，センサネットワークの発展や携帯端末の普及により時間情報が付加

されたデータを容易に取得出来るようになると考えられ，時間の経過に伴い得られるデータは時系列データ長として増加することが考えられる．このため，PAIDのように時系列データ長が長い場合でも効率的に処理を行える可能性があるアルゴリズムは重要と考えられ，処理速度の向上のための新たなアプローチの適用対象として，PAID アルゴリズムを採用した．

我々は PAID アルゴリズムの中核部分を変更することなく，下記の二点のアプローチにより CPU キャッシュの利用効率の向上と計算量及び CPU のキャッシュミスの削減を行う手法の提案に取り組んだ．

- 一点目のアプローチとして CPU キャッシュに取り込まれた特定のデータ構造等を複数の時系列パターンで一度に処理を行う Common-Prefix-At-A-Time といった手法を提案し，CPU キャッシュの再利用率を向上させる CC-PAID アルゴリズムの提案を行った．

- 二点目のアプローチは，CC-PAID を改良し，特定のデータ構造を動的に再構築し，その再構築されたデータ構造を複数の処理で有効的に利用することにより，データアクセスの省略や不要な処理の回避を行う CCDR-PAID の提案を行った．

また，性能評価では PAID，CC-PAID, CCDR-PAID の処理時間と CPU のキャッシュミスと実行命令数を計測することにより有効性の確認を行った．更に，近年では CPU のマルチコア化が進む傾向にあり，処理速度を向上させるには並列処理も重要と考え，PAID，CC-PAID，CCDR-PAID を並列化して性能評価を行った．

## 3. 論文構成

第 2 章では，時系列パターンマイニングの定義や処理方法等の説明を行い，時系列パターンマイニングを利用した際の時系列パターンの発見例も示す．更に，本論文の提案手法の改良対象となっている PAID アルゴリズムを疑似コードや例題を用いて紹介する．

第 3 章では，過去に提案された相関ルールマイニングと時系列パターンマイニングのアルゴリズムを紹介する．

第 4 章では，PAID アルゴリズムで利用される特定のデータ構造の CPU のキャッシュ利用効率に着目し，時間的局所性の改良のために Common-Prefix-At-A-Time といった手法を提案し，それを PAID に適用させた CC-PAID アルゴリズムに関して説明を行う．

第 5 章では，CC-PAID のデータアクセスの効率化に着目し，特定のデータ構造へのアクセスを回避することにより，計算量と CPU のキャッシュミスの削減を行う CCDR-PAID アルゴリズムについて説明する．

第 6 章では，PAID，CC-PAID，CCDR-PAID を用いて処理時間と CPU のキャッシュミス及び実行命令数を計測し，実験結果と考察を示す．また，並列処理とメモリ使用量の評価も行う．

第 7 章では，CC-PAID 及び CCDR-PAID の概要と評価についてまとめ，今後の課題等を示す．

# 第2章　時系列パターンマイニング

## 1. 時系列パターンマイニングの定義

本論文で用いられる時系列パターンマイニング [3] ($sequential\ pattern\ mining$) に関する用語定義及び説明を行う.

時系列パターンマイニングは時系列データベースからユーザが決定した特定の閾値を満たす出現順序を維持した部分列を時系列パターンとして発見する手法である.

時系列データベース ($sequence\ database$) は複数の時系列データ ($sequence\ data$) により構成され，時系列データはアイテムセット ($item\ set$) により構成される. 時系列データベースは，$SDB =< s_1, s_2, \cdots, s_n >$ と示され，$s_k$ は時系列 ID（$SID$）が付与された時系列データであり，SID 順に並んだ時系列データで構成される. 時系列データ $s$ は，$s =< is_1, is_2, \cdots, is_m >$ で表され，$is_k$ はアイテムセットであり，これを 1 つのトランザクションとしてトランザクション ID（$transaction\ id, TID$）と呼ばれる時間情報または出現順序により識別され，TID 順に並べられたアイテムセットで構成される．アイテムセット $is$ は，$is = (i_1, i_2, \cdots, i_c)$ であり，アイテム ($item$) により構成されている．また，同じ時系列データの同一の TID のアイテムセット内ではアイテムが重複することはないが，TID が違う場合は同じアイテムが出現する可能性がある.

時系列データに含まれる TID による順序関係を維持したアイテムセットの組合せを長さ $k$ の部分列 $\alpha$ としたとき，全時系列データの数に対して長さ $k$ の部分列 $\alpha$ が出現する割合（長さ $k$ の部分列 $\alpha$ の出現回数 / 全時系列データの数）を支持度 ($support$) と呼び，出現回数の場合は支持度の値 ($support\ value$) と呼ぶ．もし同じ SID の時系列データ内に同一の部分列 $\alpha$ が複数存在しても，アイテムセッ

ト間の順序関係を考慮する必要があるためにその部分列はユニークであり，その時系列データ内の部分列 $\alpha$ の出現回数は 1 となる．

時系列パターンの発見方法は，処理開始時にユーザが決定した支持度を最小支持度 ($minimum\ support$) とし，時系列データベース内で長さ $k$ の部分列 $\alpha$ の支持度を調べ，最小支持度を満たす全ての長さ $k$ の部分列 $\alpha$ を長さ $k$ の時系列パターンとする．一般に時系列パターンマイニングの処理の進め方として，長さ $k$ の時系列パターンが発見されたとき，その時系列パターンが接頭辞となる長さ $k+1$ の時系列パターンを発見する処理に移行し，時系列データベース内の全ての時系列パターンが発見されるまで再帰的に処理が行われる．

時系列パターンを発見するためのアルゴリズムには，candidate generation-and-test と pattern growth といった二つのアプローチが存在する [12]．前者は発見された時系列パターンを組み合わせて次の時系列パターンの候補を生成し，時系列データベース上でそれらの支持度を調べ，後者は時系列データベース上で時系列パターンが出現する位置より後を投影データベースとし，その範囲内でアイテムの支持度を調べ，投影データベース上で直接的に時系列パターンを伸ばす．

また，時系列パターンマイニングの処理には itemset-extention step （I-step）と sequence-extention step（S-step）の二つの処理が含まれている [4]．長さ $k+1$ の時系列パターンを発見する際に，$k$ 番目のアイテムと同じ TID のアイテムセット内から相関ルールマイニング [2] により，最小支持度を満たす組み合わせを導出して $k+1$ 番目のアイテムを発見するのが I-step であり，時系列パターンマイニングにより，$k$ 番目のアイテムの TID より大きい TID のアイテムセット内から最小支持度を満たすアイテムを発見し，$k+1$ 番目のアイテムとするのが S-step である．

## 2. 時系列パターンマイニングの例

時系列データベースに対して時系列パターンマイニングを適用し，発見される時系列パターンの例を用いて，時系列パターンマイニングの処理を説明する．

例として，図 2.1 の全てが長さ 1 のアイテムセットにより構成される時系列デー

| TID / SID | sequence data | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | A | C | B | C | A | D |
| 2 | A | D | B | A | C | |
| 3 | B | A | A | D | C | A |
| 4 | C | C | D | A | B | A |
| 5 | C | A | C | A | B | A |

図 2.1 時系列データベース

| length | Sequential pattern : Support |
|---|---|
| 1 | A:5, B:5, C:5, D: 4 |
| 2 | A→A:5, A→B:4, A→C:4, B→A:5, C→A:4 |
| 3 | A→B→A:4 |

図 2.2 最小支持度 (回数) 4 を満たす時系列パターン

タベースから最小支持度 (回数) を 4 として，それを満たす全ての時系列パターンが発見される例を図 2.2 に示す.

図 2.1 の時系列データベースについて説明する．図 2.1 の時系列データベースは 5 つの時系列 ID($SID$) により識別される時系列データにより，SID 順に構成されており，各時系列データにはトランザクション ID($TID$) により識別されたアイテムセットが含まれており，TID 順に並べられている．例として，SID 1 の時系列データは A，C，B，C，A，D の出現順で得られたそれぞれ長さ 1 のアイテムセットにより構築されており，そのアイテムセットが出現した順序関係である TID で表わされている.

次に図 2.1 の時系列データベースに対し，最小支持度を出現回数 4 とした時に発見される全ての時系列パターンの例を図 2.2 に示す．図 2.2 の例では発見された時系列パターンが長さごとに分けられ，時系列パターンとその出現回数を表している．例として，長さ 3 の時系列パターンに A $\rightarrow$ B $\rightarrow$ A: 4 があるが，これは長さ 3 の時系列パターン A $\rightarrow$ B $\rightarrow$ A （“ $\rightarrow$ ”はトランザクション ID による順序関係を示す）が図 2.1 の時系列データベース上でに 4 回出現することを意味する.

また，図 2.1 の SID 3 の時系列データにアイテム A が 3 つ含まれているが，これらのアイテムは TID による順序関係があるため，その時系列データにおいてアイテム A の出現回数が 3 となるのではなく，部分列 A，A $\rightarrow$ A，A $\rightarrow$ A $\rightarrow$ A

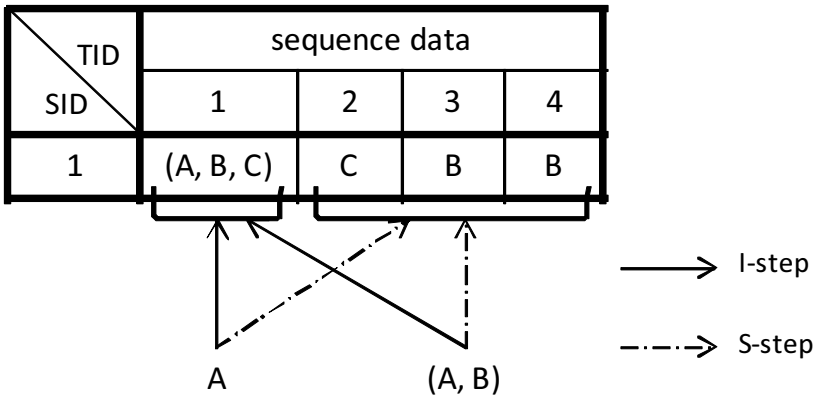



図 2.3 I-step と S-step の例

の出現回数はそれぞれ 1 となる．長さ $k$ の部分列には順序関係の制約があるため，一つの時系列データに一回しか出現しなく，支持度を出現回数とした場合，その最大値は時系列データ数となる．このため，図 2.1 の例においては時系列データ数が 5 であるため，長さ $k$ の部分列 $\alpha$ の出現回数も 5 回以上になることはない．

## 2.1 S-step と I-step の例

時系列パターンマイニングアルゴリズム Prefixspan [12] のような手法で図 2.3 の時系列データベースから長さ k+1 の時系列パターンを発見する場合の I-step と S-step の例を示す．時系列データ数が 1 つの時系列データベース (ABC)CBB があり（“ () ”は一つのアイテムセットを表す），最小支持度 (回数) を 1 と設定する．例として，長さ 1 の時系列パターン A が接頭辞となる長さ 2 の時系列パターンを発見する場合，時系列データベース上で長さ 1 の時系列パターン A 以降の範囲を投影データベース (_BC)CBB とする（(_BC) は時系列パターン A と同じ TID に出現することを表す)．この投影データベース内で接頭辞が時系列パターン A である長さ 2 の部分列と成るアイテムの支持度を調べる．I-step では時系列パターン A と同じ TID である (_BC) のアイテムの支持度を調べ，結果として時系列パターン (AB), (AC) が発見される．S-step では時系列パターン A の TID よ

り大きい TID 2 以降のアイテムセットに含まれるアイテムの支持度を調べ，結果として時系列パターン A $\rightarrow$ C，A $\rightarrow$ B が発見される．時系列パターン (AB) を対象に長さ $k+1$ の時系列パターンの発見処理を行うと，TID 1 のアイテムセットに対して I-step を適用し，TID 2 以降の範囲のアイテムセットに対して S-step を適用する．

また，本論文の提案手法は時系列パターンの発見処理となる S-step に着目している．I-step は相関ルールマイニングと同じであるため，次節からの時系列パターンマイニングアルゴリズムの説明については S-step に関するものとする．

# 3. 時系列パターンマイニングアルゴリズム PAID

PAssed Item Deduced sequential pattern mining (PAID) [18] は，Yang らによって提案された時系列パターンマイニングアルゴリズムである．これは，長さ $k$ の時系列パターンから長さ $k+1$ の時系列パターンの発見を行う際，時系列データ上で長さ $k$ の時系列パターンが出現する位置 (TID) より後ろの範囲で出現しなくなるアイテムを調べ，長さ $k$ の時系列パターンの出現回数（実際には，時系列データベース上で長さ $k-1$ の時系列パターンが出現する位置より後ろのアイテムの出現回数）から，それらのアイテムを差し引くことにより支持度を更新する．他の時系列パターンマイニングアルゴリズムは支持度の調査のためにアイテムまたは部分列をカウントするのに対し，差し引き処理により前の処理で得られた支持度の値を有効的に活用し，支持度の値を更新する点が大きな違いとなる．

## 3.1 PAID のアクセスパターン

時系列パターンマイニングでは，部分列や時系列パターンは辞書式順序木で表わすことができる [4]．PAID では深さ優先探索により一つの長さ $k$ の時系列パターンを選択し，長さ $k+1$ の時系列パターンを発見する．

例として，2 章 2 節で用いた最小支持度 (回数) 4 と時系列データベース (図 2.1) と発見される時系列パターン (図 2.2) の場合の時系列パターン A $\rightarrow$ B $\rightarrow$ A が発見されるまでのアクセスパターンを示す (図 2.4)．枠で囲まれた時系列パターン

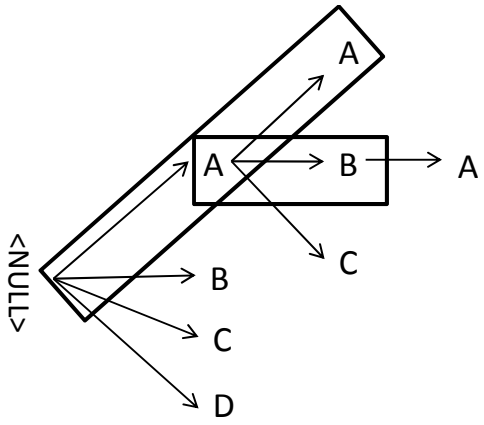



図 2.4 時系列パターンを表す木構造への PAID のアクセスパターン

は処理が終了したことを表す．PAID では，時系列パターン A → A の処理終了後，深さ優先探索で時系列パターン A → B が一つ選択され，時系列パターン A → B → A が発見される．

## 3.2 処理の流れとデータ構造

PAID アルゴリズムは以下のように大きく分けて 2 つの処理と 2 つのデータ構造に分けられる．

1. position list から長さ $k$ の時系列パターンの prefix border position を取得
2. item-last-position list を用いた支持度の値の更新 (非出現なアイテムの調査と差し引き処理)

position list は時系列データに含まれるアイテムの TID をアイテムの種類ごとにまとめたものであり，それらの TID は昇順にソートされたデータ構造である．item-last-position list (ILP-list) は時系列データに含まれる長さ 1 の時系列パターンとそれが時系列データ上で最後に出現する TID の組（TID, ItemID）を要素と

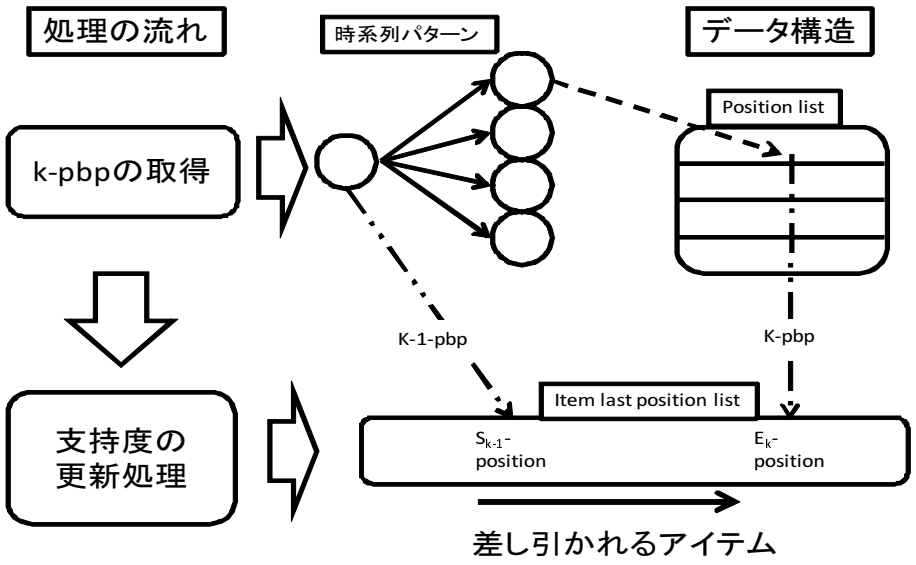



図 2.5 PAID のプロセスフローとデータ構造

して記録したデータ構造であり，それらの要素は TID 順にソートされている．時系列データベースに対応する全ての ILP-list を ILP-list データベース (ILP-DB) とし，position list を position list データベース (position list-DB) とする．また，長さ $k$ の時系列パターンの prefix border position ($k$-pbp) とは，長さ $k$ の時系列パターンが時系列データ上に出現する位置 (TID) であり，その TID は $k$ 番目のアイテムのものである．図 2.1 の時系列データベースを対象とした ILP-DB と position list-DB を図 2.6 と図 2.7 に示す．

処理の流れとデータ構造の概要を図 2.5 に示す．

まず，$k$-pbp の取得処理を行うために，処理対象の長さ $k$ の時系列パターンに対応する position list にアクセスし，$k-1$-pbp より大きい TID を取得する．($k-1$-pbp は，長さ $k$ の時系列パターンの接頭辞となる長さ $k-1$ の時系列パターンが時系列データ上に出現する位置 (TID) である．)

次に，得られた $k$-pbp を用いて支持度の更新処理を行うために，ILP-list にアクセスし，$k-1$-pbp と $k$-pbp より大きい TID の要素のインデックスを調べる．ここで，ILP-list 上のそれぞれのインデックスを $S_{k-1}$-position，$E_k$-position とする．

| SID | Item ID | Transaction ID |
|---|---|---|
| 1 | A | 1, 5 |
| | B | 3 |
| | C | 2, 4 |
| | D | 6 |
| 2 | A | 1, 4 |
| | B | 3 |
| | C | 5 |
| | D | 2 |
| 3 | A | 2, 3, 6 |
| | B | 1 |
| | C | 5 |
| | D | 4 |
| 4 | A | 4, 6 |
| | B | 5 |
| | C | 1, 2 |
| | D | 3 |
| 5 | A | 2 , 4, 6 |
| | B | 5 |
| | C | 1, 3 |
| | D | |

図 2.6 position list データベース

| SID | Item Last Position List (Transaction ID, Item ID) |
|---|---|
| 1 | (3, B), (4, C), (5, A), (6, D) |
| 2 | (2, D), (3, B), (4, A), (5, C) |
| 3 | (1, B), (4, D), (5, C), (6, A) |
| 4 | (2, C), (3, D), (5, B), (6, A) |
| 5 | (3, C), (5, B), (6, A) |

図 2.7 item last position list データベース

このとき，$S_{k-1}$-position から $E_k$-position までの範囲に含まれるアイテムが非出現なアイテムと判断される．非出現なアイテムとは，時系列データ上の長さ $k-1$ の時系列パターンが出現する位置より後ろの範囲で，長さ $k$ の時系列パターンが出現する位置より後ろに出現しなくなるアイテムを表す．ILP-list の各要素は長さ 1 の時系列パターンの最終出現位置であるため，$S_{k-1}$-position，$E_k$-position がわかれば非出現なアイテムを判断できる．

続けて，非出現なアイテムを長さ $k$ の時系列パターンの投影 ILP-DB ($k$-projILPDB) 内のアイテムの支持度の値となる $k$-sup_list からそれぞれ差し引く．$k$-sup_list は，長さ $k+1$ の時系列パターンの発見処理時に $k-1$-sup_list の支持度の値が初期値として与えられており，全ての時系列データに対する処理が終了した際に長さ $k$ の

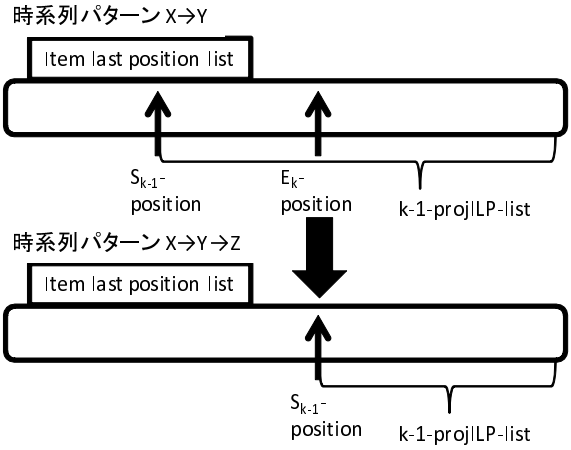



図 2.8 $S_{k-1}$-position と $E_k$-position

時系列パターンの投影 ILP-DB ($k$-projILPDB) 内のアイテムの支持度の値として $k$-sup_list が決定され，最小支持度を満たすアイテムが長さ $k$ の時系列パターンに連結され，長さ $k+1$ の時系列パターンが発見される．$k$-projILPDB は，ILP-DB 上で長さ $E_k$-position 以降の範囲を表し，$k$-sup_list はその範囲内のアイテムの出現回数を表す．また，ILP-list 上の長さ $E_k$-position 以降の範囲を $k$-projILP-list とする．

なお，ILP-list は要素の TID によりソートされているため，得られた $E_k$-position は，次の長さ $k+1$ の時系列パターンの発見処理時に $S_{k-1}$-position として利用可能である．このため，$E_k$-position は $k-1$-projILP-list の範囲から探すこととなり，時系列パターン長が長くなると $k-1$-projILP-list の範囲は小さくなる．例として，図 2.8 に時系列パターン $\mathrm{X} \rightarrow \mathrm{Y}$ から時系列パターン $\mathrm{X} \rightarrow \mathrm{Y} \rightarrow \mathrm{Z}$ が発見された際の ILP-list に対する $S_{k-1}$-position，$E_k$-position と $k-1$-projILP-list を示す．

## 3.3 PAID のアルゴリズム

PAID のアルゴリズムを疑似コードと例を用いて説明する．

```
PAID Algorithm
Input : A sequence database, minimum support
Output : All sequential patterns

Main
1. POS-DB, ILP-DB, FI_s, sup_list ← Scan_SDB()
2. For each item fi in FI_s
   2.1 Call Gen_(k+1)-patterns(fi, 0, sup_list)

Function Gen_(k+1)-patterns (k-sp, (k-1)-pbp_set, (k-1)-sup_list)
3. For each sequence sd
   3.1 k-pbp ← find_k_pbp((k-1)-pbp)
   3.2 S ← find_S_pos((k-1)-pbp)
   3.3 E ← find_E_pos((k)-pbp)
   3.4 while(S != Null && S < E)
       3.4.1 k-sup_list[S.ItemID]--; //k-sup_list is a copy of (k-1)-sup_list.
       3.4.2 S++;
4. FI_s ← get_freq_item(k-sup_list)
5. For each item fi in FI_s
   5.1 (k+1)-sp ← extend_k-sp(fi, k-sp)
   5.2 Call Gen_(k+1)-patterns((k+1)-sp, k-pbp_set, k-sup_list)
```

図 2.9 PAID アルゴリズムの疑似コード

### 疑似コード

PAID アルゴリズムの疑似コード (図 2.9) は参考文献 [18] を基に作成した．以下に，図 2.9 の疑似コードの各ステップについて説明を行う．

- Step 1
  前処理として時系列データベースから position list-DB (POS-DB)，ILP-DB，最小支持度を満たすアイテム集合 (FI_s)，ILP-DB 内のアイテムの支持度 (sup_list) の調査及び生成を行う．

- Step 2〜2.1
  最小支持度を満たすアイテム毎に，長さ $k$ の時系列パターン，$k$-pbp を SID 毎にまとめたセット ($k$-pbp_set)，sup_list を引数として，長さ $k+1$ の時系列パターンの発見処理に移行．

- Step 3
  時系列データ毎に Step 3.1〜3.4.2 を実行．

- Step 3.1

長さ $k$ の時系列パターン (k-sp) に対応するアイテムの position list にアクセスし，$k-1$-pbp より大きい TID を探し，$k$-pbp を取得.

- Step 3.2〜3.3
  ILP-list 上で $k-1$-pbp，$k$-pbp より大きいインデックスとして，$S_{k-1}$-position (S)，$E_k$-position (E) を取得.

- Step 3.4〜3.4.2
  $S_{k-1}$-position (S) から $E_k$-position (E) までの範囲に含まれる要素内のアイテムを $k$-sup_list から差し引く.

- Step 4
  $k$-sup_list から最小支持度を満たすアイテムを全て抽出.

- Step 5
  最小支持度を満たすアイテム毎に Step 5.1〜5.2 を実行.

- Step 5.1
  最小支持度を満たすアイテムを長さ $k$ の時系列パターンに連結し，長さ $k+1$ の時系列パターンとする.

- Step 5.2
  次の長さ $k+1$ の時系列パターンの発見処理に移行.

ここで，本論文の PAID 及び提案手法のアルゴリズムと実装に関する補足を行う．参考文献 [18] の PAID のアルゴリズムの疑似コードでは $k$-pbp の取得処理は時系列データごとではなく，図 2.9 の Step 3 の前で POS-DB からまとめて $k$-pbp_set を取得し，時系列データ毎の処理で $k$-pbp_set から $k$-pbp の取得を行っている．これに対し，本論文の PAID と提案手法の実装および説明では，図 2.9 のように時系列データ毎に $k$-pbp を取得する処理となっている.

また，$k-1$-pbp が NULL の場合，時系列データ上に長さ $k-1$ の時系列パターンが存在しないことを意味し，支持度の更新に関連した処理が回避可能であり，その時系列データに関する Step 3.1〜3.4.2 までの処理を回避できる．このため，

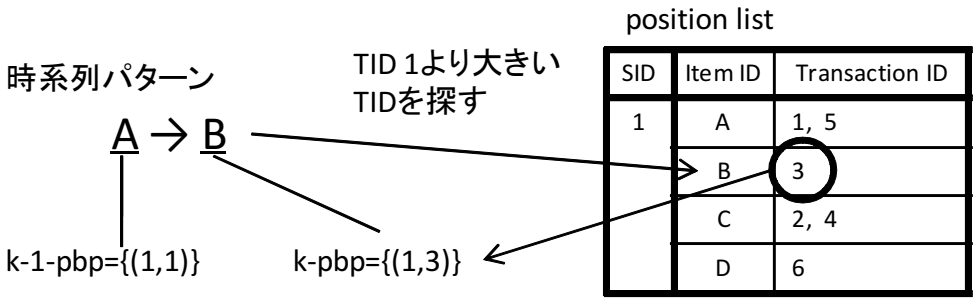



図 2.10 PAID アルゴリズムによる $k$-pbp の取得

時系列パターン長が長くなると処理される時系列データが少なくなる可能性がある.

## 3.4 PAID アルゴリズムの処理の例

図 2.1 の時系列データベースを対象に最小支持度 (回数) を 4 と設定し，長さ 2 の時系列パターン A→B から長さ 3 となる時系列パターンの発見処理の例を示す.

まず，図 2.6 の position list-DB の SID 1 の ItemID B の position list から時系列パターン A の prefix border position ($(k-1)$-pbp) を用いて，時系列パターン A $\rightarrow$ B の pbp ($k$-pbp) を調べる．結果として時系列パターン A と A $\rightarrow$ B の pbp は $k-1$-pbp $= \{(1,1)\}$，$k$-pbp $= \{(1,3)\}$ となる (図 2.10)．なお，本論文では pbp は SID と TID の組で表す.

次に，$(k-1)$-pbp $= \{(1,1)\}$，$k$-pbp $= \{(1,3)\}$ を用いて，図 2.7 の ILP-DB の SID 1 の ILP-list でそれらの TID より大きくなる位置を $S_{k-1}$-position，$E_k$-position として調べる (図 2.11)．また，図 2.11 の SID 毎のデータ内の要素は，(長さ 1 の時系列パターンの最終出現位置 (TID), Item ID) の組である．位置を調べた結果から SID 1 の ILP-List 上で時系列パターン A $\rightarrow$ B より後に出現しないアイテム (非出現なアイテム) が B であることが分かる (図 2.11)．A-sup_list (時系列パターン A の sup_list) を初期値とした A $\rightarrow$ B-sup_list (時系列パターン A $\rightarrow$ B の sup_list) から非出現なアイテム B を一つ差し引く．このような処理を全ての

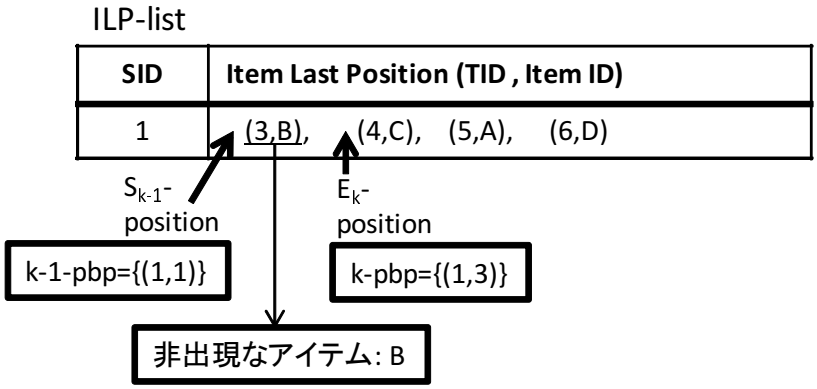



図 2.11 PAID アルゴリズムによる ILP-list を用いた非出現なアイテムの発見の例

SID を対象に行うことにより，A $\rightarrow$ B-sup_list の値が決定される (図 2.12).

この結果から，4 回以上出現するアイテムは A であることがわかり，時系列パターン A $\rightarrow$ B が接頭辞となる長さ 3 の時系列パターン A $\rightarrow$ B $\rightarrow$ A として発見される.

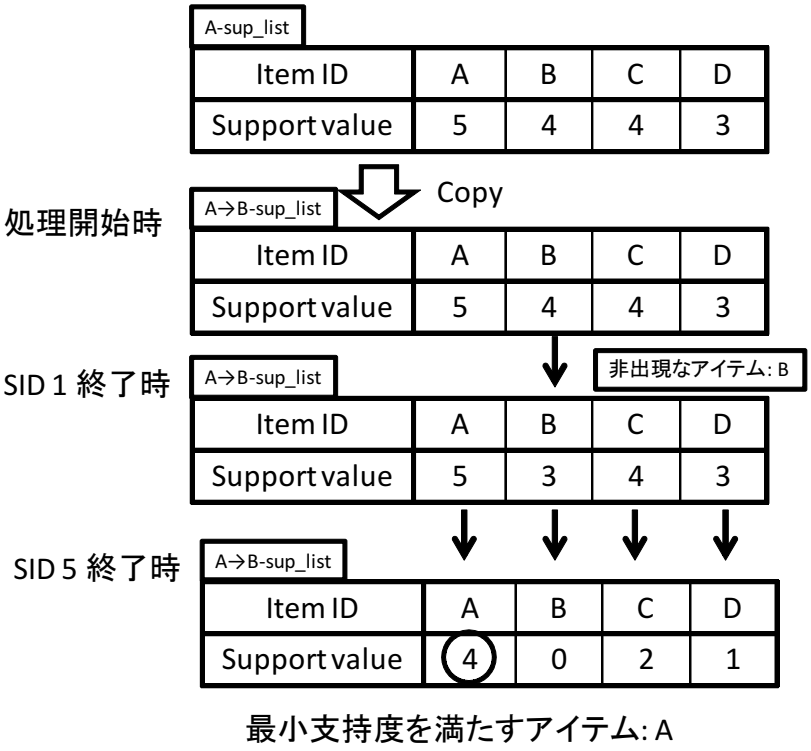



図 2.12　支持度の値の更新の例

# 第3章　関連研究

本章では，既存の相関ルールマイニングアルゴリズム Apriori と時系列パターンマイニングアルゴリズム AprioriAll，SPADE，Prefixspan，FSPM，LAPIN を紹介する．また，相関ルールマイニングの分野での CPU キャッシュの利用効率の改善手法についても紹介を行う．

Apriori は，大規模なデータベースから相関ルールを発見する基礎的なアルゴリズムであり，Apriori を拡張することにより時系列パターンを発見可能とした AprioriAll の提案が行われた．SPADE はデータベースへの I/O コストを考慮したアルゴリズムであり，メインメモリ上で独立して処理が行えるよう処理する時系列パターンを分ける手法が提案されている．また，Prefixspan は，AprioriAll といった候補の生成とその候補の支持度の調査といったアプローチに対し，時系列データベース上で直接的に時系列パターンを発見する基礎的なアルゴリズムである．これに対し，FSPM はデータへのアクセス効率を考慮し，Prefixspan のデータのスキャンコストを減らす提案が行われており，更に時系列パターンの発見処理で必要なデータのみにアクセスする手法も提案されており，本論文の提案手法 CCDR-PAID の提案に関係している．LAPIN では本研究の CPU キャッシュの利用効率の向上手法の適用対象としている PAID で用いられる item-last-position list が提案されている．また，相関ルールマイニングの分野での CPU キャッシュの利用効率の改善手法では空間的局所性と時間的局所性の改良が行われており，本論文の提案手法 CC-PAID の提案に関係している．

# 1. Apriori

Agrawal らは，データベースから X ⇒ Y といった相関ルールを発見する手法として Apriori アルゴリズムの提案を行った [2]．Apriori は，アイテムセットを一つのトランザクションとし，それらによって構成されたデータベースから，全ての最小支持度を満たす部分的なアイテムセットを large itemset として発見する．

最小支持度を満たす $k-1$ 個のアイテムを含むアイテムセットを large itemset としたとき，それらを結合 (join) することにより，$k$ 個のアイテムを含むアイテムセットを生成し，これを large itemset となる可能性のあるアイテムセット candidate itemset とする．それぞれの candidate itemset の支持度をデータベースをスキャンすることによって調べ，最小支持度を満たすものを large itemset として発見する．また，candidate itemset に large itemset ではないものが含まれていた場合，その candidate itemset は削除される．このような処理を再帰的に行うことにより，全ての large itemset を発見する．

また，X ⇒ Y といった相関ルールには信頼度 (confidence) があり，これは X が出現する数に対して X ⇒ Y が出現する割合を表し [9]，ユーザが設定する特定の信頼度を最小信頼度とし，最小支持度と合わせて閾値として用いられる場合がある．

例として，最小支持度 (回数) を 2 とし，図 3.1 に示すデータベースを処理する．まず最小支持度を満たすアイテムをデータベースから調べ，結果として large itemset L1 を得る．L1 のアイテムセットを結合し，candidate itemset C2 を生成する．C2 のアイテムセットの支持度の値をデータベースから調べ，最小支持度を満たすアイテムセットを large itemset L2 として発見する．続けて，L2 のアイテムセットを結合するとアイテムセット {1,2,3}，{1,3,4}，{2,3,4} となる．しかし，{1,2,3} のサブセット {1,2}，{1,3,4} のサブセット {1,4}，{1,2,3} のサブセット {2,4} は L2 に含まれていないため，candidate itemset から削除される．

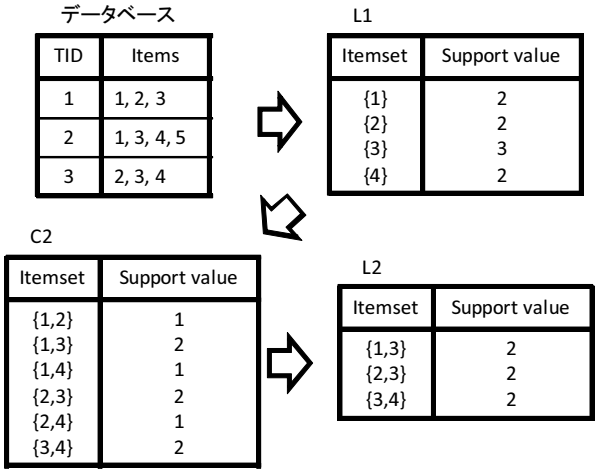



図 3.1 Apriori の処理の例

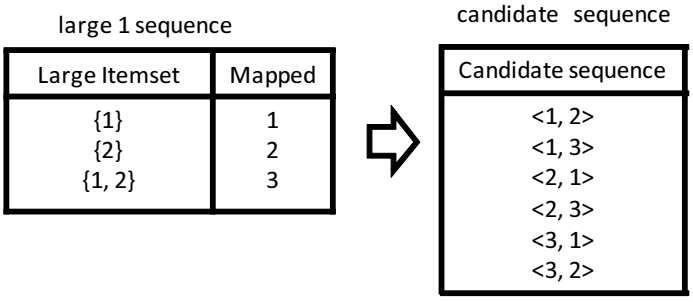



図 3.2 AprioriAll のマップされた large sequence と candidate sequence の例

## 2. AprioriAll

Agrawal らは，時系列パターンマイニングアルゴリズム AprioriAll の提案を行った [3]．これは相関ルールマイニングアルゴリズム Apriori を基に時系列データベースから時系列パターンが発見できるように拡張されたアルゴリズムである．最初に得た large itemset(長さ 1 の large sequence) を一つの要素にマップし，それを用いて時系列データベースのマップも行う．発見された large sequence 間で結合を行い，candidate sequence (large sequence の候補) を生成し，それらのカウントを行い，最小支持度を満たすものを次の large sequence として発見する．

例として，ある時系列データベースと最小支持度を用いた際のマップされた large sequence と candidate sequence を図 3.2 に示す．長さ 1 の large sequence に large itemset {1}, {2}, {1, 2} が存在する場合，これらを 1, 2, 3 といった要素にマップし，それぞれを結合することにより，candidate sequence を生成する．時系列パターンマイニングではアイテムセット間に順序関係があるため，結合では $\langle 1,2\rangle, \langle 2,1\rangle$ 等も生成される．AprioriAll では発見された large sequence の中から，時系列データに唯一存在するものを時系列パターンとしている．

また，AprioriAll のような手法に対して時間制約等を用いることにより，より高速な処理が可能となる GSP アルゴリズム [15] の提案が，Srikant らによって行われた．

## 3. Cache-conscious frequent pattern mining on a modern processor

相関ルールマイニングの分野ではアルゴリズムをキャッシュ指向化して高速化を行う研究も行われている．Ghoting らは，相関ルールマイニングアルゴリズム FP-growth [10] で利用されるデータ構造 FP-tree に対し，ノードをボトムアップ式で横断する際の速度を向上させるために，メインメモリ上で連続的にノードにアクセスできるようにデータの再配置等の改良を行い，空間的局所性の改良を行った Cache-conscious prefix-tree を提案した [8]．また，Cache-conscious prefix-tree

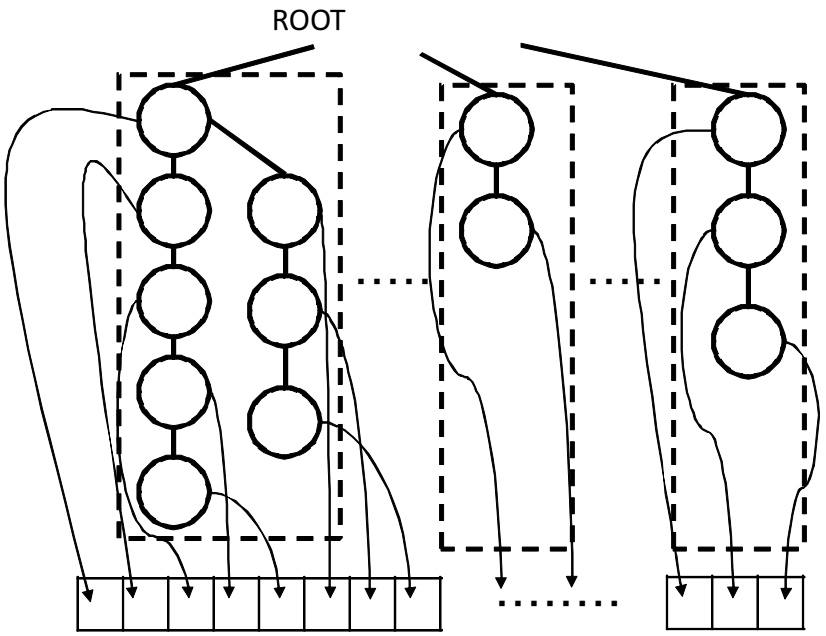



図 3.3 Cache-conscious prefix-tree と Tile の例

を CPU のキャッシュに収まるようにタイル (Tile) といった単位に分割し，フェッチしたタイルを CPU キャッシュ上で無駄なく利用することにより時間的局所性の改良も行っている．この他にも，マルチスレッドによりデータの再利用率を向上させ，最大で約 4.8 倍の速度向上が達成されている．

図 3.3 に Cache-conscious prefix-tree と Tile の例を示す．矢印はメインメモリ上の各ノードの配置を示し，点線の枠はタイルを表す．

なお，FP-growth は，FP-tree 上で相関ルールが出現するより前の部分木を調べることにより，新たな相関ルールの発見を行う Han らによって提案された手法である [10]．FP-tree は，データベース上の最小支持度を満たすアイテムをノードとした木構造で，支持度の値が大きい順でソートされている．また，それぞれのノードは支持度の値等を保持し，ノードは同じアイテム間でリンクしている．

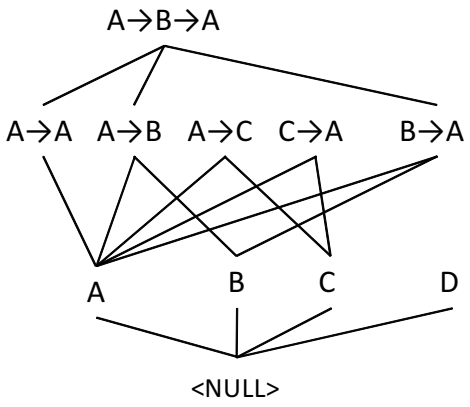



図 3.4 SPADE アルゴリズムの lattice の例

# 4. SPADE

Zaki は，時系列データベース上に出現する時系列パターンを時系列パターン毎に SID とその TID をまとめた id-list を結合し，それを用いて支持度の値を調べるといった手法により，データベースのスキャンによる I/O コストと計算コストを減らすことを目的とした時系列パターンマイニングアルゴリズム SPADE [21] の提案を行った．id-list 間で同じ SID の接頭辞となる時系列パターンよりも大きい TID をすべて選択，結合し，結合された id-list により支持度の値を決定する．

また，時系列パターンの探索範囲を格子構造 (lattice) で表し，メインメモリ上で独立して処理するために，時系列パターンの接頭辞毎に分けられた探索範囲をクラスとして分割する手法も提案されている．また，この手法はメインメモリに収まるまで再帰的に分割することも可能とされている．

2 章 2 節の例題を対象とした lattice と時系列パターン A $\rightarrow$ B $\rightarrow$ A を発見するための時系列パターン A $\rightarrow$ A と A $\rightarrow$ B の結合の例を，図 3.4 と図 3.5 に示す．

# 5. Prefixspan

Pei らは，AprioriAll や GSP といった時系列パターンの候補を生成し，その支持度の値を調べるといったアプローチに対し，時系列データベース上で直接的に

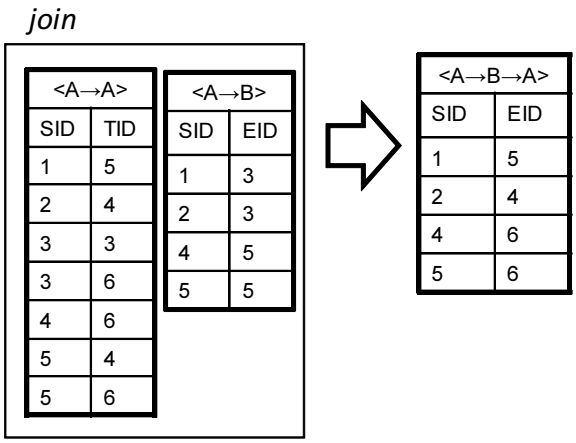



図 3.5 SPADE アルゴリズムの id-list の結合の例

長さ $k$ の時系列パターンから長さ $k+1$ の時系列パターンの発見処理を再帰的に行い，時系列パターン長を伸ばすアプローチとして，時系列パターンマイニングアルゴリズム Prefixspan [12] の提案を行った．これは，時系列データベース上で長さ $k$ の時系列パターンが出現する位置より後の範囲を投影時系列データベースとし，その範囲内のアイテムの支持度を調べ，最小支持度を満たすアイテムを長さ $k$ の時系列パターンに接続することにより，長さ $k+1$ の時系列パターンを発見する．この処理を再帰的に行うことにより，時系列パターン長が伸び，処理の進行とともに投影時系列データベースが小さくなるため，効率的に支持度の値を調べることが可能となる．また，投影時系列データベースは時系列パターンの出現位置を記録することにより得られる．

例として，2 章 2 節の例題を用いて，時系列パターン A $\rightarrow$ B の投影時系列データベースを図 3.6 に示す．“ $\downarrow$ ”より後ろの範囲が投影時系列データベースとなり，この範囲内のアイテムの支持度を調べる．

# 6. FSPM

Wang らは時系列データベースのスキャンコストを減らすことにより処理性能を改善させる FSPM アルゴリズム [16] の提案を行っている．

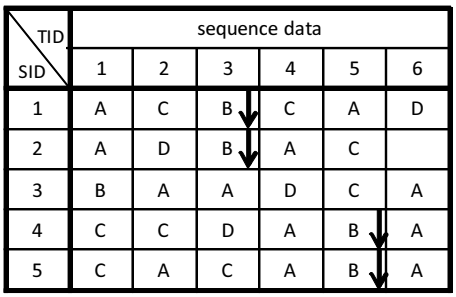



図 3.6 時系列パターン A → B の投影時系列データベースの例

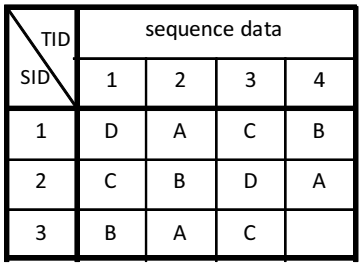



図 3.7 FSPM の例題に用いる時系列データベース

PrefixSpan が様々なアイテムにより構築される時系列データベース上で時系列パターンの発見処理を行う際に，それぞれの時系列パターンの処理で同じアイテムがスキャンされることに着目し，最小支持度を満たすアイテムごとに長さ $k+1$ の時系列パターンの発見処理を分け，その中から一つの最小支持度を満たすアイテムを処理対象として選択して，そのアイテムを含む全ての時系列パターンを抽出する．これにより，次のアイテムを対象に処理すると，前の処理で処理対象とされたアイテムを時系列データベースの探索範囲から外して処理することができ，探索範囲を減らすことができる．また，時系列データベースの探索効率を向上させるため，時系列データベース上の長さ 1 の時系列パターンをポインタのリストを用いてリンクしたデータ構造を利用している．

最小支持度 (回数) を 2 とし，図 3.7 の時系列データベースを対象としたとき，時系列パターン A を含む時系列パターンの抽出処理の例を図 3.8 を用いて説明

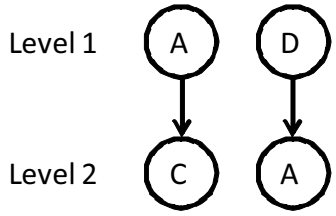



図 3.8 FSPM による時系列パターン A を含む時系列パターンの例

する．時系列データベース上で時系列パターン A が出現するまでの範囲 (SID 1 <DA>, SID 2 <CBDA>, SID 3 <BA>) で最小支持度を満たすアイテムを調べる．結果として時系列パターン A, D が発見され，これらを長さ 1 の時系列パターン (Level 1) とする．時系列パターン A にアクセスし，時系列パターン A が含まれているため，時系列データベース上で時系列パターン A より後の範囲 (SID 1 <CB>, SID 3 <C>) で最小支持度を満たすアイテムを調べ，時系列パターン A $\rightarrow$ C を発見する．次に時系列パターン D にアクセスし，時系列パターン A がまだ含まれていないため，D より後の A までの範囲 (SID 1 <DA>, SID 2 <DA>) から最小支持度を満たすアイテムを調べ，時系列パターン D $\rightarrow$ A が発見される．次の長さ 1 の時系列パターンを含む時系列パターンの発見処理に移行した際は探索範囲から A を除外する．

# 7. LAPIN

Yang らは，長さ $k+1$ の時系列パターンの $k+1$ 番目のアイテムとなるか判断するために，時系列データ上のアイテムの最終出現位置を利用する時系列パターンマイニングアルゴリズム LAPIN [19] の提案を行った．

LAPIN では長さ $k$ の時系列パターンより後に出現するアイテムの支持度を調べる手法として，長さ 1 のアイテムの最終出現位置を時系列データの TID 毎に bit vector で表した ITEM_IS_EXIST_TABLE を用いて，長さ $k$ の時系列パターンの TID に一致する bit vecter を調べる LAPIN_LCI と長さ 1 のアイテムの最終出現位置を記録した item-last-position list を用いて，長さ $k$ の時系列パターンより

後に出現するアイテムを調べる LAPIN_Suffix の二つのアプローチが提案されている．

item-last-position list を用いる点で LAPIN_Suffix が PAID に最も近いアルゴリズムと考えられる．LAPIN と PAID の違いとしては，長さ $k$ の時系列パターンより後の範囲で出現するアイテムをカウントするのと出現しないアイテムを差し引くといった点である．

# 第4章　時系列パターンマイニングアルゴリズム CC-PAID

## 1. 緒言

我々は，PAID アルゴリズムの CPU キャッシュの利用効率に着目し，時間的局所性を向上させる CC-PAID アルゴリズムの提案を行った [27].

相関ルールマイニングの分野では，処理速度を向上させるためにアルゴリズムを変更することなく，特定のデータ構造の時間的，空間的局所性を向上させるアプローチが行われた [8]．時系列パターンマイニングは相関ルールマイニングと密接な関係があり，アルゴリズムの特性にも似た面がある．そのため，局所性を考慮したアプローチは時系列パターンマイニングでも同様に有効である可能性が高いと考えた.

時系列パターンマイニングのアルゴリズムの特性として，全ての時系列パターンが発見されるまで，時系列データベース全体に対して非常に多くの再帰的なアクセスが発生するといった傾向がある．一般に，CPU キャッシュは非常に高速だがサイズは非常に小さく，大規模な時系列データベースを対象とした場合，最も長い期間参照されなかったデータを置き換えるよう指定するキャッシュの置き換えポリシーとして LRU (Least Recently Used) [7] を用いて考えると，再帰的な処理によってキャッシュにアクセスした際に必要なデータが残っている可能性は低いと考えられ，多くのキャッシュミスによるレイテンシは時系列パターンマイニングの処理でボトルネックとなりえる.

図 4.1 に，アルゴリズムを特定せず時系列データベースにアクセスする際のキャッシュミスの発生例を示す．時系列データベース全体にアクセスするようなアルゴリズムの場合，例えば SID 1 のデータを要求し，CPU キャッシュに存在し

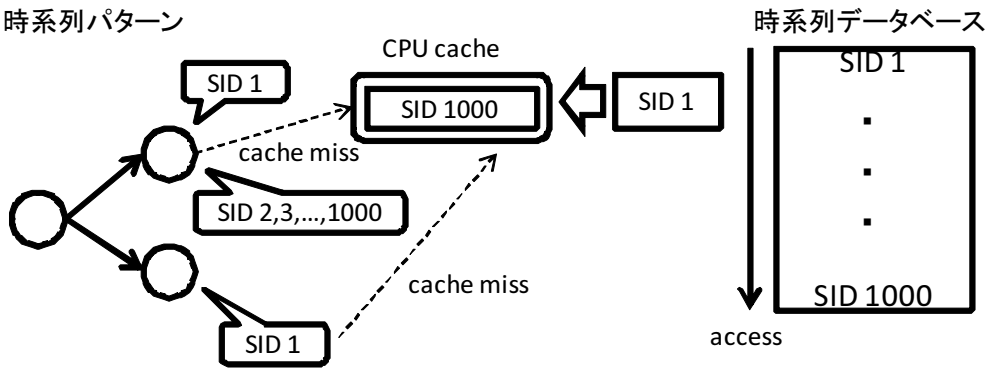



図 4.1 時系列パターンマイニングによるキャッシュミスの一例

ない場合にキャッシュミスが発生する．続けて，他の SID のデータにアクセスすると SID 1 のデータは他のデータに置き換えられ，次の時系列パターンの処理で再び SID 1 のデータを要求してキャッシュミスが発生するといった可能性が考えられる．ただし，PAID アルゴリズムは，長さ $k$ の時系列パターンを処理したとき，接頭辞となる長さ $k-1$ の時系列パターンが時系列データ上に出現しない場合，その時系列データに関する処理を避けることが可能である (2 章 3.3 節) ため，必ずしも全ての時系列データを処理する必要は無い．

我々は，PAID アルゴリズムの CPU キャッシュの利用効率をアクセスパターンの面から考察し，CPU キャッシュの利用効率を向上させるために Common-Prefix-At-A-Time といった時間的局所性の向上を狙った提案を行った．

## 2. PAID のキャッシュ利用効率についての考察

我々は，PAID アルゴリズムのアクセスパターンによる ILP-list の CPU キャッシュの利用効率に着目し，キャッシュミスが発生する可能性のあるデータ構造の一つとして考察を行った．

ILP-list は時系列パターン毎に支持度の更新処理を行うために多くの再帰的なアクセスが生じるデータ構造である．PAID では，長さ $k+1$ の時系列パターンを発見する処理に移行する際に，前の処理で発見された長さ $k$ の時系列パターン

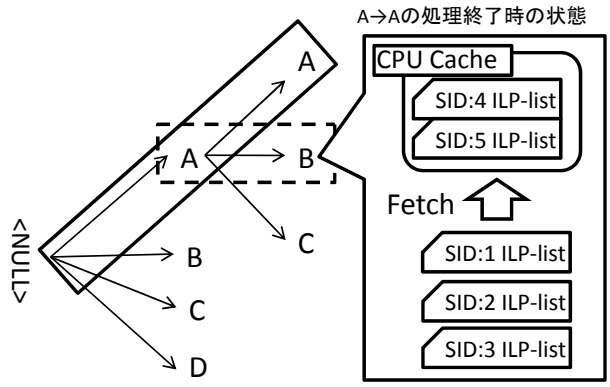



図 4.2 PAID における ILP-list のキャッシュミス

の中から深さ優先探索を用いて 1 つの長さ $k$ の時系列パターンを選択する (2 章 3.1 節)．このようなアクセスパターンを用いた場合，ILP-list に関するキャッシュミスが発生する可能性が考えられる．

PAID のアクセスパターンによる ILP-list のキャッシュミスの原因と利用効率についての考察を行う．

**ILP-list のキャッシュミス**

ILP-list のキャッシュミスに関する説明を例を用いて行う (図 4.2)．図 4.2 は長さ 2 の時系列パターン A → A の処理終了時の CPU キャッシュの状態を表わし，破線は現在の処理対象となっていることを示す．この例では時系列パターン A → A の処理終了時に CPU キャッシュに SID 4，5 の ILP-list しか存在していない．このため，時系列パターン A → B の処理開始時に SID 1，2，3 をフェッチする必要があり，これがキャッシュミスの原因の一つとなる．

時系列パターンマイニングでは，基本的に SID に対応するデータに対して昇順にアクセスされるため，SID の番号が小さいと早くに CPU キャッシュにフェッチされる可能性が高い．このため，規模の大きいデータベース（例としては時系列データ数が非常に多い等）の処理では SID の番号が小さい ILP-list はキャッシュから外れてしまう可能性が高くなるため，キャッシュミスが増加しやすいと考えられる．

**ILP-list の利用効率**

図 4.3 を用いて ILP-list の利用効率に関する説明を行う．図 4.3 では，2 章 3.4 節の例を用い，長さ 3 の時系列パターン A → B → A の処理と長さ 2 の時系列パターン A → C の処理を表わしている．時系列パターン A → B の処理では，図 2.6 の position list データベースから SID 3 の $k-1$-pbp (時系列パターン A → B の TID) は存在しないことがわかり，支持度の更新が必要ないため，SID 3 の ILP-list にはアクセスが生じない．このため，時系列パターン A → B を処理した時の SID 3 の ILP-list が CPU キャッシュに存在していても時系列パターン A → B → A の処理時に再利用されない．それに対し，時系列パターン A → C では，図 2.6 の SID 3 から $k-1$-pbp (時系列パターン A の TID) は 2 であるために，ILP-list 上での支持度の更新のためにアクセスが生じる．

ここで，時系列パターン A → B → A の処理終了時に SID 3 の ILP-list がキャッシュから外れていた場合を考えると，時系列パターン A → C の処理で再び SID 3 の ILP-list をフェッチする必要がでてくる．このため，PAID の従来のアクセスパターンでは CPU キャッシュに存在する ILP-List を効率的に再利用されない可能性がある．また，処理する時系列パターン長が長くなるほど，処理対象から外れる ILP-list が増える可能性があるため，このような問題が起こりやすくなると考えられる．

ILP-list は長さ $k+1$ の時系列パターンの発見処理毎に，$k$-projILPDB 内の支持度の値を調べるための非常に多くの再帰的なアクセスが生じるデータ構造であると考えられ，ILP-list に関連したキャッシュミスの発生によるレイテンシは処理時間を大きく増やす可能性がある．これらの問題は CPU キャッシュにフェッチされたデータがすぐに再利用されないために起こりうると考えられ，従来のアクセスパターンでは時間的局所性が低いと考えられる．このため，時間的局所性を向上させる手法として Common-Prefix-At-A-Time を提案し，それを PAID に適用した時系列パターンマイニングアルゴリズム CC-PAID の提案を行った．

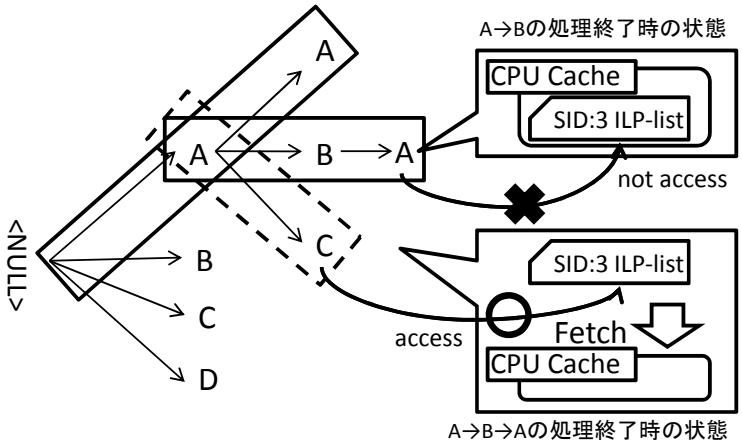



図 4.3 PAID における ILP-list の利用効率

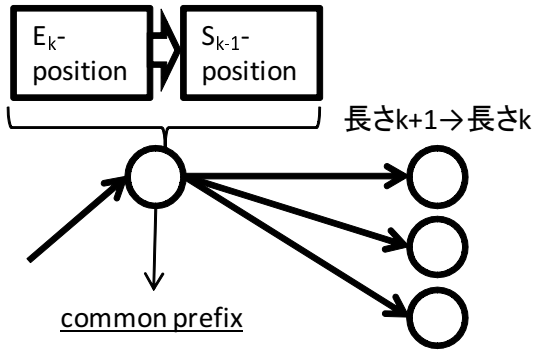



図 4.4 common prefix と $S_{k-1}$-position の関係

# 3. Common-Prefix-At-A-Time

PAID アルゴリズムでは，一つの長さ $k$ の時系列パターンから複数の長さ $k+1$ のパターンが発見される場合がある．このため，発見された長さ $k+1$ の時系列パターンは長さ $k$ の共通する接頭辞を持つ．そして，深さ優先探索により，長さ $k+1$ の時系列パターンの一つを長さ $k$ の時系列パターンとして次の長さ $k+1$ の時系列パターンの発見処理に用いる．また，支持度の更新に用いられる ILP-list の探索開始範囲も前の処理で得た $E_k$-position を $S_{k-1}$-position として利用される．そのため，長さ $k-1$ の時系列パターンである共通する接頭辞（$common\ prefix$）を持つ長さ $k$ の時系列パターンは ILP-list の処理範囲が同じになる (図 4.4)．また，$k-1$-pbp は $common\ prefix$ の TID であるため，$k-1$-pbp が NULL でないかどうかによって時系列データが処理対象となるかを判断することができるために処理する ILP-list の SID が同じになる．

$common\ prefix$ を含む時系列パターンの例として，図 2.2 の長さ 2 の時系列パターンでは A $\rightarrow$ A，A$\rightarrow$ B，A $\rightarrow$ C が $common\ prefix$ A を含む時系列パターンであり，これらの時系列パターンの $(k-1)$-pbp は $common\ prefix$ A の TID となる．

Common-Prefix-At-A-Time は $common\ prefix$ を含む複数の長さ $k$ の時系列パターンを時系列データ毎に一括して処理する手法である．PAID が一つの長さ $k$ の時系列パターンを選択するのに対して，Common-Prefix-At-A-Time では $common\ prefix$ を含む複数の長さ $k$ の時系列パターンを長さ $k+1$ の時系列パターンの発見処理の対象とする．また，次の $k+1$ の時系列パターの発見処理への移行では，深さ優先探索により一つの長さ $k$ の時系列パターンが $common\ prefix$ として選択される．

例として，図 2.4 と同様の条件で $common\ prefix$ が A である長さ 2 の時系列パターンの処理が終了した際の Common-Prefix-At-A-Time の例を図 4.5 に示す．まず長さ 1 の時系列パターンから長さ 2 の時系列パターンを発見するために，深さ優先探索により NULL を選択し，長さ k の時系列パターンとして長さ 1 の時系列パターン A, B, C, D を一括して処理する．結果としてそれぞれの時系列パターンから図 4.5 に示す長さ 2 の時系列パターン A $\rightarrow$ A，A$\rightarrow$ B，A $\rightarrow$ C，B$\rightarrow$ A，

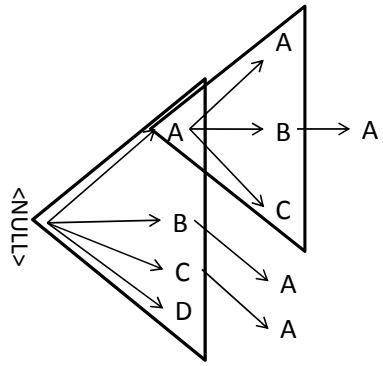



図 4.5 Common-Prefix-At-A-Time の例

C → A が発見される．続けて次の長さ $k+1$ の時系列パターンの発見処理に移行し，深さ優先探索により *common prefix* として A を選択し，長さ 2 の時系列パターン A → A，A→ B，A → C を一括して処理する．

我々は Common-Prefix-At-A-Time を用いて PAID の時間的局所性を向上させるため，CC-PAID アルゴリズムの提案を行った．

# 4. CC-PAID の処理の流れ

CC-PAID の処理の流れを図 4.6 を用いて説明する．CC-PAID では PAID のアルゴリズムの中核は変更していないため，PAID 同様に処理は二つに分けられる．

まず，*common prefix* を含む複数の長さ $k$ の時系列パターンの $k$-pbp の取得処理を行う．処理対象の SID と各長さ $k$ の時系列パターンに対応する item position list から $k-1$-pbp となる *common prefix* の TID を用いて $k$-pbp を取得する．取得した時系列パターン毎の $k$-pbp はアイテムと $k$-pbp の組としてまとめられ，それを multi-$k$-pbp とする．

次に multi-$k$-pbp 内の $k$-pbp 毎に支持度の更新処理を行う．$k$-pbp を用いて，ILP-list の $S_{k-1}$-position 以降の範囲から $E_k$-position を調べ，非出現なアイテムを決定し，それらを対象となる $k$-sup_list から差し引く．

上記の処理が全時系列データで行われた際に，各長さ $k$ の時系列パターンの

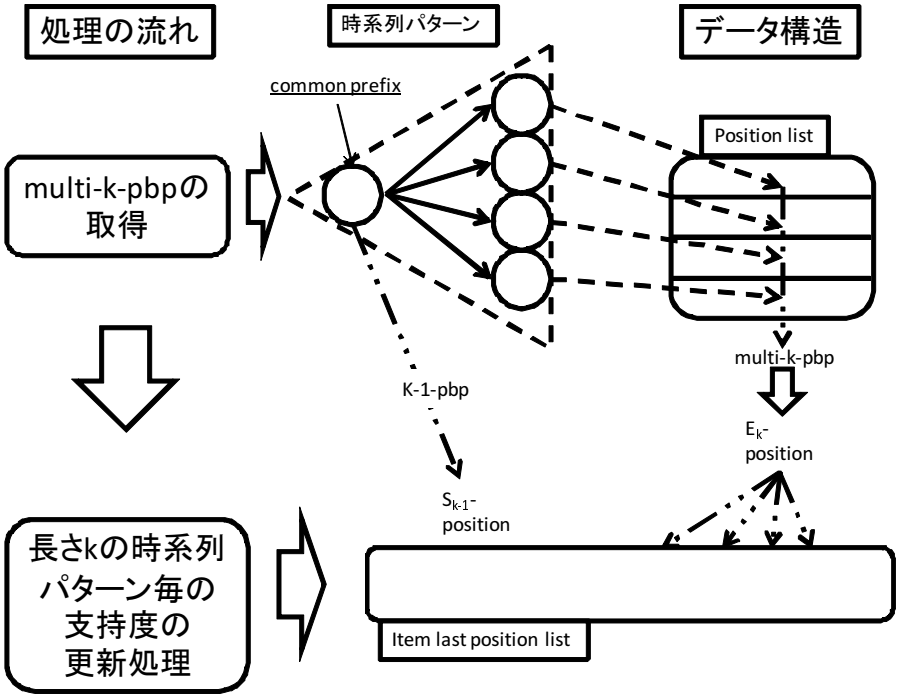



図 4.6 CC-PAID のプロセスフローとデータ構造

$k$-sup_list の支持度の値が決定され，最小支持度を満たすものを調べることにより，各々の長さ $k$ の時系列パターンが $common\ prefix$ となる複数の長さ $k+1$ の時系列パターンが発見される．次の長さ $k+1$ の時系列パターンの発見処理への移行では，深さ優先探索により $common\ prefix$ を選択し，$common\ prefix$ を含む複数の長さ $k$ の時系列パターンを次の長さ $k+1$ の時系列パターンの発見処理に用いる．これらの処理を全ての時系列パターンが発見されるまで行う．また，PAID 同様に $k-1$-pbp が NULL の場合，その時系列データに関連する処理を避けることが可能である．

# 5. CC-PAID のアルゴリズム

CC-PAID アルゴリズムを疑似コード (図 4.7) と CC-PAID による position list と ILP-list の処理例 (図 4.8 と図 4.9) を用いて説明する．この処理例は 2 章 3.4 節

```
CC-PAID Algorithm
Input : A sequence database, minimum support
Output : All sequential patterns

Main
do parallel {
1. POS-DB, ILP-DB, FI_s, sup_list ← Scan_SDB()
}
2. Call Gen_multi-(k+1)-patterns(FI_s, 0, sup_list)

Function Gen_multi-(k+1)-patterns (multi-k-sp, (k-1)-pbp_set, (k-1)-sup_list)
3. For each sequence sd do parallel{
   3.1 For each k-sp in multi-k-sp
       3.1.1 multi-k-pbp ← find_multi-k-pbp((k-1)-pbp)
   3.2 S ← find_S_pos((k-1)-pbp)
   3.3 For each k-pbp in multi-k-pbp
       3.3.1 E ← find_E_pos((k)-pbp)
       3.3.2 while(S != Null && S < E )
             3.3.2.1 (k-sp)'s k-sup_list[S.ItemID]--; //(k-sp)'s k-sup_list is a copy of (k-1)-sup_list.
             3.3.2.2 S++;
}
4. For each k-sp in multi-k-sp
   4.1 FI_s ← get_freq_item((k-sp)'s k-sup_list)
   4.2 multi-(k+1)-sp ← extend_k-sp(FI_s, k-sp)
   4.3 k-pbp_set ← get_k-pbp_set(multi-k-pbp_set)
   4.4 Call Gen_multi-(k+1)-patterns(multi-(k+1)-sp, k-pbp_set, (k-sp)'s k-sup_list)
```

図 4.7 CC-PAID アルゴリズムの疑似コード

の例と同じ条件を用いており，図 4.8 と図 4.9 は図 2.6 の SID 1 の position list と図 2.7 の SID 1 の ILP-list を *common prefix* A を選択して処理した例である．

1. (図 4.7) Step 1. 前処理として PAID と同様に処理を行う．

2. (図 4.7) Step 2. で関数 Gen_multi-(k+1)-patterns を呼び出す．CC-PAID では第一引数として複数の最小支持度を満たすアイテム (長さ 1 の時系列パターン) を渡す．

3. (図 4.7) Step 3. 各々の時系列データ毎で Step 3.3.2.2 までの以下の処理を行う．

4. (図 4.7) Step 3.1〜3.1.1 multi-$k$-sp は複数の長さ $k$ の時系列パターンであり，各々の長さ $k$ の時系列パターン毎に長さ $k$ の時系列パターンの $k$ 番目のアイテムとその $k$-pbp の組を (ItemID, TID) として，multi-$k$-pbp に挿入して

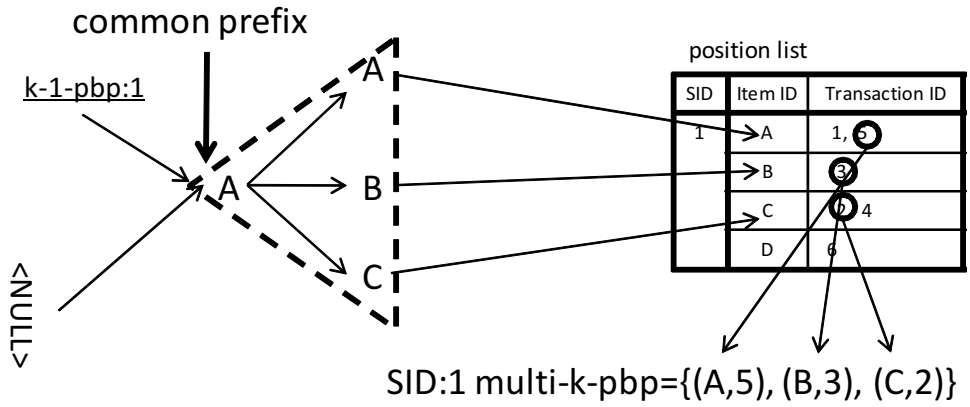



図 4.8 CC-PAID による position list の処理の例

いく．また，時系列データに対応する全ての multi-$k$-pbp をまとめたものを multi-$k$-pbp_set とする．

- 図4.8では，図2.6のSID 1のposition listから$k-1$-pbp（$common\ prefix$ A）を用いて，時系列パターン A $\rightarrow$ A，A $\rightarrow$ B，A $\rightarrow$ C のそれぞれの位置 (TID) を $k$-pbp として調べる．結果として，SID 1 の multi-$k$-pbp= $\{(A,5),\ (B,3),\ (C,2)\}$ が得られる．

5. (図 4.7) Step 3.2 PAID アルゴリズム同様の処理で $S_{k-1}$-position を取得.

   - 例において，$k-1$-pbp（$common\ prefix$ A）はTIDが1なので図4.9のILP-list上で(3,B)の位置を $S_{k-1}$-position とする．

6. (図 4.7) Step 3.3〜3.3.2.2 multi-$k$-pbp の各々の $k$-pbp 毎に以下の処理を行う．

   ILP-list 上で $k$-pbp より大きい TID の場所を探し，$E_k$-position を取得．ILP-list 上の $S_{k-1}$-position から $E_k$-position までの範囲のアイテムを非出現なアイテムとし，$k-1$-sup_list のコピーであり $k$-pbp に対応する長さ $k$ の時系列パターンの $k$-sup_list((k-sp)'s $k$-sup_list) から非出現なアイテムを差し引く．

   - 例においては，まず時系列パターン A $\rightarrow$ A の $k$-pbp 5 より大きい位置

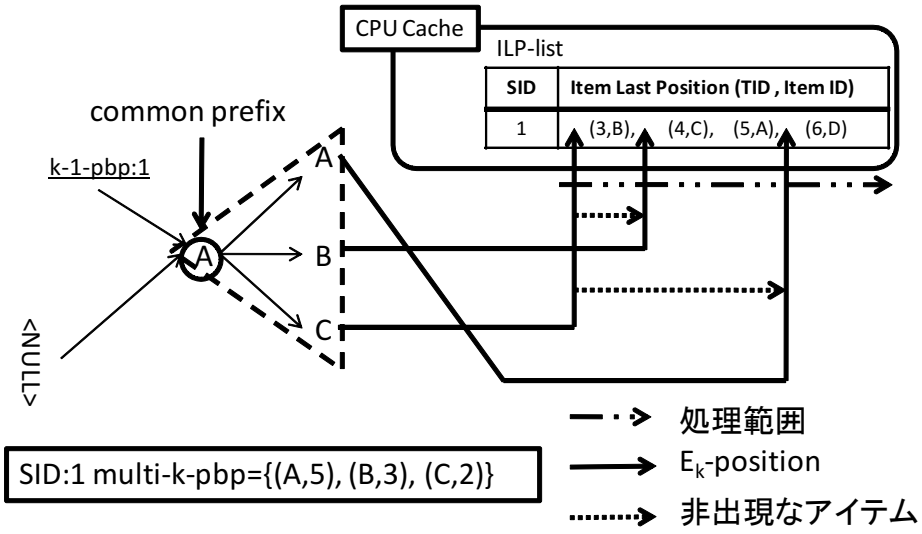



図 4.9 CC-PAID による ILP-list の処理の例

を探し，その結果 (6,D) の位置が得られ，非出現となるアイテムが A，B，C となることが分かる．時系列パターン A-sup_list をコピーした時系列パターン A $\rightarrow$ A-sup_list から A，B，C の出現回数として一つずつ差し引く．同様の処理を SID 1 の ILP-list に対し，時系列パターン A $\rightarrow$ B，A $\rightarrow$ C で行う．

7. (図 4.7) Step 4. 各々の長さ $k$ の時系列パターン毎に Step 4.4 までの以下の処理を行う．

8. (図 4.7) Step 4.1〜4.2 $k$-sup_list から最小支持度を満たすアイテム集合を取得．それらのアイテムを k-sp の後ろに連結し，$k+1$-sp とした各々のセットを multi-$(k+1)$-sp とする．

   - 例においては，時系列パターン A $\rightarrow$ B で最小支持度を満たすアイテム A が発見され，長さ $k+1$ として時系列パターン A $\rightarrow$ B $\rightarrow$ A が発見される．

9. (図 4.7) Step 4.3 multi-$k$-pbp_set から時系列データ毎の $k$-pbp を取得し，

$k$-pbp_set としてまとめる．

- 例においては，時系列パターン A $\rightarrow$ B では図 4.9 の SID 1 の multi-$k$-pbp からアイテム B を選択して 3 を取得，同様に以降の SID からも $k$-pbp を取得してまとめる．

10. (図 4.7) Step 4.4 関数 Gen_multi-(k+1)-patterns を呼び出す．

- 例においては，時系列パターン A $\rightarrow$ A では長さ $k+1$ の時系列パターンが発見されなかったので，深さ優先探索に従い，時系列パターン A $\rightarrow$ B を *common prefix* A $\rightarrow$ B として選択し，次の長さ $k+1$ の発見処理に移行する．

# 6. CC-PAID の CPU キャッシュの利用効率

CC-PAID では，Common-Prefix-At-A-Time により，支持度の更新関連の処理において，複数の *common prefix* を持つ長さ $k$ の時系列パターン間で ILP-list の処理範囲を共有することが可能となる．このため，一度長さ $k$ の時系列パターンの処理で CPU のキャッシュにフェッチされた ILP-list は，その長さ $k$ の時系列パターンの ILP-list に関連した処理が終了した後，直ぐに *common prefix* を持つ他の長さ $k$ の時系列パターンにアクセスされるため，時間的局所性が向上する．例として，図 4.9 では時系列パターン A $\rightarrow$ A によって CPU キャッシュにフェッチされた SID 1 の ILP-list が直ぐに時系列パターン A $\rightarrow$ B，A $\rightarrow$ C によってアクセスされる．また，Common-Prefix-At-A-Time による ILP-list の処理は時系列データ数が膨大な場合でも，時系列データ毎に複数の長さ $k$ の時系列パターンで処理されるため，時間的局所性が低下するといった影響はないと考えられる．

更に，ILP-list のキャッシュミスの削減や Common-Prefix-At-A-Time により，position list 等の他のデータ構造のキャッシュ利用効率等が向上する可能性も考えられる．

# 7. CC-PAIDの並列化

時系列パターンマイニングでは，時系列データ間に依存関係がないため，支持度の更新に関する処理を時系列データ毎に分割して処理することが可能であり，CPU コア数に応じて時系列データベースを分割することによる並列化が可能である．また，大規模なデータベースでは時系列データ数が膨大になる可能性があるため，データ並列性が高いと考えられる．そこで，CC-PAID では並列化手法として分割された時系列データベース毎にスレッドの処理を分けるデータ並列 [23] を採用し，並列処理による処理時間の短縮も試みた．

CC-PAID では do parallel で示す図 4.7 Step 1. の前処理と図 4.7 Step 3. 〜3.3.2.2 の部分が並列処理可能である．PAID では図 2.9 Step 1 と図 2.9 Step 3〜3.4.2 の部分がこれに相当する．

PAID 及び CC-PAID における並列化では，時系列データベースの時系列データ数を長さ $k+1$ の時系列パターン抽出処理を行うスレッド数で水平分割 [22] し，それぞれのスレッドで処理対象とする SID の範囲を決定する．例えば，時系列データ数が 100，実行スレッド数が 2 の場合，スレッド 1 は SID 1〜50，スレッド 2 は SID 51〜100 までが割り当てられ，position list-DB 及び ILP-DB の対応する SID のデータ範囲に対してそれぞれのスレッド毎にアクセスされる．そして，各スレッドの全ての SID に対応するデータの処理終了時に長さ $k+1$ の時系列パターンが発見される．

また，各スレッドで $k$-sup_list から非出現となるアイテムの差し引き処理を行う際は，長さ $k$ の時系列パターン毎に $k$-sup_list は一つしかないため，その部分がクリティカルセクション [5] となり，排他制御が必要となる．PAID を含む多くの時期パターンマイニングアルゴリズムでは，支持度の値の更新に関係した処理が非常に多く発生する傾向にあり，それが再帰的に行われるため，$k$-sup_list に対する時系列データ毎の排他制御による待ち時間が大きくなる可能性が考えられる．

このため，実装においては，時系列データ毎の支持度の値からの差し引き処理を避けるために，各スレッドのメモリ空間に非出現となるアイテムのカウントを保持するための配列を用意し，これに対し加算を行う．スレッドの処理対象となる全ての時系列データに関して処理が終わった際，排他制御された $k$-sup_list か

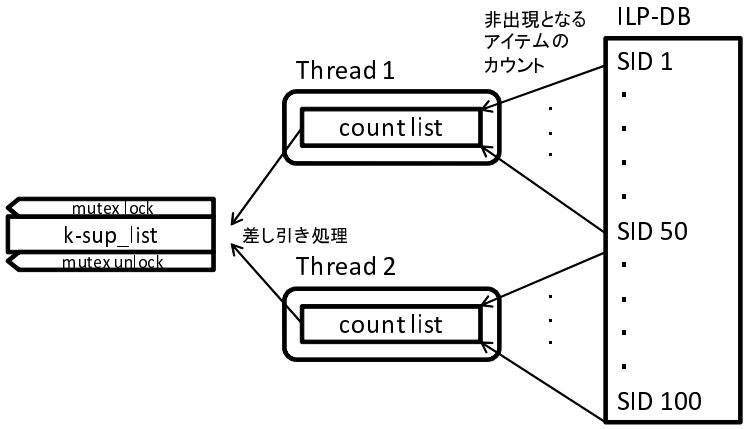



図 4.10　並列処理時の支持度の更新処理の例

ら加算された値を一度に差し引く (図 4.10)．この処理が全てのスレッドで終了した際に，$k$-sup_list の値が決まる．これにより並列処理を行った際，排他制御を極力避けることが可能となる．

# 第5章　時系列パターンマイニングアルゴリズム CCDR-PAID

## 1. 緒言

我々は，CC-PAID の計算量と CPU のキャッシュミスを削減することで処理速度を向上させる時系列パターンマイニングアルゴリズム CCDR-PAID の提案を行った [11]．CC-PAID は時系列データ毎の処理で $common\ prefix$ を持つ長さ $k$ の時系列パターンを一括して処理する Common-Prefix-At-A-Time といった手法により，特定のデータ構造の時間的局所性を向上させる手法として提案された．

CCDR-PAID では CC-PAID で行われたアクセスパターンの改良といったアプローチに対して，データ構造へのアクセスの効率化を行い，データ構造へのアクセスの無駄を省くと同時に計算量の削減も行う．

CC-PAID では，主に multi-$k$-pbp の取得処理のために利用される position list と支持度の更新の処理に利用される ILP-list といった二つのデータ構造に対してアクセスが行われる．我々は，これらのデータ構造に対するアクセス効率を考察し，データアクセスの効率化を試みた．なお，CC-PAID と同様に並列化も行われている．

## 2. position list に関するデータアクセス

CC-PAID では，position list は長さ $k$ の時系列パターン毎の prefix border position ($k$-pbp) の取得 (multi-$k$-pbp) に用いられる．また，長さ $k-1$ の時系列パターンの prefix border position ($k-1$-pbp) が NULL である場合を除いて，成立している長さ $k$ の時系列パターンに関する $k$-pbp の取得処理が行われる．

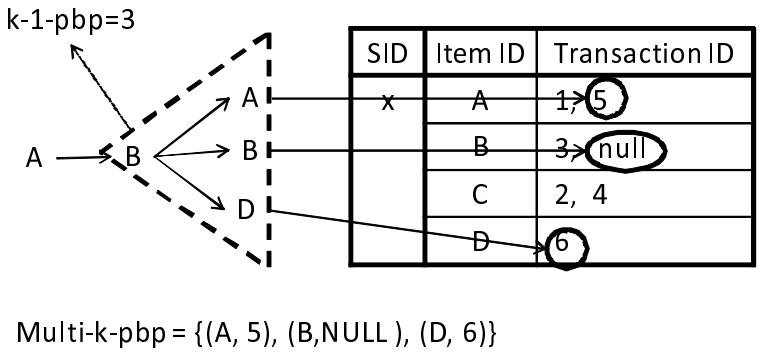



図 5.1 CC-PAID の position list へのアクセスし，NULL が取得される例

CC-PAID では，長さ $k$ の時系列パターンとしては成立しているが，時系列データにはその時系列パターンが含まれないといった場合が考えられ，同じ SID の position list からの $k$-pbp の取得結果は NULL となる．これは，時系列データ内に部分列が存在しなくても他の時系列データに部分列が存在し，最小支持度を満たすためである．そして，$k$-pbp は NULL であっても，支持度の更新処理が必要となる．

図 5.1 の例はある時系列データベースで長さ 3 の時系列パターン A $\rightarrow$ B $\rightarrow$ A, A $\rightarrow$ B $\rightarrow$ B, A $\rightarrow$ B $\rightarrow$ D が成立しており，その中の SID x の時系列データの position list の処理を行っている．時系列パターン A $\rightarrow$ B $\rightarrow$ B はアイテム B の position list で探索処理を行い, $k$-pbp として NULL を得る．これは，SID $x$ の時系列データに時系列パターン A $\rightarrow$ B $\rightarrow$ B が存在しないことを意味する。SID $x$ に時系列パターン A $\rightarrow$ B $\rightarrow$ B が存在しないということは，その時系列データ上で時系列パターン A $\rightarrow$ B $\rightarrow$ B の後に来るアイテムも存在しないため，ILP-list 上では，時系列パターン A $\rightarrow$ B の位置 ($S_{k-1}$-position) 以降のアイテムを差し引かなければならない (図 5.2)．なお，以降の例では図 5.1 と同じ時系列データベースと時系列パターンを対象とする．

ここで，時系列データ上の時系列パターンの出現について考えると，出現した場合は $k$-pbp は様々な TID の値となるが，出現しない場合は NULL として一意に決めことができ，自動的に ILP-list 上の $E_k$-position も決まる。

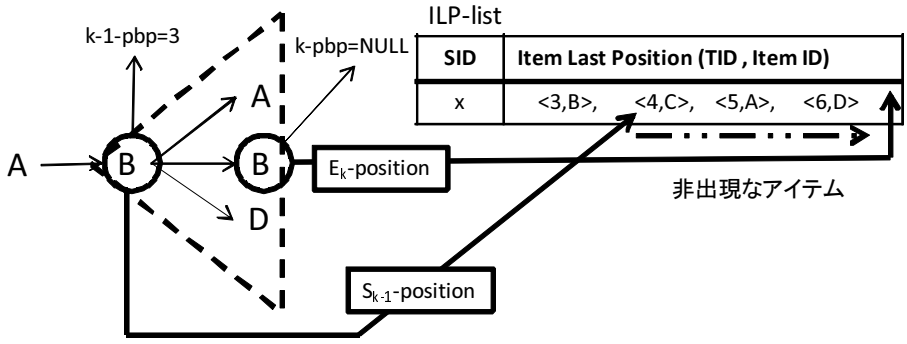



図 5.2 k-pbp が NULL の場合の ILP-list の処理の例

**データアクセスの効率化**

$k$-pbp を取得する前に，対象となっている長さ $k$ の時系列パターンが時系列データに含まれているか否かが判断できれば，含まれていない時系列パターンの $k$-pbp は NULL と決められるので，出現しない長さ $k$ の時系列パターンの $k$-pbp の処理を省略でき，position list に関連したデータの読み込みと処理を抑えることが出来ると考えられる．上記の判断を行うために CCDR-PAID では ILP-list を利用した．

ILP-list は長さ 1 の時系列パターンの最終出現位置 (TID) を TID 順にソートしたデータ構造である．そして，支持度の更新処理の際に利用される $S_{k-1}$-position 以降の範囲には時系列データ上で長さ $k-1$ の時系列パターンより後に出現するアイテムが含まれている．

これは，$S_{k-1}$-position 以降のアイテムは長さ $k$ の部分列の $k$ 番目のアイテムとして考えられる．このため，ILP-list 上の $S_{k-1}$-position 以降の範囲を調べれば，長さ $k$ の時系列パターンがその時系列データに出現するかが判断可能と考えられる．

例として，図 5.2 の $S_{k-1}$-position 以降にアイテム B は存在しない．この範囲内から，時系列パターン A $\rightarrow$ B $\rightarrow$ A, A $\rightarrow$ B $\rightarrow$ B, A $\rightarrow$ B $\rightarrow$ D のそれぞれの $k$ 番目のアイテムを調べるとアイテム A, D が得られ，その時系列データ上で時系列パターン A $\rightarrow$ B より後に出現することがわかる．それを元にアイテム A, D の position list のみにアクセスを行う．時系列パターン A $\rightarrow$ B $\rightarrow$ B は $k$-pbp

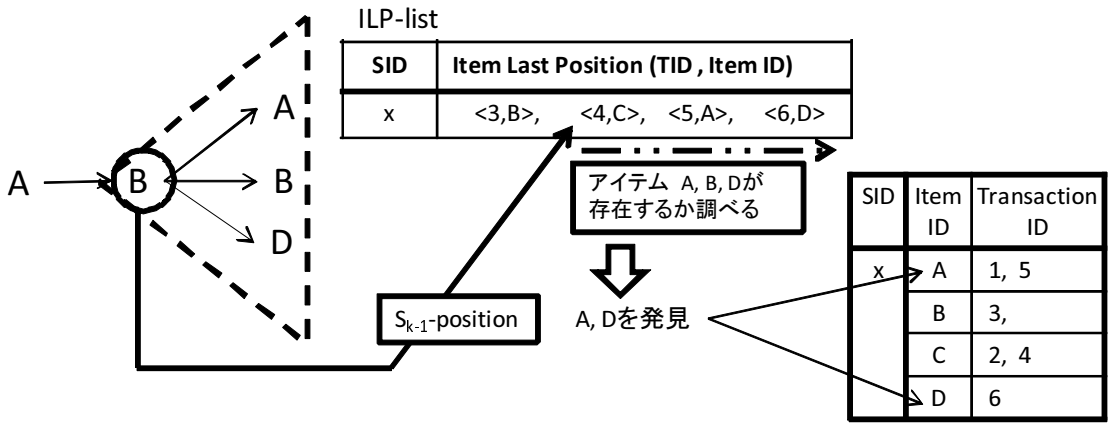



図 5.3 position list へのアクセスの回避の例

が NULL と決定されるため, position list B にアクセスしなくてよい (図 5.3)．なお，C を取得しない理由は，長さ $k$ の時系列パターンの $k$ 番目のアイテムではなく，$k$-pbp を取得する必要がないためである．

ここで，position list のアクセス効率を向上させるのに必要なデータを明確にすると，ILP-list の $S_{k-1}$-position 以降の範囲に出現する $common\ prefix$ を持つ長さ $k$ の時系列パターンの $k$ 番目のアイテムとなる．

## 3. ILP-list に関するデータアクセス

$common\ prefix$ を持つ長さ $k$ の時系列パターンから，長さ $k+1$ の時系列パターンを発見することを考えると，時系列パターンには順序関係があるために長さ $k$ の時系列パターンの $k$ 番目のアイテムとして成立していなければ，そのアイテムは既に最小支持度を満たしていないため長さ $k+1$ の時系列パターンの $k+1$ 番目のアイテムとは成りえない．このようなアイテムは時系列パターンマイニングでは支持度の更新対象から外して処理が可能である．

なお，FSPM アルゴリズム [16] では，長さ $k+1$ の時系列パターンの $k+1$ 番目のアイテムとして成立する可能性のあるアイテムを candidate-set として，それに含まれないアイテムを支持度の値の調査に利用される範囲から除外しており，こ

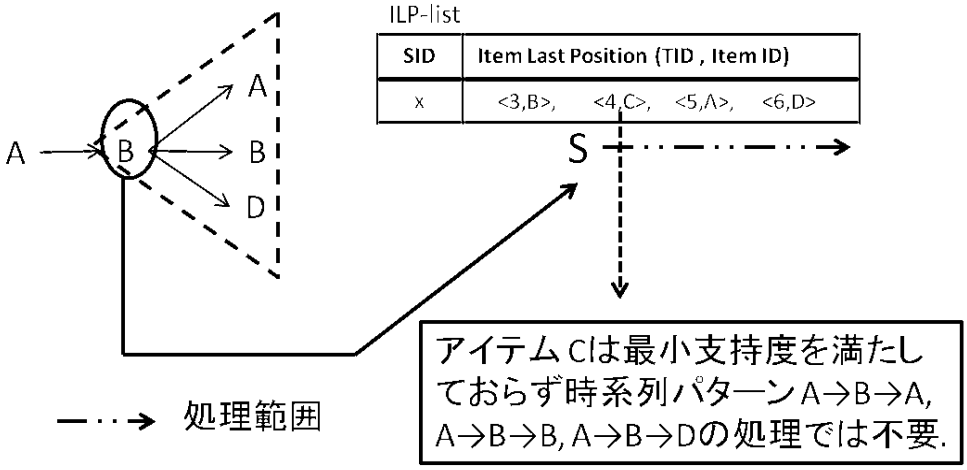



図 5.4 CC-PAID の ILP-list での処理範囲に含まれる不要な要素

れを candidate-driven として提案を行っている．ILP-list でも同様に $k+1$ 番目として成立する可能性のないアイテムを処理範囲から除外できると考えられる．

**データアクセスの効率化**

ILP-list では複数の長さ $k$ の時系列パターンにより $S_{k-1}$-position 以降の範囲から $E_k$-position の探索と非出現なアイテムの差し引き処理が行われる．$S_{k-1}$-position 以降に出現するアイテムについて考察すると，長さ $k+1$ の時系列パターンの $k+1$ 番目になる可能性の無いアイテムが含まれる可能性がある．これは長さ $k$ の時系列パターンの $k$ 番目のアイテムとして存在するか否かで判断できる．このため，提案手法では長さ $k+1$ の時系列パターンの $k+1$ 番目になる可能性の無いアイテムを不要なアイテムと考え，支持度の値の更新に関する処理から除外する．これにより，$E_k$-position の探索と非出現なアイテムの差し引き処理の計算量を削減し，同時に CPU のキャッシュに読み込まれるデータ量を少なくできると考えられる．

図 5.4 の例では，長さ 3 の時系列パターン A $\rightarrow$ B $\rightarrow$ A, A $\rightarrow$ B $\rightarrow$ B, A $\rightarrow$ B $\rightarrow$ D が成立しているため，長さ 4 の時系列パターンの 4 番目のアイテムとなる可能性があるのはアイテム A，B，D である．しかし，$S_{k-1}$-position (時系列パターン A $\rightarrow$ B の対応位置) 以降にアイテム C といった不要なアイテムが含まれてい

る．このため，$E_k$-position の探索や支持度の値からの差し引き処理で，このような不要なアイテムにアクセスする可能性がある．

不要なアクセスを発生させないために必要なデータを明確にすると，5 章 2 節と同じように ILP-list の $S_{k-1}$-position 以降の範囲に出現する $common\ prefix$ を持つ長さ $k$ の時系列パターンの $k$ 番目のアイテムとなる．

# 4. ILP-list の再構築

必要なデータへのアクセスを行う場合，ポインタを用いたアクセス等が考えられるが，我々は，キャッシュ指向により必要なデータへのアクセスを行うために，必要なデータのみを複製した新たな ILP-list の構築を行う．

データの複製はメモリ消費が大きくなる可能性があるが，CPU キャッシュについて考えると，必要なデータのアドレスが近くなり，それにアクセスした際に CPU キャッシュに取り込まれるデータサイズは小さくなると考えられる．また，データのアドレスが近くなることにより，プリフェッチされる可能性が高くなり，空間的局所性の向上にも繋がると考えられる．しかし，multi-$k$-pbp の取得と支持度の更新関連の処理で，別々に必要なデータのみにアクセスして再構築を行う場合，再構築にかかるコストが大きくなる可能性がある．

これに対して，5 章 2 節と 5 章 3 節から，position list と ILP-list のデータアクセスの効率化には，ILP-list の $S_{k-1}$-position 以降の範囲に出現する共通する長さ $k$ の時系列パターンの $k$ 番目のアイテムが必要となり，必要なデータが共通していることがわかる．このため，CCDR-PAID では上記二点での処理が行われる前にあらかじめ ILP-list から必要なデータを取り出して，ILP-list の再構築を行う (図 5.6)．

また，再構築された ILP-list は次の長さ $k+1$ の時系列パターンの発見処理で利用され，そこから更に再構築を行い，長さ $k+1$ の時系列パターンの発見処理毎に毎回動的に再構築される．そのため，図 5.5 に示すように，ILP-list の長さは処理の進行と共にデータ長が短くなる可能性がある．

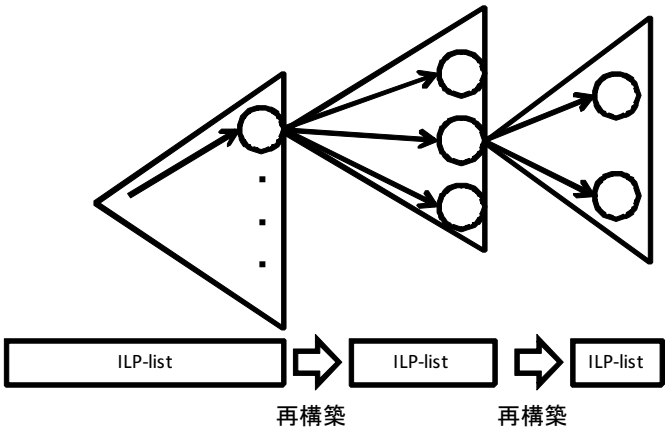



図 5.5 ILP-list の再構築によるデータ長の縮小

# 5. CCDR-PAID の処理の流れ

CCDR-PAID による支持度更新までの処理の流れ (図 5.6) を例を用いて示す.

1. ILP-list の再構築を行う.
   $S_{k-1}$-position 以降の長さ $k$ の時系列パターンの k 番目として成立しているアイテムとその TID を全て抜き出し，それらのデータをコピーすることにより ILP-list の再構築を行う.

   例 (図 5.7) として，長さ 3 の時系列パターン A → B → A, A → B → B, A → B → D があり, *common prefix* である時系列パターン A → B の $k-1$-pbp が 3 のとき，$S_{k-1}$-position は <4, C> からであり，それ以降からアイテム A，B，D と一致する要素を探す．結果として，抜き出されるアイテムと TID は <5, A>，<6, D> となり，これが再構築された ILP-list の要素となる.

2. 再構築された ILP-list に含まれるアイテムに対応する position list へアクセスし，multi-$k$-pbp を取得.
   図 5.8 では，再構築された ILP-list にはアイテム A と D が含まれているのでそれらの position list にアクセスし，$k$-pbp を取得する．CC-PAID では

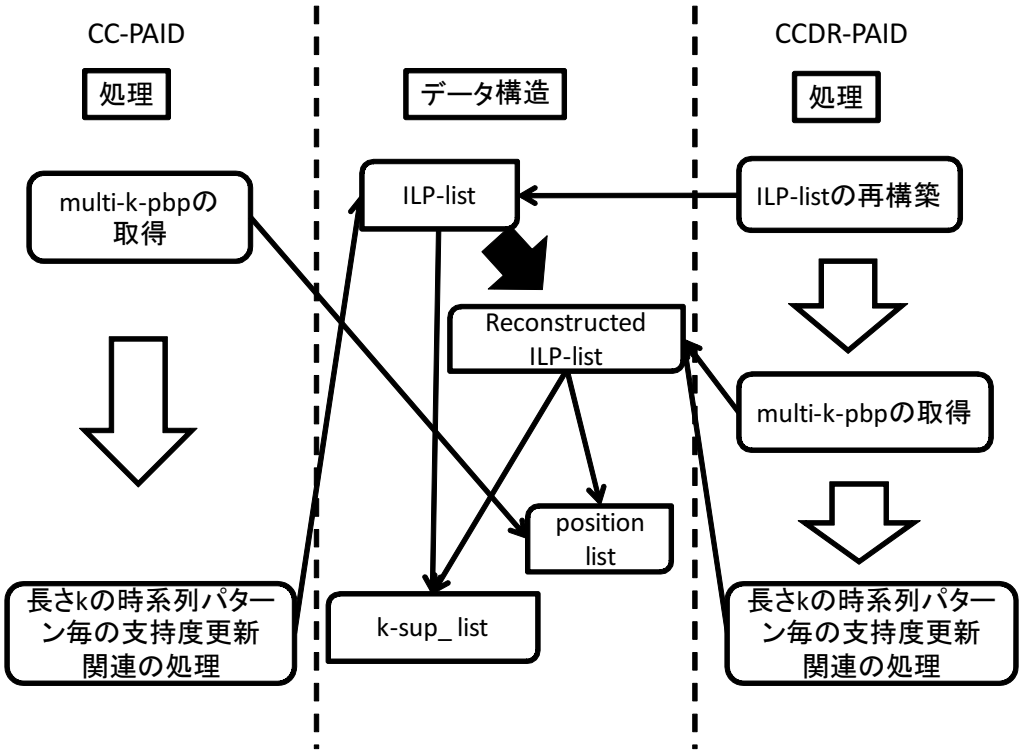



図 5.6 CC-PAID と CCDR-PAID の処理とデータ構造

アイテム B の position list が処理されるが CCDR-PAID ではその処理は省かれる．

3. 再構築した ILP-list にアクセスし，長さ $k$ の時系列パターンのそれぞれで CC-PAID と同様に支持度の更新に関係する処理を行う．
   再構築された ILP-list の処理範囲となるデータは <5, A>，<6, D> の二つである（図 5.9）．CC-PAID では $S_{k-1}$-position 以降を考えると <4,C> を含む三つの要素が処理範囲となるので，再構築された ILP-list は処理範囲を縮めることが可能である．なお，再構築した ILP-list は次の長さ $k+1$ の時系列パターンの発見処理で再利用される．

CCDR-PAID では，再構築された ILP-list を用いて，multi-$k$-pbp の処理では省略が可能な position list へのアクセスを省略すると共に $k$-pbp の取得処理も省略し，支持度の更新処理では不要なアイテムへのアクセスを避けると同時に $E_k$-position

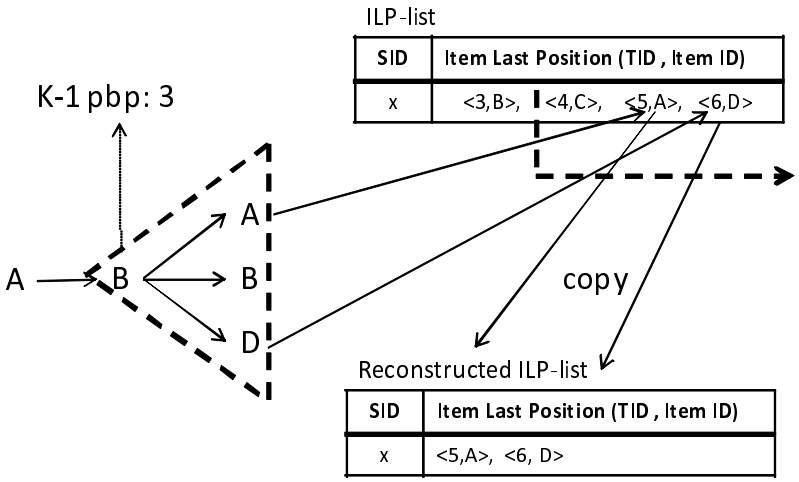



図 5.7 ILP-list の再構築の例

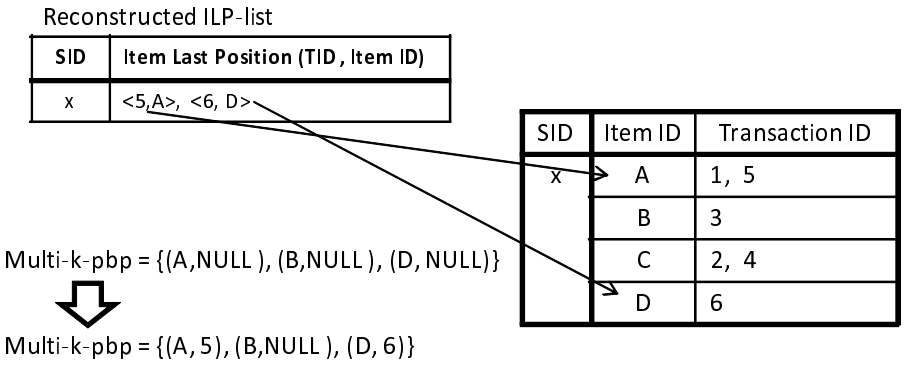



図 5.8 再構築された ILP-list を用いた multi-$k$-pbp の取得の例

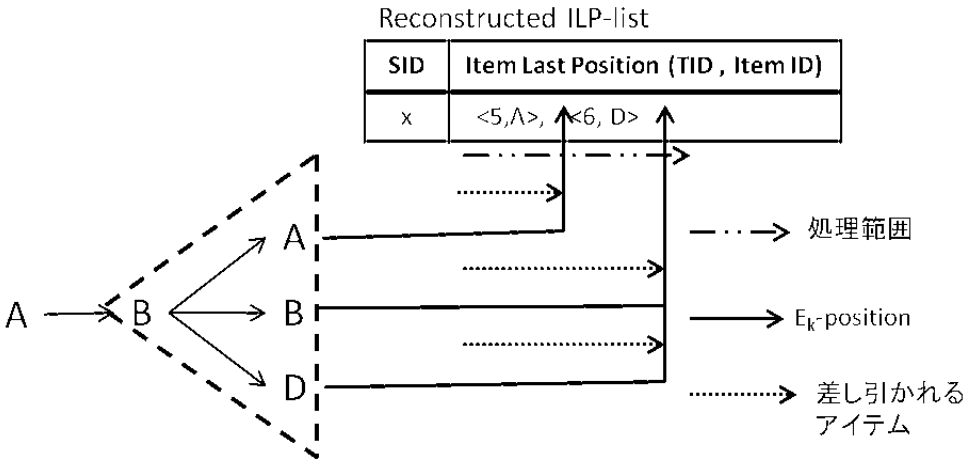



図 5.9 再構築された ILP-list を用いた支持度更新の関連処理の例

の探索や支持度の値からの差し引き処理を少なくすることが可能となる.

# 6. CCDR-PAID のアルゴリズム

図 5.10 の疑似コードを用いて，CCDR-PAID のアルゴリズムの説明を行う.

1. Step 1. 前処理として PAID と同様に処理を行う.
2. Step 2. で関数 Gen_multi-$(k+1)$-patterns を呼び出す．CCDR-PAID では引数に ILP-DB を渡す.
3. Step 3. 各々の時系列データ毎で Step 3.3.3.2. までの以下の処理を行う.
4. Step 3.1. ILP-list の再構築を行う.
   ILP-list の $S_{k-1}$-position 以降の範囲から長さ $k$ の時系列パターンの $k$ 番目のアイテムと TID を調べ，ILP-list を再構築する.
5. Step 3.2.〜3.2.1. 再構築された ILP-list を用いて multi-$k$-pbp を取得.
   再構築された ILP-list 内のアイテムに対応する position list にアクセスし，$k$-pbp を取得.

```
CCDR-PAID Algorithm
Input : A sequence database, minimum support
Output : All sequential patterns

Main
do parallel {
1. POS-DB, ILP-DB, FI_s, sup_list ← Scan_SDB()
2. Call Gen_multi-(k+1)-patterns(FI_s, 0, sup_list, ILP-DB)
}

Function Gen_multi-(k+1)-patterns (multi-k-sp, (k-1)-pbp_set, (k-1)-sup_list, ILP-DB)
3. For each sequence sd do parallel {
   3.1. reconstructed_ILP-list ← reconstruct_ILPlist(multi-k-sp, ILP-list)
   3.2. For each item in reconstructed_ILP-list
        3.2.1. multi-k-pbp ← find_multi-k-pbp(k-1-pbp)
   3.3. For each k-pbp in multi-k-pbp
        3.3.1. S=0
        3.3.2. E ← find_E_pos((k)-pbp, reconstructed_ILP-list)
        3.3.3. while(S < E )
                3.3.3.1. (k-sp)'s k-sup_list[S.ItemID]--; //(k-sp)'s k-sup_list is a copy of (k-1)-sup_list.
                3.3.3.2. S++;
}
4. For each k-sp in multi-k-sp
   4.1 FI_s ← get_freq_item((k-sp)'s k-sup_list)
   4.2 multi-(k+1)-sp ← extend_k-sp(FI_s, k-sp)
   4.3 k-pbp_set ← get_k-pbp_set(multi-k-pbp_set)
   4.4 Call Gen_multi-(k+1)-patterns(multi-(k+1)-sp, k-pbp_set, (k-sp)'s k-sup_list, reconstructed_ILP-DB)
```

図 5.10 CCDR-PAID の疑似コード

6. Step 3.3. 各々の $k$-pbp 毎に 3.3.1〜3.3.3.2 までの処理を行う.

7. Step 3.3.1 $S_{k-1}$-position を再構築された ILP-list の先頭に設定.
   再構築された ILP-list には ILP-list の $S_{k-1}$-position 以降のアイテムと TID が含まれているため.

8. 3.3.2.〜3.3.3.2. 再構築された ILP-list を用いて支持度の値の更新処理を行う.

9. 4.〜4.3. CC-PAID の処理と同様.

10. 4.4 . 関数 Gen_multi-$(k+1)$-patterns を呼び出し，再構築された ILP-list のデータベースを引数として渡す.

# 第6章　性能評価

本章では，既存の時系列パターンマイニングアルゴリズム PAID と提案手法 CC-PAID と CCDR-PAID の三つのアルゴリズムを特徴の違うデータセットを用いて，処理時間，キャッシュミス，実行命令数，メモリ使用量により評価を行った．また，並列処理時の性能評価も行った．

PAID の並列化に関しては図 2.9 Step 1. と図 2.9 Step 3.〜3.4.2 の部分を並列化したものを利用した．なお，最小支持度は割合で表され，例えば，最小支持度 0.1 としたとき，10%を意味する．また，本論文の提案手法は S-step に着目しているため，PAID, CC-PAID, CCDR-PAID の評価は S-step のみを対象とした．

データセットの読み込みから全ての時系列パターンが発見されるまでを評価範囲とし，計測時は時系列パターンの標準出力は行っていない．評価の回数は記載がない限り，1 回となっている．

## 1. 評価環境

本研究で用いた評価環境を表 6.1 に示す．

プログラムのコンパイルには g++ (GCC，version 4.4.5) を用い，記載がない限り最適化オプションとして $-O2$ を指定し，プログラムの生成を行った．CPU のキャッシュミスと実行命令数の測定には Intel VTune Amplifier XE 2011 [1] を用い，評価環境において主に最もキャッシュミスによるレイテンシの大きい Last Level Cache (LLC) として L3 キャッシュミスの見積もり (MEM_LOAD_RETIRED.LLC_MISS の合計値) と実行命令数 (INST_RETIRED.ANY の合計値) の調査を行った．

実験で用いるデータセットは IBM Almaden Research Center で公開されていた IBM Data Generator により生成を行った．これは Prefixspan を始めとする他の

表 6.1 評価環境

| | | |
|---|---|---|
| OS | Fedora 13 64bit | |
| Kernel | 2.6.34.7-61 | |
| CPU | Intel Core i7 980X | |
| | 周波数 | 3.33GHz |
| | コア数 | 6 |
| | L2 cache | 256KB x 6 |
| | L3 cache | 12MB |
| Main memory | 12GB | |

時系列パターンマイニングの研究で広く用いられているものであり，時系列データ数や平均トランザクション数といったパラメータを設定することにより様々な時系列データベースが生成可能である．

本研究では，時系列データ数，時系列データ内の平均トランザクション数，トランザクション内の平均アイテム数，アイテムの種類数を時系列データベースの生成における重要なパラメータと考え，これらの値を変更してデータセットを生成した．なお，設定可能な他のパラメータは初期値を使用した．

性能評価で使用された 5 つのデータセットの内容を表 6.2 に示す．データセット $D_1$ を基準とし，データセット $D_1$ に対して，$D_2$ は時系列データ数が二倍，$D_3$ は時系列データ内の平均トランザクション数が二倍，$D_4$ はアイテムの種類数が二倍となっている．なお，データセット $D_5$ はデータセット $D_1$ に対し，時系列データ数と時系列データ内の平均トランザクション数が二倍となっており，並列処理の評価にも用いた．

## 2. 処理時間

PAID，CC-PAID，CCDR-PAID によりデータセット $D_1$，$D_2$，$D_3$，$D_4$，$D_5$ を用い，処理時間と処理時間の内訳を調べた．また，最小支持度の変更幅を変えた際に，PAID に対する CC-PAID の処理時間の変化の割合と CC-PAID に対する

表 6.2 性能評価で利用されるデータセット

| データセット名 | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ |
|---|---|---|---|---|---|
| 時系列データ数 | 50,000 | 100,000 | 50,000 | 50,000 | 100,000 |
| 時系列データ内の平均トランザクション数 | 75 | 75 | 150 | 75 | 150 |
| トランザクション内の平均アイテム数 | 4 | 4 | 4 | 4 | 4 |
| アイテムの種類数 | 250 | 250 | 250 | 500 | 250 |
| データサイズ | 96.3MB | 192.6MB | 194MB | 96.3MB | 387.9MB |

CCDR-PAID の処理時間の変化の割合が変化する傾向に違いが生じるか確認するために最小支持度の変更幅を 0.1 と 0.02 として調べた．

## 2.1 最小支持度の変更幅を 0.1 とした場合の評価

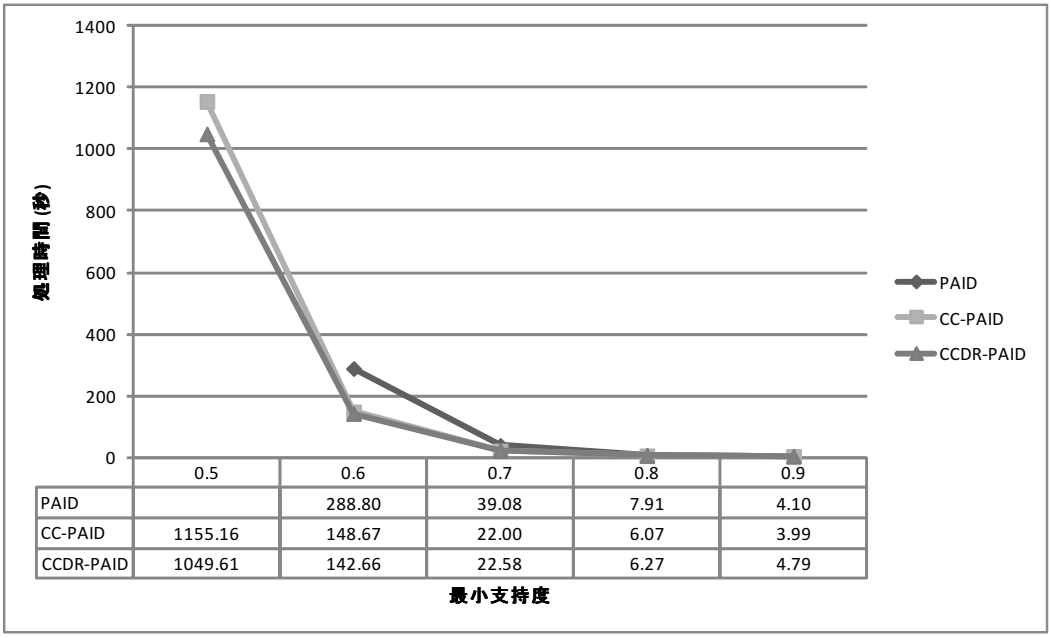
最小支持度の変更幅を 0.1 とし，最小支持度を 0.9 から 0.1 ずつ下げて 1800 秒を超えない最小支持度までの計測を行った．最小支持度の変更幅を 0.1 とすることで，最小支持度を網羅的に調べ，各アルゴリズムの処理時間の変化を確認する．

- (図 6.1) 基準となるデータセット $D_1$，最小支持度 0.5〜0.9
  - 処理時間

    CC-PAID では，PAID に対して最小支持度 0.6 の時に約 49%，0.7 の時に約 44%，0.8 の時に約 23%，0.9 の時に約 3%，処理時間を短縮した．

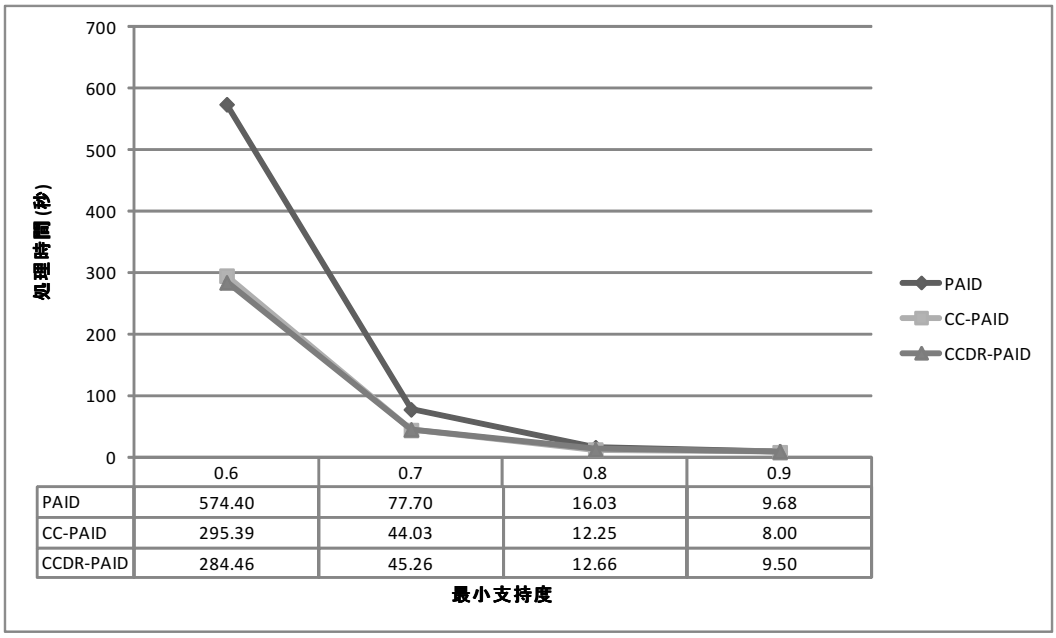
    CCDR-PAID では，CC-PAID に対して最小支持度 0.5 の時に約 9%，0.6 の時に約 4%，処理時間を短縮し，最小支持度 0.7 と 0.8 の時に約 3%，0.9 の時に約 20%，処理時間が増加した．

    また，PAID に対して CCDR-PAID は最小支持度 0.6 の時に最大で約 51%，処理時間を短縮した．




| | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|
| PAID | | 288.80 | 39.08 | 7.91 | 4.10 |
| CC-PAID | 1155.16 | 148.67 | 22.00 | 6.07 | 3.99 |
| CCDR-PAID | 1049.61 | 142.66 | 22.58 | 6.27 | 4.79 |

図 6.1 最小支持度の変更幅が 0.1 の時のデータセット $D_1$ の処理時間




| | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|
| PAID | 574.40 | 77.70 | 16.03 | 9.68 |
| CC-PAID | 295.39 | 44.03 | 12.25 | 8.00 |
| CCDR-PAID | 284.46 | 45.26 | 12.66 | 9.50 |

図 6.2 最小支持度の変更幅が 0.1 の時のデータセット $D_2$ の処理時間

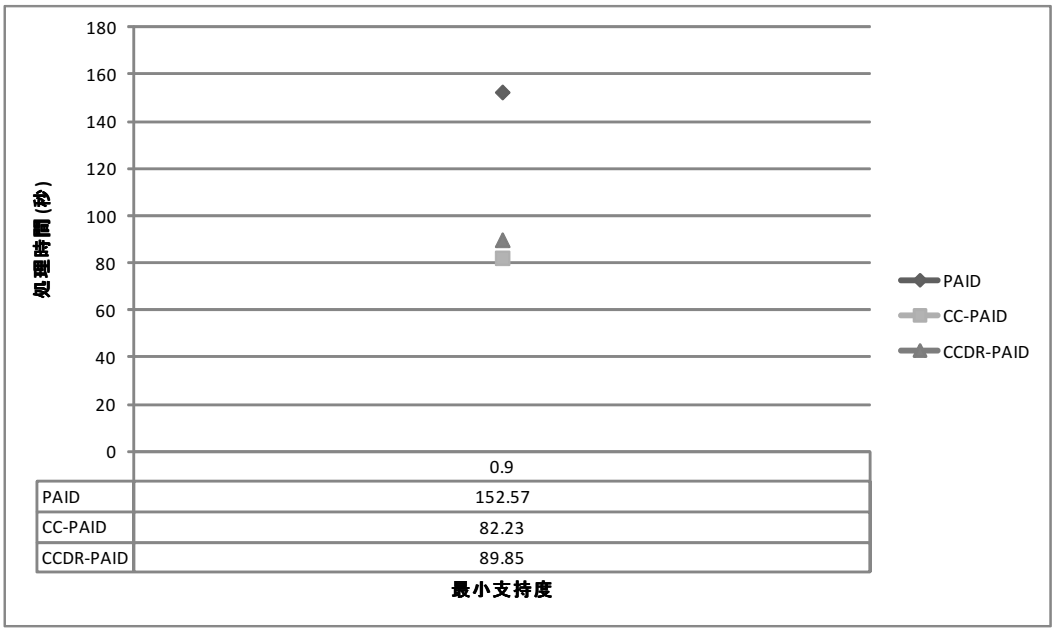



図 6.3 最小支持度の変更幅が 0.1 の時のデータセット $D_3$ の処理時間

- (図 6.2) 基準に対して時系列データ数が 2 倍のデータセット $D_2$，最小支持度 0.6〜0.9

  – 処理時間

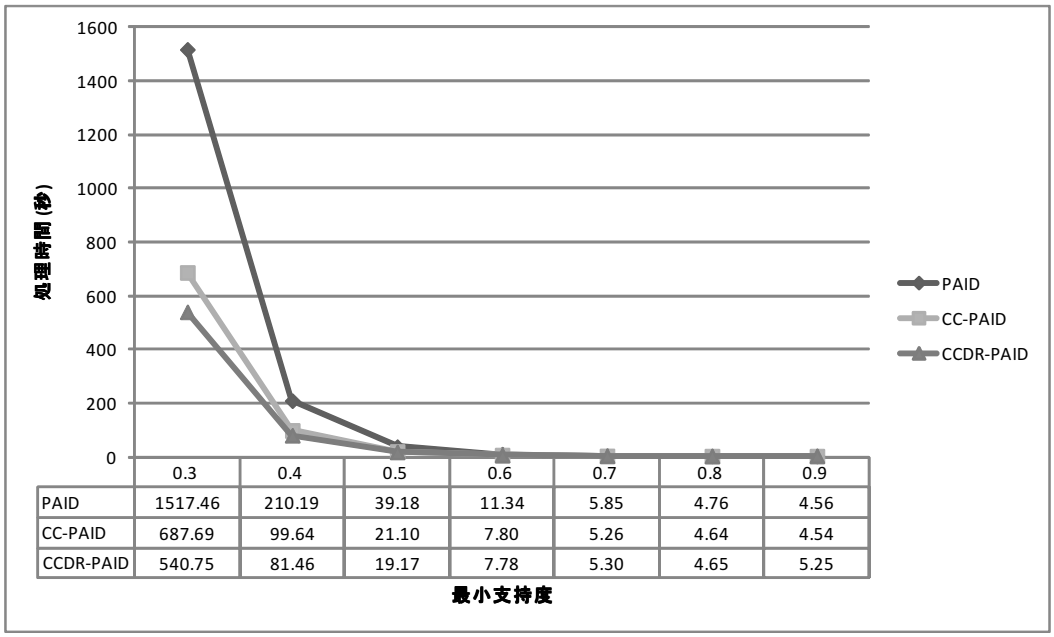
  CC-PAID では，PAID に対して最小支持度 0.6 の時に約 49%，0.7 の時に約 43%，0.8 の時に約 24%，0.9 の時に約 17%，処理時間を短縮した．

  CCDR-PAID では，CC-PAID に対して最小支持度 0.6 の時に約 4%，処理時間を短縮し，最小支持度 0.7 と 0.8 の時に約 3%，0.9 の時に約 19%，処理時間が増加した．

  また，PAID に対して CCDR-PAID は最小支持度 0.6 の時に最大で約 50%，処理時間を短縮した．

- (図 6.3) 基準に対して平均トランザクション数が 2 倍のデータセット $D_3$，最小支持度 0.9

  – 処理時間




| | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|
| PAID | 1517.46 | 210.19 | 39.18 | 11.34 | 5.85 | 4.76 | 4.56 |
| CC-PAID | 687.69 | 99.64 | 21.10 | 7.80 | 5.26 | 4.64 | 4.54 |
| CCDR-PAID | 540.75 | 81.46 | 19.17 | 7.78 | 5.30 | 4.65 | 5.25 |

図 6.4 最小支持度の変更幅が 0.1 の時のデータセット $D_4$ の処理時間

CC-PAID では，PAID に対して最小支持度 0.9 の時，約 46%，処理時間を短縮した．

CCDR-PAID では，CC-PAID に対して最小支持度 0.9 の時，約 9%，処理時間が増加した．

また，PAID に対して CCDR-PAID は最小支持度 0.9 の時に約 41%，処理時間を短縮した．

- (図 6.4) 基準に対してアイテムの種類数が 2 倍のデータセット $D_4$，最小支持度 0.3〜0.9
  - 処理時間

    CC-PAID では，PAID に対して最小支持度 0.3 の時に約 55%，0.4 の時に約 53%，0.5 の時に約 46%，0.6 の時に約 31%，0.7 の時に約 10%，0.8 の時に約 3%，0.9 の時に約 0.5%，処理時間が短縮された．

    CCDR-PAID では，CC-PAID に対して最小支持度 0.3 の時に約 21%，0.4 の時に約 18%，0.5 の時に約 9%，0.6 の時に約 0.3%，処理時間が短

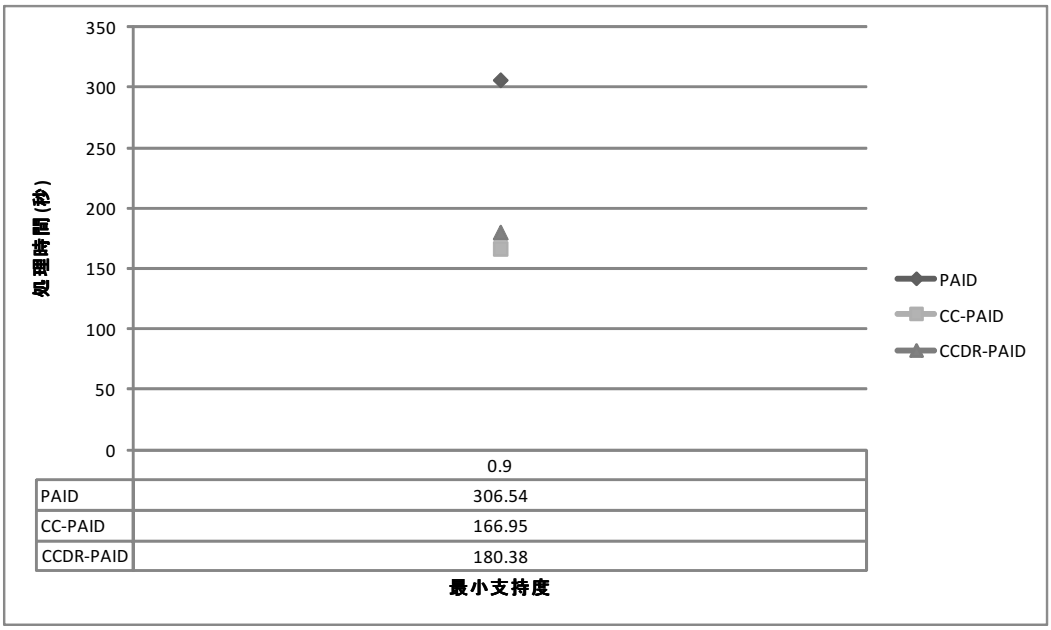



図 6.5 最小支持度の変更幅が 0.1 の時のデータセット $D_5$ の処理時間

縮され，最小支持度 0.7 の時に約 1%，0.8 の時に約 0.3%，0.9 の時に約 16%，処理時間が増加した．

また，PAID に対して CCDR-PAID は最小支持度 0.3 の時に最大で約 64%，処理時間を短縮した．

- (図 6.5) 基準に対して時系列データ数と平均トランザクション数が 2 倍のデータセット $D_5$，最小支持度 0.9
  - 処理時間

    CC-PAID では，PAID に対して最小支持度 0.9 の時に約 46%，処理時間を短縮した．

    CCDR-PAID では，CC-PAID に対して最小支持度 0.9 の時に約 8%，処理時間が増加した．

    また，PAID に対して CCDR-PAID は最小支持度 0.9 の時に最大で約 41%，処理時間を短縮した．

**考察**

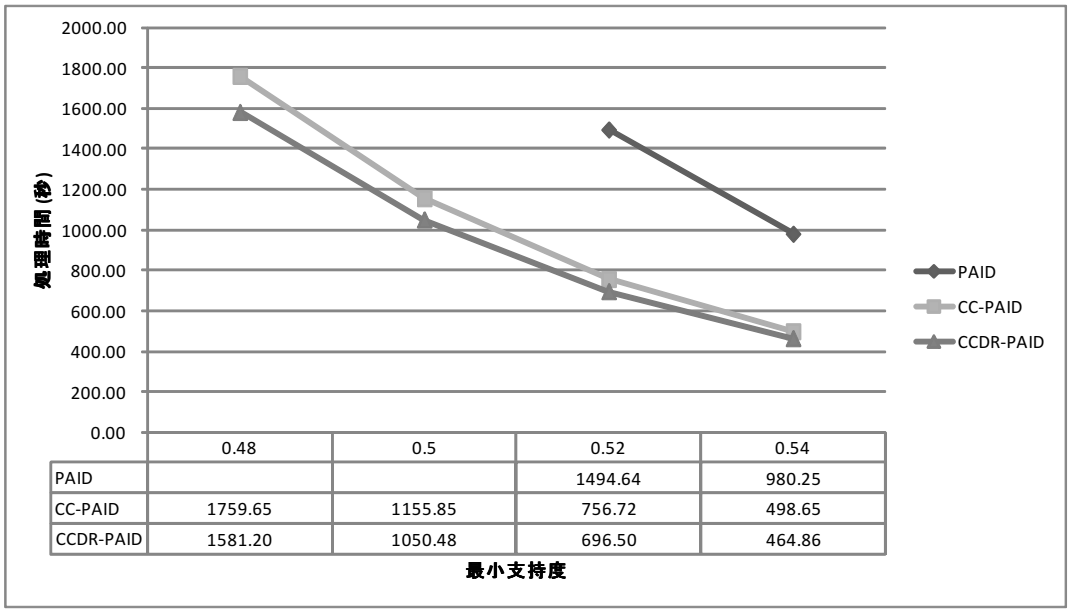
計測した全てのデータセットと最小支持度で PAID に対して，CC-PAID は処理時間が短縮され，最大で約 55%，処理時間が短縮された．CCDR-PAID に関しては，基準となるデータセット $D_1$，基準に対して時系列データ数が 2 倍のデータセット $D_2$，基準に対してアイテムの種類数が 2 倍のデータセット $D_4$ で CC-PAID に対して最小支持度が小さい場合に処理時間が短縮され，最小支持度が大きい場合に処理時間が増加し，基準に対して平均トランザクション数が 2 倍のデータセット $D_3$，基準に対して時系列データ数と平均トランザクション数が 2 倍のデータセット $D_5$ で，処理時間の増加が確認され，最大で約 21%の処理時間の短縮となった．なお，処理時間が数秒以下の場合，ほとんど時系列パターンが発見されず，position list-DB や ILP-DB の作成等を行う前処理の時間となっている可能性があり，アルゴリズム間の処理時間の差が数秒以下の場合は，誤差が生じている可能性が考えられる．

また，最小支持度を下げると CC-PAID の PAID に対する処理時間が短縮される割合と CCDR-PAID の CC-PAID に対する処理時間が短縮する割合が増加する傾向が確認できた．

## 2.2 最小支持度の変更幅を 0.02 とした場合の評価

6 章 2.1 節の最小支持度の変更幅を 0.1 とした評価結果から，データセット毎に CCDR-PAID の処理時間が 1800 秒を超えない最小支持度を選び，その最小支持度から 0.02 ずつ下げていき，CCDR-PAID が 1800 秒を超えない最小支持度を調べ，その最小支持度から 0.02 ずつ増加させた 4 つの最小支持度を用いて計測を行った．また，3 回計測して平均した．最小支持度の変更幅を 0.02 とすることで，最小支持度を詳細に調べ，各アルゴリズムの処理時間の変化を確認する．

- (図 6.6) 基準となるデータセット $D_1$，最小支持度 0.48〜0.54
  - 処理時間
    CC-PAID では，PAID に対して最小支持度 0.52 と 0.54 の時に約 49%，処理時間が短縮された．




| | 0.48 | 0.5 | 0.52 | 0.54 |
|---|---|---|---|---|
| PAID | | | 1494.64 | 980.25 |
| CC-PAID | 1759.65 | 1155.85 | 756.72 | 498.65 |
| CCDR-PAID | 1581.20 | 1050.48 | 696.50 | 464.86 |

図 6.6 最小支持度の変更幅が 0.02 の時のデータセット $D_1$ の処理時間

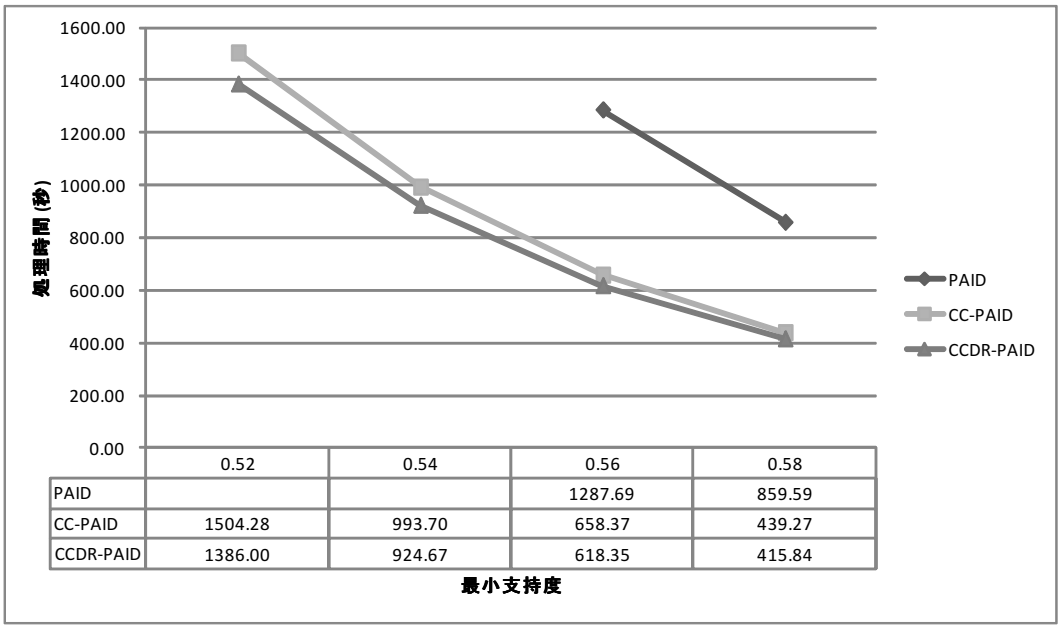
CCDR-PAID では，CC-PAID に対して最小支持度 0.48 の時に約 10%，0.5 の時に約 9%，0.52 の時に約 8%，0.54 の時に約 7%，処理時間が短縮された．

また，PAID に対して CCDR-PAID は最小支持度 0.52 と 0.54 の時に最大で約 53%，処理時間を短縮した．

- (図 6.7) 基準に対して時系列データ数が 2 倍のデータセット $D_2$，最小支持度 0.52〜0.58
  - 処理時間

    CC-PAID では，PAID に対して最小支持度 0.56 と 0.58 の時に約 49%，処理時間が短縮された．

    CCDR-PAID では，CC-PAID に対して最小支持度 0.52 の時に約 8%，0.54 の時に約 7%，0.56 の時に約 6%，0.58 の時に約 5%，処理時間が短縮された．




| | 0.52 | 0.54 | 0.56 | 0.58 |
|---|---|---|---|---|
| PAID | | | 1287.69 | 859.59 |
| CC-PAID | 1504.28 | 993.70 | 658.37 | 439.27 |
| CCDR-PAID | 1386.00 | 924.67 | 618.35 | 415.84 |

図 6.7 最小支持度の変更幅が 0.02 の時のデータセット $D_2$ の処理時間

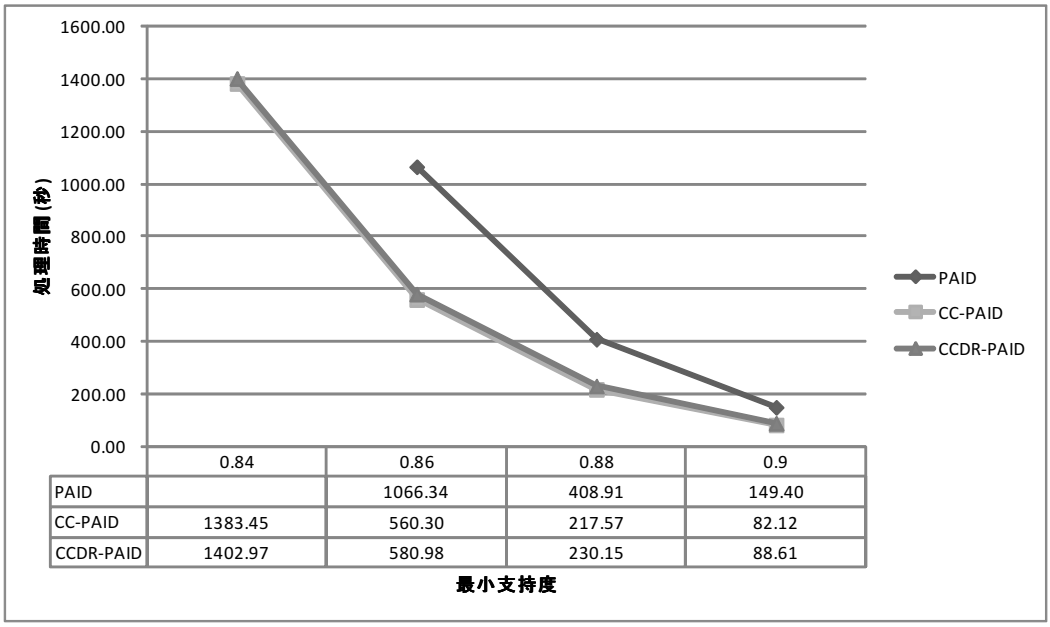
また，PAID に対して CCDR-PAID は最小支持度 0.56 と 0.58 の時に最大で約 52%，処理時間を短縮した．

- (図 6.8) 基準に対して平均トランザクション数が 2 倍のデータセット $D_3$，最小支持度 0.84〜0.9
  - 処理時間

    CC-PAID では，PAID に対して最小支持度 0.86 と 0.88 の時に約 47%，0.9 の時に約 45%，処理時間が短縮された．

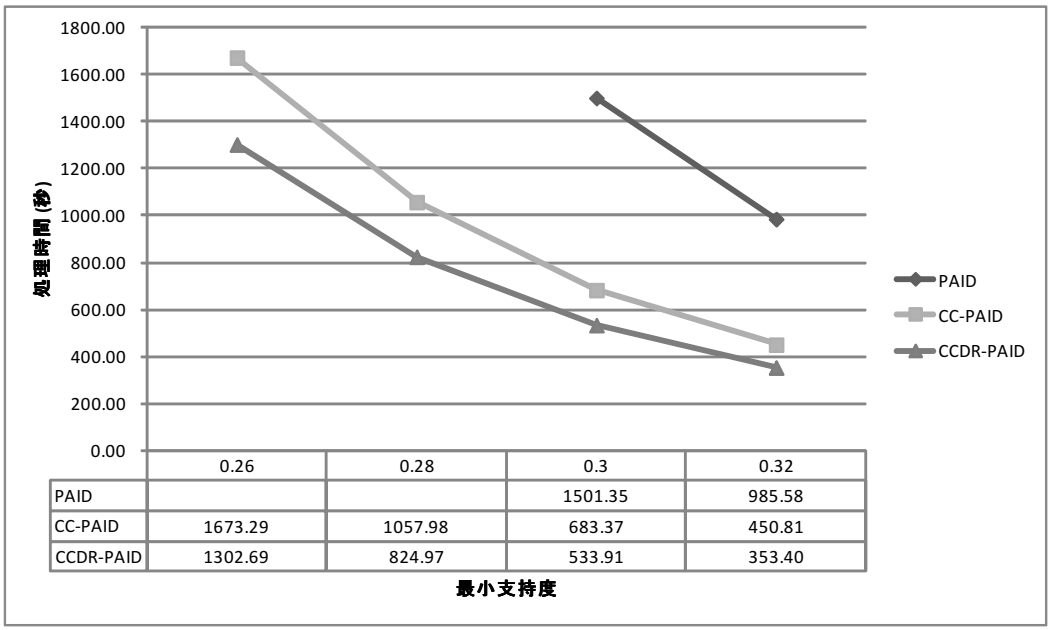
    CCDR-PAID では，CC-PAID に対して最小支持度 0.84 の時に約 1%，0.86 の時に約 4%，0.88 の時に約 6%，0.9 の時に約 8%，処理時間が増加した．

    また，PAID に対して CCDR-PAID は最小支持度 0.86 の時に最大で約 46%，処理時間を短縮した．



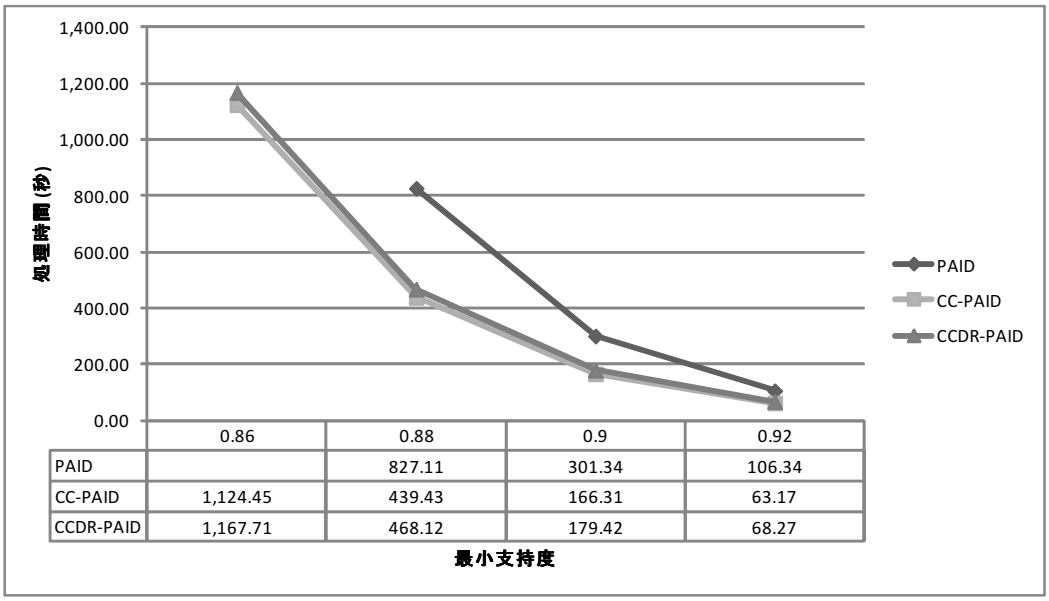

| | 0.84 | 0.86 | 0.88 | 0.9 |
|---|---|---|---|---|
| PAID | | 1066.34 | 408.91 | 149.40 |
| CC-PAID | 1383.45 | 560.30 | 217.57 | 82.12 |
| CCDR-PAID | 1402.97 | 580.98 | 230.15 | 88.61 |

図 6.8 最小支持度の変更幅が 0.02 の時のデータセット $D_3$ の処理時間




| | 0.26 | 0.28 | 0.3 | 0.32 |
|---|---|---|---|---|
| PAID | | | 1501.35 | 985.58 |
| CC-PAID | 1673.29 | 1057.98 | 683.37 | 450.81 |
| CCDR-PAID | 1302.69 | 824.97 | 533.91 | 353.40 |

図 6.9 最小支持度の変更幅が 0.02 の時のデータセット $D_4$ の処理時間




| | 0.86 | 0.88 | 0.9 | 0.92 |
|---|---|---|---|---|
| PAID | | 827.11 | 301.34 | 106.34 |
| CC-PAID | 1,124.45 | 439.43 | 166.31 | 63.17 |
| CCDR-PAID | 1,167.71 | 468.12 | 179.42 | 68.27 |

図 6.10 最小支持度の変更幅が 0.02 の時のデータセット $D_5$ の処理時間

- (図 6.9) 基準に対してアイテムの種類数が 2 倍のデータセット $D_4$，最小支持度 0.26〜0.32
  - 処理時間

    CC-PAID では，PAID に対して最小支持度 0.3 と 0.32 の時に約 54%，処理時間が短縮された．

    CCDR-PAID では，CC-PAID に対して最小支持度 0.26，0.28，0.3，0.32 の時に約 22%，処理時間が短縮された．

    また，PAID に対して CCDR-PAID は最小支持度 0.3 と 0.32 の時に最大で約 64%，処理時間を短縮した．

- (図 6.10) 基準に対して時系列データ数と平均トランザクション数が 2 倍のデータセット $D_5$，最小支持度 0.86〜0.92
  - 処理時間

    CC-PAID では，PAID に対して最小支持度 0.88 の時に約 47%，0.9 の時に約 45%，0.92 の時に約 41%，処理時間を短縮した．

CCDR-PAID では，CC-PAID に対して最小支持度 0.86 の時に約 4%，0.88 の時に約 7%，0.9 と 0.92 の時に約 8%，処理時間が増加した．

また，PAID に対して CCDR-PAID は最小支持度 0.88 の時に最大で約 43%，処理時間を短縮した．

**考察**

CC-PAID は PAID に対して，最大で約 54%の処理時間の短縮が確認され，CCDR-PAID は CC-PAID に対して，最大で約 22%の処理時間の短縮が確認された．

基準に対してアイテムの種類数が 2 倍のデータセット $D_4$ で PAID に対する CC-PAID の処理時間が短縮される割合と CC-PAID に対する CCDR-PAID の処理時間が短縮される割合が大きく，基準に対して平均トランザクション数が 2 倍のデータセット $D_3$，基準に対して時系列データ数と平均トランザクション数が 2 倍のデータセット $D_5$ で，CC-PAID は PAID に対して処理時間が短縮される割合が小さく，CCDR-PAID に関しては CC-PAID に対して処理時間が増加する結果となった．このため，平均トランザクション数とアイテムの種類数が処理時間に影響を与える可能性が高いと考えられる．

また，最小支持度の変更幅が 0.1 の時と同様に，最小支持度を下げると PAID に対する CC-PAID の処理時間が短縮される割合と CC-PAID に対する CCDR-PAID の処理時間が短縮する割合が増加する傾向が確認できた．

## 2.3 処理時間の内訳

各アルゴリズムの処理時間の内訳を調べるために，PAID，CC-PAID，CCDR-PAID の前処理 (前処理)，position list に関連した処理 (PL 関連)，ILP-list に関連した処理 (ILP 関連)，CCDR-PAID では更に ILP-DB の再構築処理 (再構築) の時間の計測を行った．“ () ”内は図 6.11〜図 6.15 と図 6.17 の項目名に対応する．

前処理は長さ 1 の時系列パターンの発見と position list-DB の生成と ILP-DB の生成であり，position list に関連した処理は $k$-pbp_set または multi-$k$-pbp_set の

取得処理，ILP-list に関連した処理は $E_k$-position の調査と非出現なアイテムの差し引き処理となる．

最小支持度は 6 章 2.2 節の変更幅を 0.02 とした評価結果から，それぞれのデータセットで PAID，CC-PAID，CCDR-PAID の三つのアルゴリズムが計測された最も小さな最小支持度を用い，3 回計測して平均した．

計測には gettimeofday 関数を利用した計測用関数を用いた．各処理の処理時間の計測を行う際には計測用関数のオーバーヘッドが生じるため，計測用関数を 10,000,000 回呼び出してオーバーヘッドの平均を 0.034 マイクロ秒とし，PAID，CC-PAID，CCDR-PAID の各処理の処理時間からそれぞれのオーバーヘッドの合計時間を差し引いた．プログラム全体の処理時間からプログラム全体で生じたオーバーヘッドの合計を差し引いた時間と 6 章 2.2 節の結果で示されている処理時間で差が生じていたため，更に，プログラム全体の処理時間からプログラム全体で生じたオーバーヘッドの合計を差し引いた時間で 6 章 2.2 節の結果で示されている処理時間を割り，その値を前処理を除いたそれぞれの処理の処理時間に掛けることによって補正を行った．このため，実際の処理時間に対して誤差が生じている可能性があり，評価の目安として結果を示す．

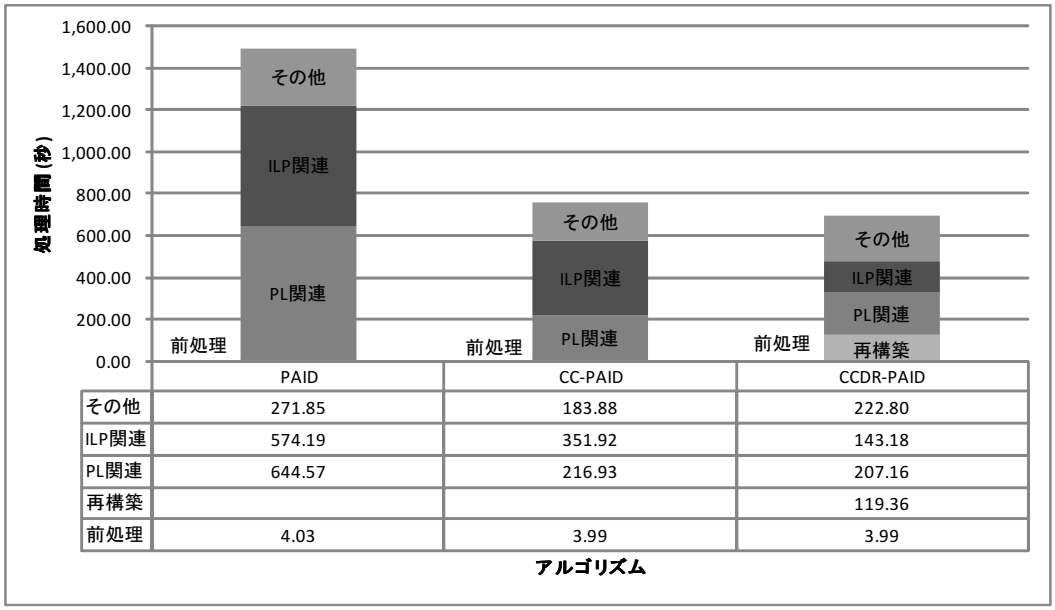
また，計測結果の図に示されるその他の項目は前処理，PL 関連，ILP 関連，再構築以外の処理時間となる．なお，前処理の項目に関してはアルゴリズム毎の処理時間の差は誤差と考えられ，その他の項目に関しては計測用関数によって計測した値ではないため，比較は行わない．

図 6.11〜図 6.15 と図 6.17 の前処理の処理時間が小さく，他の処理のラベルと重なるため，前処理のラベルを左側に記載している．

- (図 6.11) 基準となるデータセット $D_1$，最小支持度 0.52
  - 処理時間

    CC-PAID では，PAID に対して PL 関連の処理が約 66%，ILP 関連の処理が約 39%，処理時間が短縮された．

    CCDR-PAID では，CC-PAID に対して PL 関連の処理で約 5%，ILP 関連の処理で約 59%，処理時間が短縮された．また，CCDR-PAID に




| | PAID | CC-PAID | CCDR-PAID |
|---|---|---|---|
| その他 | 271.85 | 183.88 | 222.80 |
| ILP関連 | 574.19 | 351.92 | 143.18 |
| PL関連 | 644.57 | 216.93 | 207.16 |
| 再構築 | | | 119.36 |
| 前処理 | 4.03 | 3.99 | 3.99 |

図 6.11 データセット $D_1$ の処理時間の内訳

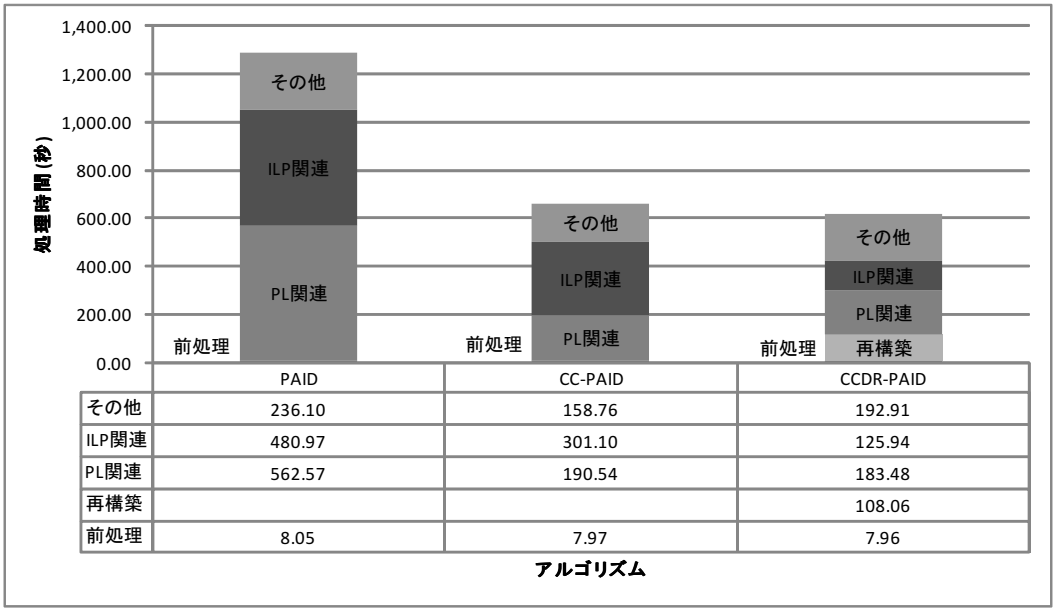
おいて再構築と PL 関連と ILP 関連の合計に対する再構築の割合は約 25%となった.

- (図 6.12) 基準に対して時系列データ数が 2 倍のデータセット $D_2$，最小支持度 0.56
  - 処理時間

    CC-PAID では，PAID に対して PL 関連の処理が約 66%，ILP 関連の処理が約 37%，処理時間が短縮された.

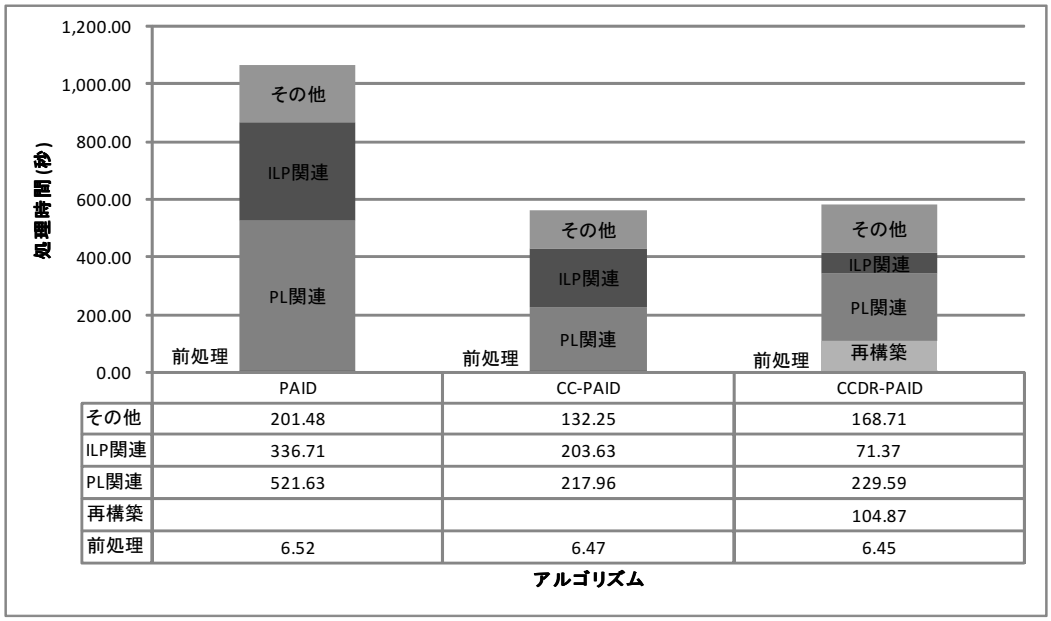
    CCDR-PAID では，CC-PAID に対して PL 関連の処理で約 4%，ILP 関連の処理で約 58%，処理時間が短縮された．また，CCDR-PAID において再構築と PL 関連と ILP 関連の合計に対する再構築の割合は約 26%となった.

- (図 6.13) 基準に対して平均トランザクション数が 2 倍のデータセット $D_3$，最小支持度 0.86



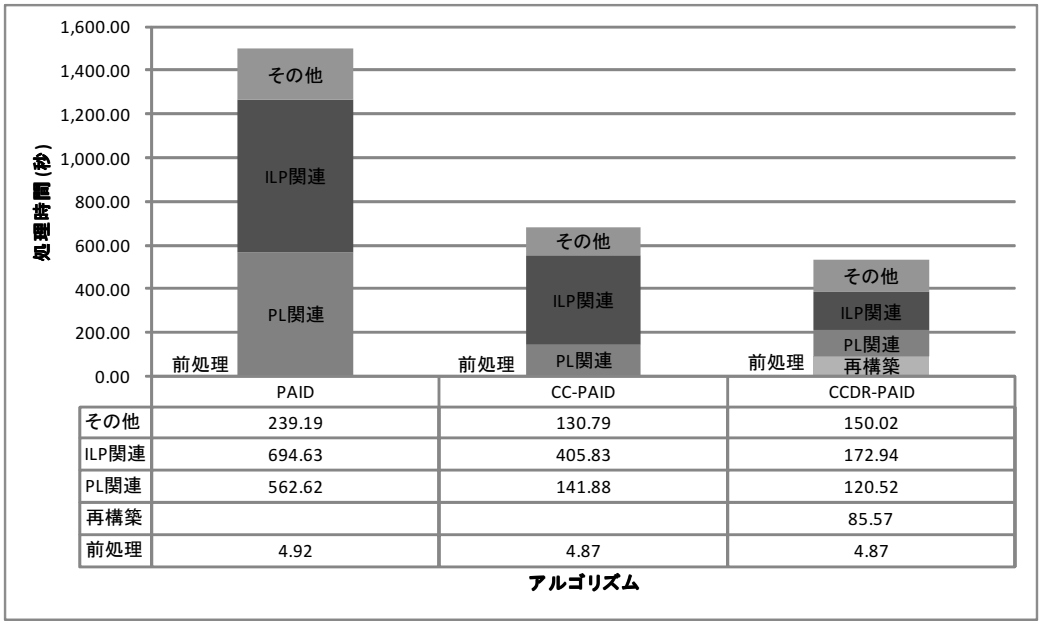

| | PAID | CC-PAID | CCDR-PAID |
|---|---|---|---|
| その他 | 236.10 | 158.76 | 192.91 |
| ILP関連 | 480.97 | 301.10 | 125.94 |
| PL関連 | 562.57 | 190.54 | 183.48 |
| 再構築 | | | 108.06 |
| 前処理 | 8.05 | 7.97 | 7.96 |

図 6.12 データセット $D_2$ の処理時間の内訳




| | PAID | CC-PAID | CCDR-PAID |
|---|---|---|---|
| その他 | 201.48 | 132.25 | 168.71 |
| ILP関連 | 336.71 | 203.63 | 71.37 |
| PL関連 | 521.63 | 217.96 | 229.59 |
| 再構築 | | | 104.87 |
| 前処理 | 6.52 | 6.47 | 6.45 |

図 6.13 データセット $D_3$ の処理時間の内訳




| | PAID | CC-PAID | CCDR-PAID |
|---|---|---|---|
| その他 | 239.19 | 130.79 | 150.02 |
| ILP関連 | 694.63 | 405.83 | 172.94 |
| PL関連 | 562.62 | 141.88 | 120.52 |
| 再構築 | | | 85.57 |
| 前処理 | 4.92 | 4.87 | 4.87 |

図 6.14　データセット $D_4$ の処理時間の内訳

– 処理時間

CC-PAID では，PAID に対して PL 関連の処理が約 58%，ILP 関連の処理が約 40%，処理時間が短縮された．

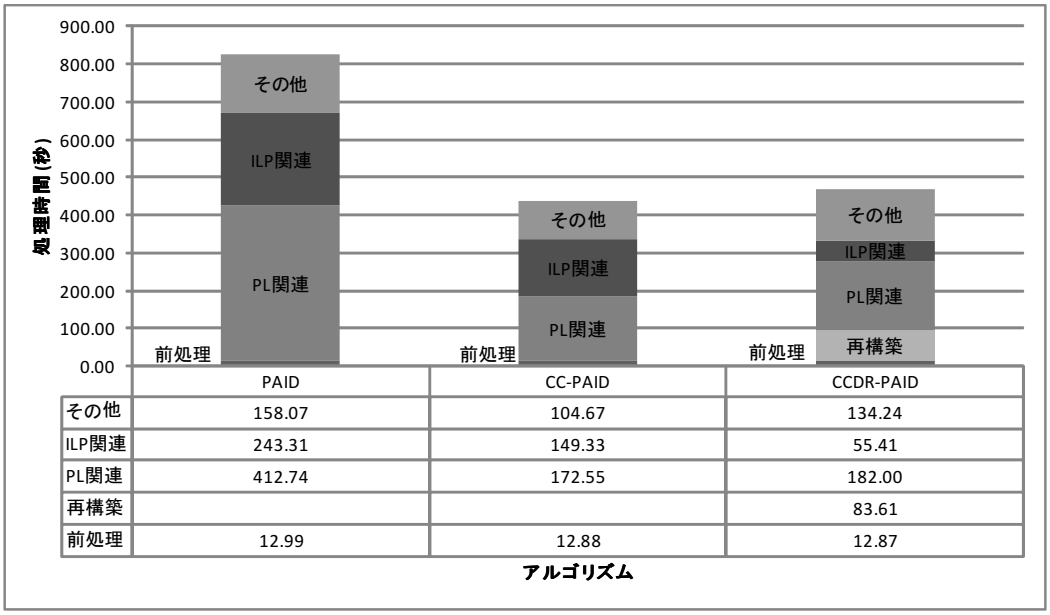
CCDR-PAID では，CC-PAID に対して ILP 関連の処理で約 65%，処理時間が短縮され，PL 関連の処理で約 5%，処理時間が増加した．また，CCDR-PAID において再構築と PL 関連と ILP 関連の合計に対する再構築の割合は約 26%となった．

- (図 6.14) 基準に対してアイテムの種類数が 2 倍のデータセット $D_4$，最小支持度 0.3

– 処理時間

CC-PAID では，PAID に対して PL 関連の処理が約 75%，ILP 関連の処理が約 42%，処理時間が短縮された．

CCDR-PAID では，CC-PAID に対して PL 関連の処理で約 15%，ILP 関連の処理で約 57%，処理時間が短縮された．また，CCDR-PAID に




| | PAID | CC-PAID | CCDR-PAID |
|---|---|---|---|
| その他 | 158.07 | 104.67 | 134.24 |
| ILP関連 | 243.31 | 149.33 | 55.41 |
| PL関連 | 412.74 | 172.55 | 182.00 |
| 再構築 | | | 83.61 |
| 前処理 | 12.99 | 12.88 | 12.87 |

図 6.15 データセット $D_5$ の処理時間の内訳

おいて再構築と PL 関連と ILP 関連の合計に対する再構築の割合は約 23%となった.

- (図 6.15) 基準に対して時系列データ数と平均トランザクション数が 2 倍のデータセット $D_5$，最小支持度 0.88
  - 処理時間

    CC-PAID では，PAID に対して PL 関連の処理が約 58%，ILP 関連の処理が約 39%，処理時間が短縮された.

    CCDR-PAID では，CC-PAID に対して ILP 関連の処理で約 63%，処理時間が短縮され，PL 関連の処理で約 5%，処理時間が増加した．また，CCDR-PAID において再構築と PL 関連と ILP 関連の合計に対する再構築の割合は約 26%となった.

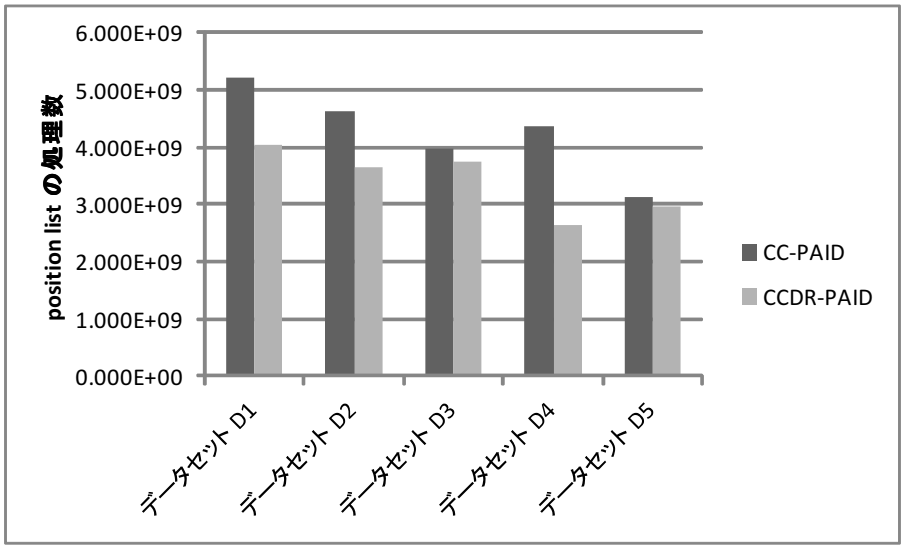



図 6.16 CC-PAID と CCDR-PAID の position list の処理数

**考察**

CC-PAID は PAID に対して，全てのデータセットで PL 関連と ILP 関連の処理時間の短縮が確認され，ILP 関連に対して PL 関連の処理時間の方が短縮割合が大きいといった傾向が確認できた．

CCDR-PAID は CC-PAID に対して，全てのデータセットで ILP 関連の処理時間は短縮できたが，PL 関連の処理に関しては基準のデータセットに対して平均トランザクション数が 2 倍となっているデータセット $D_3$，$D_5$ で処理時間が増加し，他のデータセットでは処理時間が短縮されているがその割合は小さいといった結果となった．

このため，データセット毎の CC-PAID と CCDR-PAID の position list の処理数を調べた (図 6.16)．データセット毎の最小支持度は本節で利用されている値を用いた．

結果として，CC-PAID に対して CCDR-PAID はデータセット $D_1$ で約 22%，$D_2$ で約 21%， $D_3$ で約 6%， $D_4$ で約 39%， $D_5$ で約 5%， position list の処理の省略が行われたことがわかった．しかしながら，基準のデータセットに対して平均トランザクション数が 2 倍となっているデータセット $D_3$，$D_5$ では CC-PAID

に対してのCCDR-PAIDのPL関連の処理時間が増加するといった結果となっている．これは，CCDR-PAIDにおけるmulti-$k$-pbpの取得処理の実装が原因と考えられる．CCDR-PAIDは取得した$k$-pbpを一時的に他のデータ構造に保持し，そこからmulti-$k$-pbpのデータ構造に代入を行っており，CC-PAIDよりも余分な処理が行われている．このため，CCDR-PAIDのmulti-$k$-pbpのデータ構造への代入処理がボトルネックとなり，position listが多く省略されないと処理時間を短縮できないといった原因が考えられる．

また，CCDR-PAIDではCC-PAIDに対するILP関連の処理時間の減少は確認できたが，ILP-listの再構築処理の処理時間が加算され，処理時間を効率的に短縮できていないことが，CC-PAIDの前処理，PL関連，ILP関連の処理時間の合計時間とCCDR-PAIDの前処理，再構築，PL関連，ILP関連の処理時間の合計時間により，処理時間の内訳の図(6.11〜6.15，6.17)から確認できる．

CCDR-PAIDの上記二点の解決方法として，まず，multi-$k$-pbpのデータ構造への代入処理のボトルネックについては，一時的に$k$-pbpを保持するデータ構造をmulti-$k$-pbpとして利用できる可能性があり，実装の最適化を行うことにより解消する可能性がある．ILP-listの再構築処理に関しても，閾値を用いてILP-listのデータ長がある程度の短さになった場合，再構築をやめることにより，再構築のコストを抑えることが可能であると考えられる．

また，6章2.1と2.2節の結果から，基準のデータセットに対して平均トランザクション数が2倍になった場合，CC-PAIDよりもCCDR-PAIDの処理時間が大きくなったため，基準のデータセット$D_1$に対して平均トランザクション数が5倍のデータセット$D_6$を作成し，最小支持度0.987を用いて，トランザクション数を更に増やした時にCC-PAIDに対するCCDR-PAIDの処理時間の変化の割合に変化が生じるか調べた(図6.17)．なお，処理時間の内訳の計測方法とオーバーヘッドの差し引き及び補正は他のデータセットと同じ条件で行い，計測回数は一回となっている．

結果として，CCDR-PAIDはCC-PAIDに対してILP関連の処理で約65%，処理時間を短縮し，PL関連の処理で約6%，処理時間が増加し，再構築とPL関連とILP関連の合計に対する再構築の割合は約23%となった．この結果から，本節

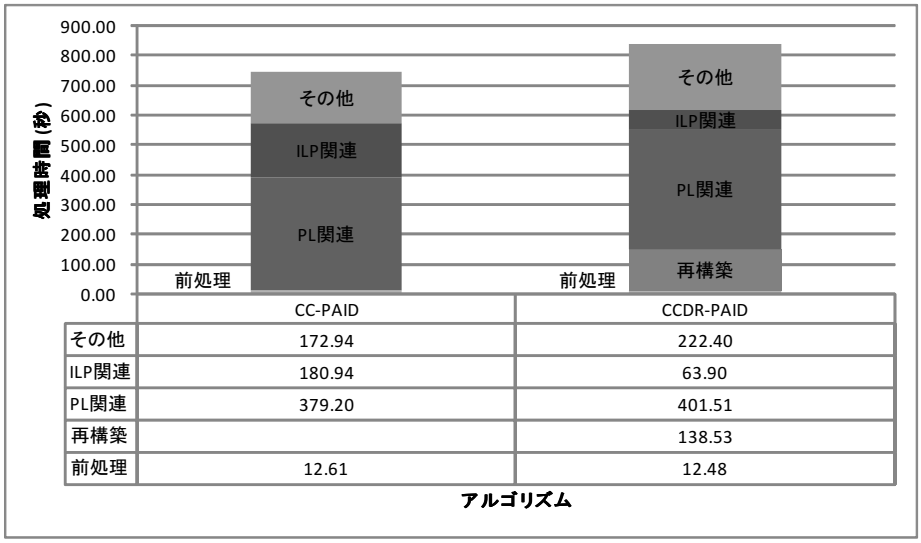



図 6.17 基準のデータセットに対して平均トランザクション数が 5 倍のデータセットの処理時間の内訳

で計測した基準のデータセットに対して平均トランザクション数が 2 倍となっているデータセット $D_3$，$D_5$ と CC-PAID に対する CCDR-PAID の処理時間の変化の割合は近いと考えられ，トランザクション数の増加が直接的に CCDR-PAID の処理時間の削減割合の低下の原因には繋がっていないと考えられる．このため，本節のデータセット $D_3$ の処理時間の内訳から，基準となるデータセット $D_1$ では PL 関連の処理時間が CC-PAID に対して CCDR-PAID が短縮したのに対して，基準のデータセットに対して平均トランザクション数が 2 倍のデータセットでは処理時間が増加しているため，PL 関連の処理時間の増加が CC-PAID に対して CCDR-PAID の処理時間が増加した原因と考えられる．

これに対し，基準のデータセットに対してアイテムの種類数が 2 倍のデータセット $D_4$ は 6 章 2.1 と 2.2 節の結果において，CCDR-PAID が CC-PAID に対して最も処理時間の削減割合が大きいデータセットである．これは，本節のデータセット $D_4$ の処理時間の内訳から，基準となるデータセット $D_1$ の CC-PAID に対する CCDR-PAID の PL 関連の処理時間の削減割合が約 5%なのに対し，データセット $D_4$ では約 15%となっており，PL 関連の処理時間の削減割合が影響していると考えられる．また，他のデータセットに対し，PL 関連と ILP 関連と再構築の処理

時間の合計に対する再構築の処理時間が占める割合も小さいことから，ILP-list の再構築のコストの小ささも影響したと考えられる．

データセット $D_3$ と $D_4$ の違いは，時系列データ内の同じアイテムの出現頻度と考えられる．$D_3$ は基準のデータセットに対してアイテムの種類数がそのままの状態で平均トランザクション数を 2 倍にしているため，時系列データ内の同じアイテムの出現頻度は高くなり，CCDR-PAID による position list 関連の処理が省略されにくく，一方，$D_4$ は基準のデータセットに対して平均トランザクション数はそのままの状態でアイテムの種類数を 2 倍にしているため，時系列データ内の同じアイテムの出現頻度は低くなり，CCDR-PAID による position list 関連の処理が省略されやすいと考えられる．これは，図 6.16 の CC-PAID に対する CCDR-PAID の position list の処理が省略される割合からも，省略される割合として基準のデータセットに対して平均トランザクション数が 2 倍となっているデータセット $D_3$ と $D_5$ が低く，基準のデータセットに対してアイテムの種類数が 2 倍のデータセット $D_4$ が最も高くなるといったことからもわかる．

また，ILP 関連の処理では CC-PAID に対する CCDR-PAID の処理時間の削減割合では，基準のデータセットに対して平均トランザクション数が 2 倍となっているデータセット $D_3$ と $D_5$ が大きく，基準のデータセットに対してアイテムの種類数が 2 倍のデータセット $D_4$ が最も小さいといった結果となった．

基準のデータセットに対して平均トランザクション数が 2 倍となると時系列データ内の同じアイテムの出現頻度が高くなり，CC-PAID が ILP-list を処理した際に不要なアイテムを含んだ状態で処理されたのに対し，CCDR-PAID では効率よく不要なアイテムを ILP-list から取り除いて再構築された ILP-list を処理できたためと考えられる．これに対し，基準のデータセットに対してアイテムの種類数が 2 倍になると，時系列データ内の同じアイテム出現頻度が低くなり，ILP-list から不要なアイテムを削減した際の CC-PAID に対する CCDR-PAID の効果が小さくなったためと考えられる．

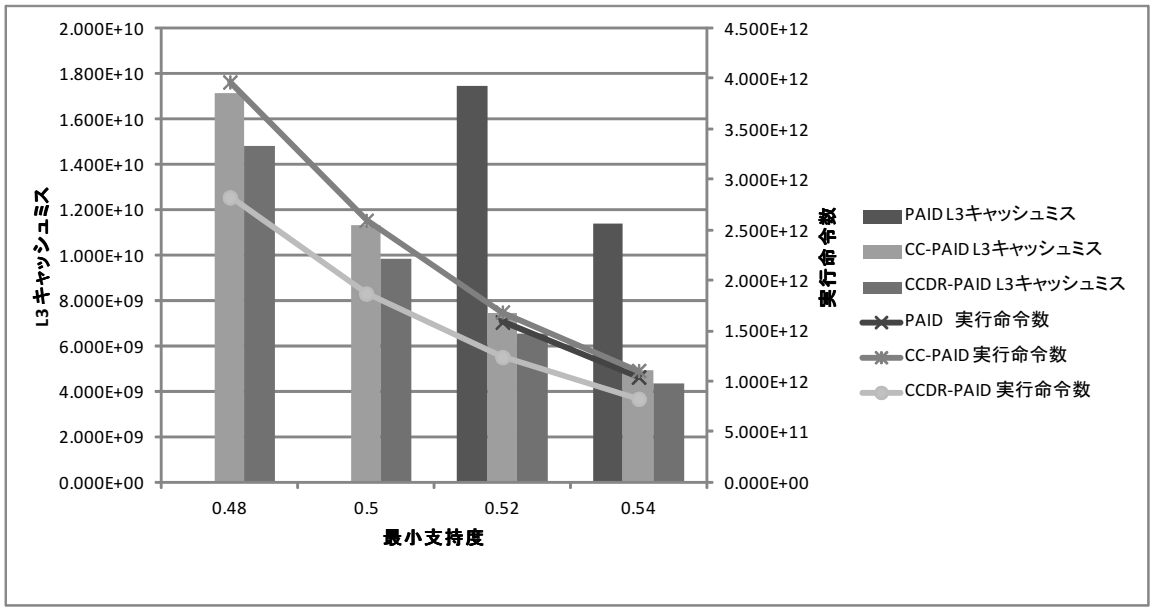



図 6.18 データセット $D_1$ の L3 キャッシュミスと実行命令数

## 2.4 実行命令数とキャッシュミス

最小支持度は 6 章 2.2 節の変更幅を 0.02 とした評価と同じ最小支持度を用い，それぞれのデータセットで PAID，CC-PAID，CCDR-PAID の実行命令数と L3 キャッシュミスを計測した.

- (図 6.18) 基準となるデータセット $D_1$，最小支持度 0.48〜0.54
  - 実行命令数

    CC-PAID では，PAID に対して最小支持度 0.52 と 0.54 の時に約 6%，実行命令数が増加した.

    CCDR-PAID では，CC-PAID に対して最小支持度 0.48 の時に約 29%，0.5 の時に約 28%，0.52 の時に約 26%，0.54 の時に約 25%，実行命令数が削減された.
  - キャッシュミス

    CC-PAID では，PAID に対して最小支持度 0.52 と 0.54 の時に約 57%，L3 キャッシュミスが削減された.

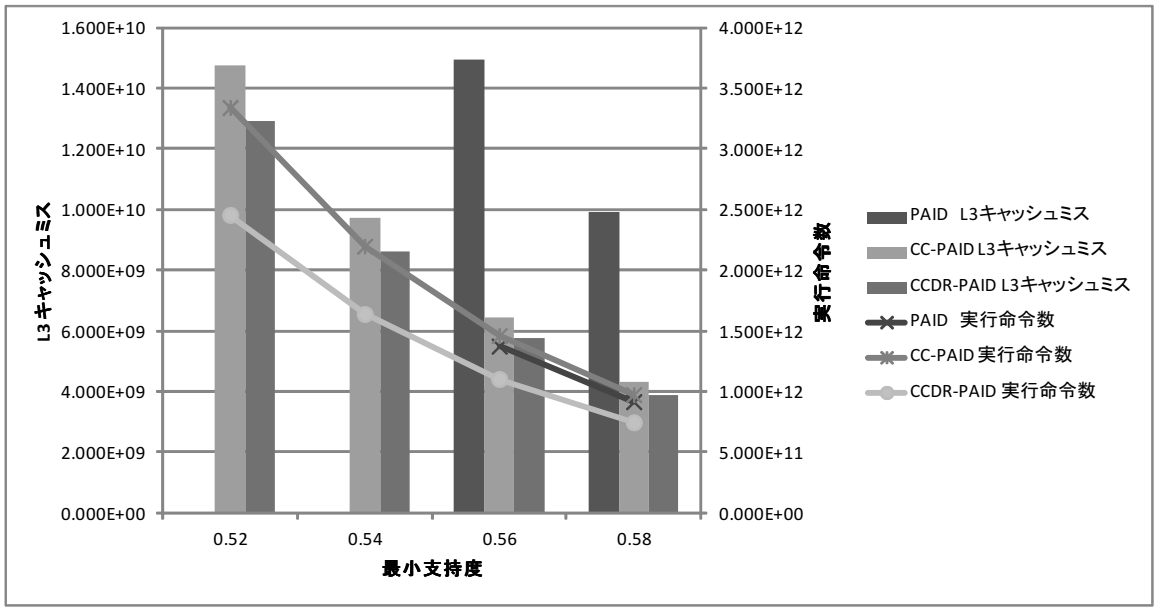



図 6.19 データセット $D_2$ の L3 キャッシュミスと実行命令数

CCDR-PAID では，CC-PAID に対して最小支持度 0.48 の時に約 14%，0.5 と 0.52 の時に約 13%，0.54 の時に約 12%，L3 キャッシュミスが削減された．

- (図 6.19) 基準に対して時系列データ数が 2 倍のデータセット $D_2$，最小支持度 0.52〜0.58
  - 実行命令数

    CC-PAID では，PAID に対して最小支持度 0.56 と 0.58 の時に約 6%，実行命令数が増加した．

    CCDR-PAID では，CC-PAID に対して最小支持度 0.52 の時に約 26%，0.54 の時に約 25%，0.56 の時に約 24%，0.58 の時に約 23%，実行命令数が削減された．
  - キャッシュミス

    CC-PAID では，PAID に対して最小支持度 0.56 と 0.58 の時に約 57%，L3 キャッシュミスが削減された．

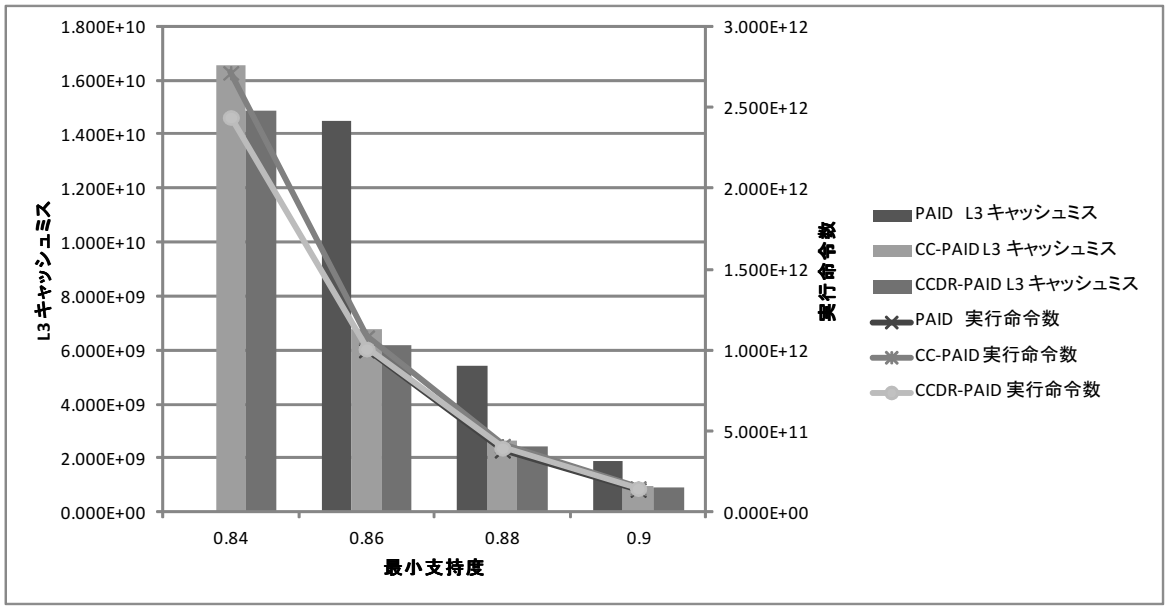



図 6.20 データセット $D_3$，L3 キャッシュミスと実行命令数

CCDR-PAID では，CC-PAID に対して最小支持度 0.52 と 0.54 の時に約 12%，0.56 の時に約 11%，0.58 の時に約 10%，L3 キャッシュミスが削減された．

- (図 6.20) 基準に対して平均トランザクション数が 2 倍のデータセット $D_3$，最小支持度 0.84〜0.9
  - 実行命令数
    CC-PAID では，PAID に対して最小支持度 0.86，0.88，0.9 の時に約 8%，実行命令数が増加した．

    CCDR-PAID では，CC-PAID に対して最小支持度 0.84 の時に約 10%，0.86 の時に約 7%，0.88 の時に約 4%，0.9 の時に約 2%，実行命令数が削減された．
  - キャッシュミス
    CC-PAID では，PAID に対して最小支持度 0.86 の時に約 53%，0.88 の時に約 51%，0.9 の時に約 49%，L3 キャッシュミスが削減された．

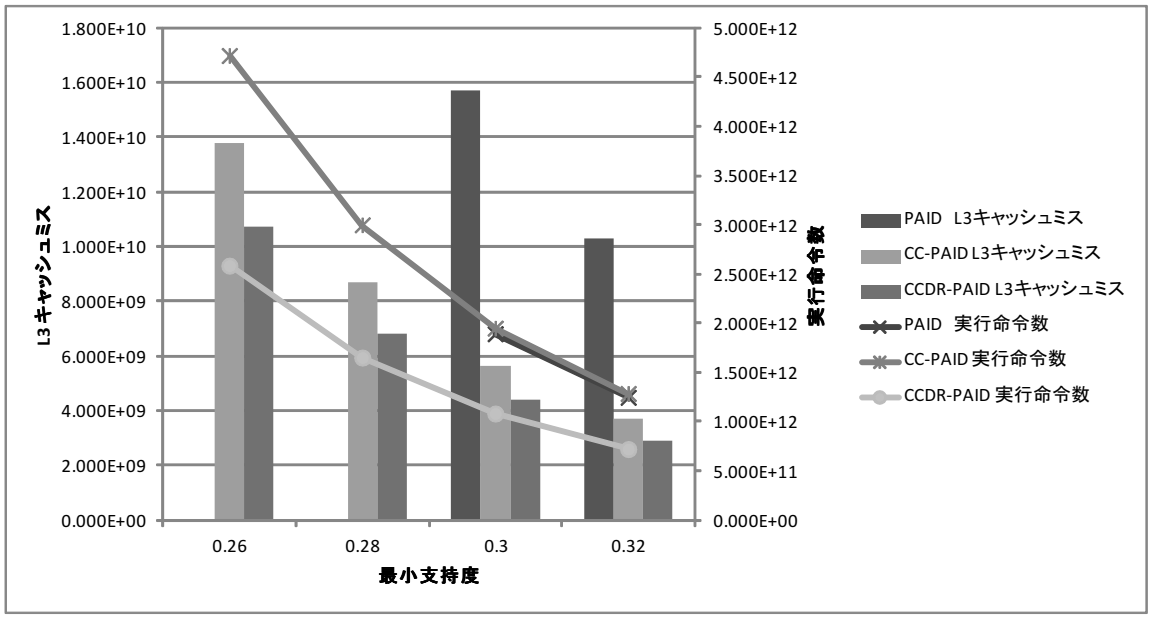



図 6.21 データセット $D_4$，L3 キャッシュミスと実行命令数

CCDR-PAID では，CC-PAID に対して最小支持度 0.84 の時に約 10%，0.86 の時に約 9%，0.88 の時に約 7%，0.9 の時に約 5%，L3 キャッシュミスが削減された．

- (図 6.21) 基準に対してアイテムの種類数が 2 倍のデータセット $D_4$，最小支持度 0.26〜0.32
  - 実行命令数

    CC-PAID では，PAID に対して最小支持度 0.3 と 0.32 の時に約 3%，実行命令数が増加した．

    CCDR-PAID では，CC-PAID に対して最小支持度 0.26，0.28，0.3 の時に約 45%，0.32 の時に約 44%，実行命令数が削減された．
  - キャッシュミス

    CC-PAID では，PAID に対して最小支持度 0.3 と 0.32 の時に約 64%，L3 キャッシュミスが削減された．

    CCDR-PAID では，CC-PAID に対して最小支持度 0.26 と 0.28 の時に

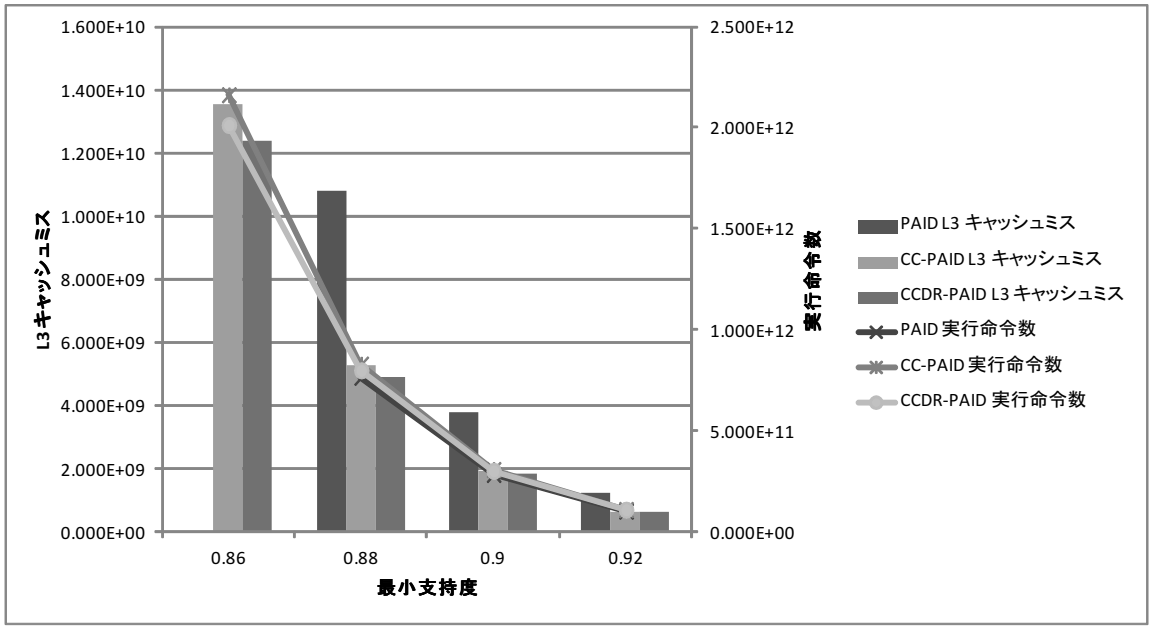



図 6.22 データセット $D_5$，L3 キャッシュミスと実行命令数

約 22%，0.3 と 0.32 の時に約 21%，L3 キャッシュミスが削減された．

- (図 6.22) 基準に対して時系列データ数と平均トランザクション数が 2 倍のデータセット $D_5$，最小支持度 0.86〜0.92
  - 実行命令数

    CC-PAID では，PAID に対して最小支持度 0.88，0.9，0.92 の時に約 8%，実行命令数が増加した．

    CCDR-PAID では，CC-PAID に対して最小支持度 0.86 の時に約 7%，0.88 の時に約 4%，0.9 の時に約 2%，0.92 の時に約 1%，実行命令数が削減された．
  - キャッシュミス

    CC-PAID では，PAID に対して最小支持度 0.88 の時に約 51%，0.9 の時に約 49%，0.92 の時に約 47%，L3 キャッシュミスが削減された．

    CCDR-PAID では，CC-PAID に対して最小支持度 0.86 の時に約 8%，0.88 の時に約 7%，0.9 の時に約 5%，0.92 の時に約 4%，L3 キャッシュ

ミスが削減された．

**考察**

CC-PAID は，PAID に対して実行命令数が最大で約 8%，増加し，L3 キャッシュミスは最大で約 64%，削減された．CCDR-PAID は CC-PAID に対して実行命令数が最大約 45%，削減され，L3 キャッシュミスは最大で約 22%，削減された．また，最小支持度が下がるほど，PAID に対する CC-PAID の L3 キャッシュミスが削減され，CC-PAID に対する CCDR-PAID の実行命令数と L3 キャッシュミスにも同様の傾向が確認された．

最小支持度を下げると PAID に対する CC-PAID の処理時間が削減される割合が増加する傾向が確認されている．これは，最小支持度を下げた場合，時系列パターン数は増加するが，同時に長さ $k-1$ の時系列パターンを $common\ prefix$ として持つ $k$ 番目のアイテムも増える可能性があり，Common-Prefix-At-A-Time で一度に処理できる時系列パターンが増えることにより，CPU キャッシュの利用効率等が向上したと考えられる．

同様に，最小支持度を下げると CC-PAID に対する CCDR-PAID の処理時間の削減される割合が増加する傾向が確認されている．上記の理由と同様に長さ $k-1$ の時系列パターンを $common\ prefix$ として持つ $k$ 番目のアイテムが増加し，そのアイテムが多い分，CC-PAID に対する CCDR-PAID の position list に関連した処理の省略される割合が増え，ILP-list に関しては CC-PAID が不要なアイテムを含んだ状態の ILP-list を処理する数が増えるに対し，CCDR-PAID が再構築された ILP-list を効率的に処理できたためと考えられる．このため，時系列パターン数の増加傾向や長さ $k-1$ の時系列パターンを $common\ prefix$ として持つ $k$ 番目のアイテムの平均数に関して調査を行っていく必要があると考えられる．

CC-PAID と CCDR-PAID の実行命令数とキャッシュミスの関係に着目すると，PAID に対して CC-PAID は実行命令数は増加するが L3 キャッシュミスは削減され，同時に 6 章 2.2 節の結果から処理時間も短縮されている．このため，CC-PAID は主に CPU キャッシュの利用効率の向上により処理時間を短縮させたと考えられる．また，CC-PAID に対し CCDR-PAID は実行命令数と L3 キャッシュミスの

表 6.3　プログラム全体の実行命令数，L3 キャッシュミス，レイテンシの割合，CPI (レイテンシの割合と CPI は参考に用いる値)

| | 実行命令数 | L3 miss | レイテンシの割合 (%) | CPI |
|---|---|---|---|---|
| PAID | 1,588,312,000,000 | 17,436,550,000 | 63.87 | 3.09 |
| CC-PAID | 1,687,220,000,000 | 7,449,080,000 | 63.82 | 1.25 |
| CCDR-PAID | 1,241,628,000,000 | 6,510,090,000 | 68.98 | 1.37 |

削減が確認されたため，処理時間が短縮される要因は計算量と L3 キャッシュミスの削減によるものと考えられる．

なお，CCDR-PAID は PAID に対して，実行命令数ではデータセット $D_4$ の最小支持度 0.3 の時に最大で約 43%，実行命令数が削減され，L3 キャッシュミスではデータセット $D_4$ の最小支持度 0.3 と 0.32 の時に最大で約 72%，L3 キャッシュミスが削減された．

## 2.5 データセット $D_1$ を用いた詳細な評価

本節では，基準となるデータセット $D_1$ と最小支持度 0.52 を用いて，6 章 2.2 節の結果で CC-PAID と CCDR-PAID の処理時間が短縮された要因をより詳しく調べるために，プログラム全体とプログラム内部の各々の処理に関連した実行命令数と L3 キャッシュミス数，更に評価の参考として CPI 値 (Cycles Per Instruction) と全サイクル数の内のレイテンシが占める割合も示し，多角的に考察を行った．

CPI 値は 1 命令の実行に必要なクロック数を示す．

プログラム内の詳細な評価を行うためにコンパイラオプションとして $-O2\ -g$ を用いた．

### プログラム全体の評価

プログラム全体の実行命令数と L3 キャッシュミス (L3 miss)，また，評価の参考として全サイクル数の内のレイテンシが占める割合 (レイテンシの割合) と CPI 値を示す (表 6.3)．“ () ”内は表 6.3 と表 6.4 の各項目に対応する．

### 考察

CPI 値は CC-PAID が最も小さく，CPU の利用効率が PAID よりも 2 倍以上高くなっていると考えられる．CC-PAID の L3 キャッシュミスの値は PAID に対して約 57%，削減され，実行命令数の値は約 6%，増加していることから，PAID に対して CC-PAID が処理時間を短縮する要因は主に CPU のキャッシュの利用効率が向上するためと考えられる．

CCDR-PAID に関しては，CC-PAID よりも CPI 値が大きいがその差は小さく，誤差と考えられる．L3 キャッシュミスの値は CC-PAID に対して約 13%，削減され，実行命令数の値は約 26%，削減していることから，CC-PAID に対して処理時間が削減される要因は計算量と CPU のキャッシュミスが削減されたためと考えられる．

### プログラム内部の部分的な評価

プログラム全体の評価では，position list や ILP-list に関連した主要な処理以外で，計測した値に影響がある可能性があるため，プログラム内部の position list 関連 (PL 関連) と ILP-list 関連 (ILP 関連) の処理毎に評価を行った．また，CCDR-PAID では ILP-list の再構築処理 (再構築) についても表 6.4 に評価を示す．“ () ” 内は表 6.4 のアルゴリズム毎の各項目に対応する．なお，最適化オプションを利用しているため，各処理の検出のズレによる誤差が生じている可能性があり，表 6.4 の結果は評価の参考として用いる．

### 考察

PAID，CC-PAID，CCDR-PAID に関して PL 関連の処理に着目すると，PAID に対して CC-PAID では実行命令数が約 34%，削減され，L3 キャッシュミスは約 67%，削減された．CCDR-PAID では CC-PAID に対して実行命令数が約 11%，削減され，L3 キャッシュミスは約 14%，削減された．

PAID，CC-PAID，CCDR-PAID に関して ILP 関連の処理に着目すると，PAID に対して CC-PAID では実行命令数が約 22%，増加し，L3 キャッシュミスは約 77%，

表 6.4 プログラム内部の参考に用いる部分的な，実行命令数，L3 キャッシュミス，レイテンシの割合，CPI

| | 実行命令数 | L3 miss | レイテンシの割合 (%) | CPI |
|---|---|---|---|---|
| PAID PL 関連 | 423,596,000,000 | 15,718,600,000 | 91.86 | 7.27 |
| CC-PAID PL 関連 | 280,450,000,000 | 5,262,850,000 | 120.03 | 2.81 |
| CCDR-PAID PL 関連 | 248,896,000,000 | 4,528,620,000 | 107.07 | 3.06 |
| PAID ILP 関連 | 1,097,040,000,000 | 1,661,860,000 | 16.88 | 1.62 |
| CC-PAID ILP 関連 | 1,341,500,000,000 | 387,260,000 | 5.78 | 0.90 |
| CCDR-PAID ILP 関連 | 631,028,000,000 | 28,950,000 | 1.02 | 0.81 |
| CCDR-PAID 再構築 | 281,664,000,000 | 191,010,000 | 11.38 | 1.07 |

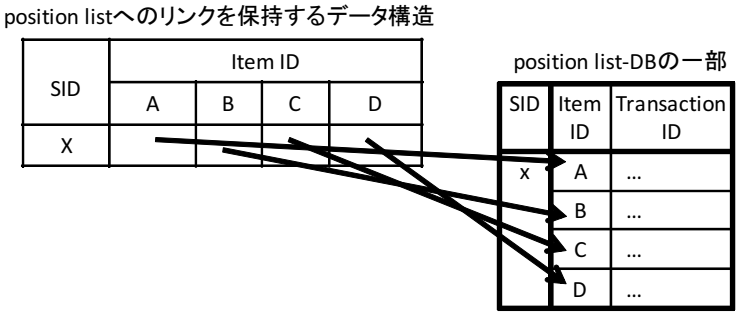



図 6.23 position list へのリンクを保持するデータ構造

削減された．CCDR-PAID では CC-PAID に対して実行命令数が約 53%，削減され，L3 キャッシュミスは約 93%，削減された．また，全サイクル数の内のレイテンシが占める割合と CPI 値が PAID，CC-PAID，CCDR-PAID の順に減少傾向にあることが確認できた．CC-PAID の ILP 関連の処理では，実行命令数が増加したのに対して，L3 キャッシュミスは削減され，6 章 2.3 節の ILP-list 関連の処理時間も削減されていることから，Common-Prefix-At-A-Time により CPU キャッシュの利用効率が向上したと考えられる．

なお，PAID の L3 キャッシュミスの値が大きいが，これは本論文で実装を行った PAID の position list へのアクセス方法が影響していると考えられる．本論文

における position list へのアクセスの実装は，時系列データ毎に各アイテムに関する position list へのリンクを保持するデータ構造を介して行っている (図 6.23)．このリンクを保持するデータ構造では，時系列データ毎に，CC-PAID と CCDR-PAID では複数の時系列パターンで position list へのリンクを得ることができるが，PAID では 1 つしかリンクを取得できないため，多くのキャッシュミスが生じたと考えられる．

このため，PAID においては，FP-growth [10] や FSPM [16] で利用されているようなアクセス方法として，アイテム毎に各時系列データのそのアイテムの position list へのリンクを保持するデータ構造を利用し，参考文献 [18] で行われているように，時系列データ毎の $k$-pbp をまとめて取得することでキャッシュミスを改善できる可能性がある．これに対し，CC-PAID，CCDR-PAID では PL 関連の処理に関して PAID よりもキャッシュミスは少ないが ILP 関連のキャッシュミスよりは多いため，同じような position list へのアクセス方法の改良を行っていく必要があると考えられる．

## 2.6 並列処理

データセット $D_5$ を用いて，スレッド数を変更し，PAID，CC-PAID，CCDR-PAID のスケーラビリティと実行命令数と L3 キャッシュミスを調べた．

最小支持度は 6 章 2.2 節の変更幅を 0.02 とした評価結果から，それぞれのデータセットで PAID，CC-PAID，CCDR-PAID の三つのアルゴリズムが計測された最も小さな最小支持度 0.88 を用い，スケーラビリティに関しては 3 回計測して平均を調べ，実行命令数と L3 キャッシュミスについても計測を 1 回行って調査した．

なお，並列処理時のキャッシュミスと実行命令数の計測で，スレッド数が少ない時に各コアで L3 キャッシュミスと実行命令数にバラつきがあるが，OS のスレッドスケジュールによるものと考えられる．

- (図 6.24) 基準に対して時系列データ数と平均トランザクション数が 2 倍のデータセット $D_5$，最小支持度 0.88， スレッド数 1,2,4,6

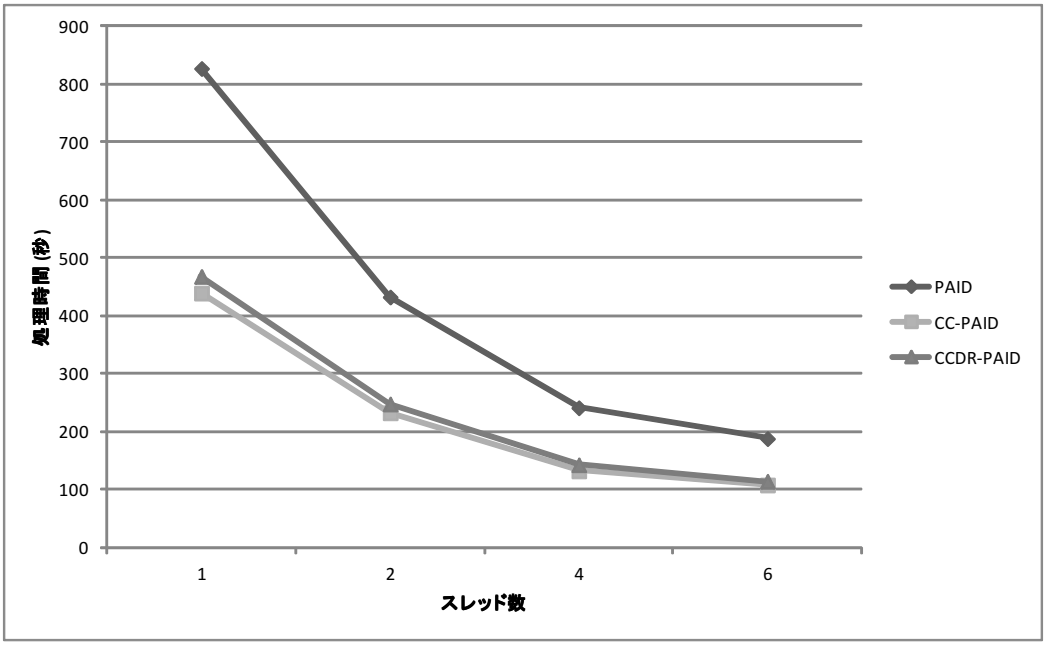
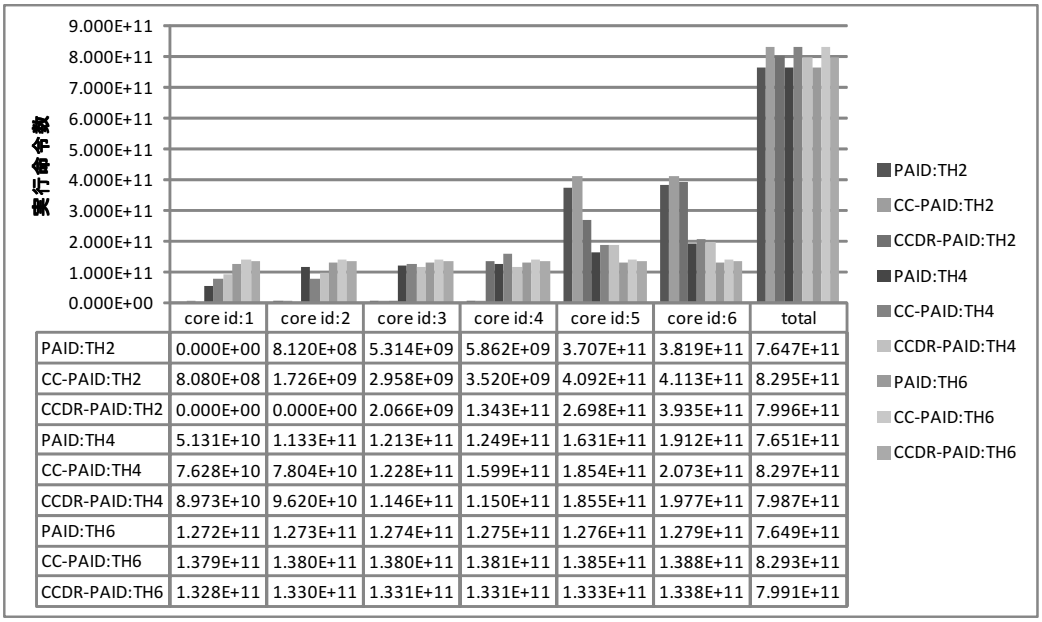



図 6.24 データセット $D_5$ のスレッド数変更時の処理時間

– スケーラビリティ
PAID では，スレッド数 1 に対して 6 の時，4.4 倍の速度向上となった．CC-PAID では，スレッド数 1 に対して 6 の時，4.06 倍の速度向上となった．CCDR-PAID では，スレッド数 1 に対して 6 の時，4.09 倍の速度向上となった．

– (図 6.25) スレッド数 2,4,6 の時の実行命令数
実行命令数の合計値から PAID に対して CC-PAID ではスレッド数 2，4，6 の時，約 8%の実行命令数の増加となった．
CC-PAID に対して CCDR-PAID ではスレッド数 2，4，6 の時，約 4%の実行命令数の削減となった．
また，PAID に対して CCDR-PAID はスレッド数 2 の時，約 5%の実行命令数の増加となり，スレッド数 4，6 の時，約 4%の実行命令数の増加となった．




| | core id:1 | core id:2 | core id:3 | core id:4 | core id:5 | core id:6 | total |
|---|---|---|---|---|---|---|---|
| PAID:TH2 | 0.000E+00 | 8.120E+08 | 5.314E+09 | 5.862E+09 | 3.707E+11 | 3.819E+11 | 7.647E+11 |
| CC-PAID:TH2 | 8.080E+08 | 1.726E+09 | 2.958E+09 | 3.520E+09 | 4.092E+11 | 4.113E+11 | 8.295E+11 |
| CCDR-PAID:TH2 | 0.000E+00 | 0.000E+00 | 2.066E+09 | 1.343E+11 | 2.698E+11 | 3.935E+11 | 7.996E+11 |
| PAID:TH4 | 5.131E+10 | 1.133E+11 | 1.213E+11 | 1.249E+11 | 1.631E+11 | 1.912E+11 | 7.651E+11 |
| CC-PAID:TH4 | 7.628E+10 | 7.804E+10 | 1.228E+11 | 1.599E+11 | 1.854E+11 | 2.073E+11 | 8.297E+11 |
| CCDR-PAID:TH4 | 8.973E+10 | 9.620E+10 | 1.146E+11 | 1.150E+11 | 1.855E+11 | 1.977E+11 | 7.987E+11 |
| PAID:TH6 | 1.272E+11 | 1.273E+11 | 1.274E+11 | 1.275E+11 | 1.276E+11 | 1.279E+11 | 7.649E+11 |
| CC-PAID:TH6 | 1.379E+11 | 1.380E+11 | 1.380E+11 | 1.381E+11 | 1.385E+11 | 1.388E+11 | 8.293E+11 |
| CCDR-PAID:TH6 | 1.328E+11 | 1.330E+11 | 1.331E+11 | 1.331E+11 | 1.333E+11 | 1.338E+11 | 7.991E+11 |

図 6.25 データセット $D_5$ のスレッド数 2,4,6 の時の実行命令数

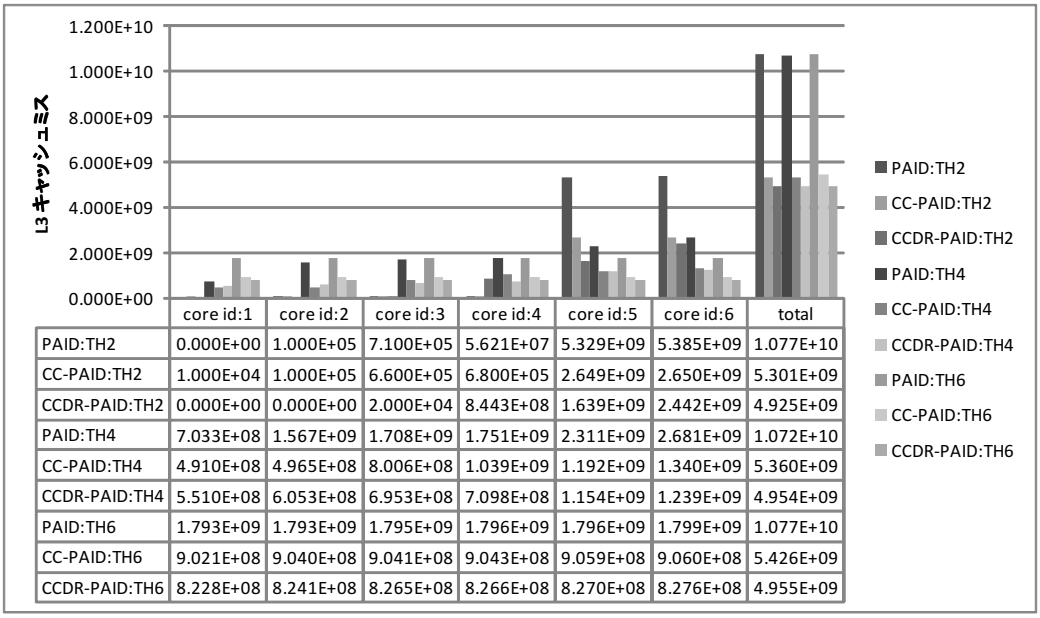
– (図 6.26) スレッド数 2,4,6 の時のキャッシュミス
L3 キャッシュミスの合計値から PAID に対して CC-PAID ではスレッド数 2 の時，約 51%，4 と 6 の時，約 50%の L3 キャッシュミスの削減となった.

CC-PAID に対して CCDR-PAID ではスレッド数 2 の時，約 7%，4 の時，約 8%，6 の時，約 9%の L3 キャッシュミスの削減となった.

また，PAID に対して CCDR-PAID はスレッド数 2，4，6 の時，約 54%の L3 キャッシュミスの削減となった.

**考察**

6 章 2.4 節のシングルスレッド時の PAID に対する CC-PAID の実行命令数と L3 キャッシュミスの変化の割合，CC-PAID に対する CCDR-PAID の実行命令数と L3 キャッシュミスの変化の割合はスレッド数 6 の時とほぼ同じとなり，並列処理時においても CC-PAID に関しては PAID に対して CPU キャッシュの利用効率の




| | core id:1 | core id:2 | core id:3 | core id:4 | core id:5 | core id:6 | total |
|---|---|---|---|---|---|---|---|
| PAID:TH2 | 0.000E+00 | 1.000E+05 | 7.100E+05 | 5.621E+07 | 5.329E+09 | 5.385E+09 | 1.077E+10 |
| CC-PAID:TH2 | 1.000E+04 | 1.000E+05 | 6.600E+05 | 6.800E+05 | 2.649E+09 | 2.650E+09 | 5.301E+09 |
| CCDR-PAID:TH2 | 0.000E+00 | 0.000E+00 | 2.000E+04 | 8.443E+08 | 1.639E+09 | 2.442E+09 | 4.925E+09 |
| PAID:TH4 | 7.033E+08 | 1.567E+09 | 1.708E+09 | 1.751E+09 | 2.311E+09 | 2.681E+09 | 1.072E+10 |
| CC-PAID:TH4 | 4.910E+08 | 4.965E+08 | 8.006E+08 | 1.039E+09 | 1.192E+09 | 1.340E+09 | 5.360E+09 |
| CCDR-PAID:TH4 | 5.510E+08 | 6.053E+08 | 6.953E+08 | 7.098E+08 | 1.154E+09 | 1.239E+09 | 4.954E+09 |
| PAID:TH6 | 1.793E+09 | 1.793E+09 | 1.795E+09 | 1.796E+09 | 1.796E+09 | 1.799E+09 | 1.077E+10 |
| CC-PAID:TH6 | 9.021E+08 | 9.040E+08 | 9.041E+08 | 9.043E+08 | 9.059E+08 | 9.060E+08 | 5.426E+09 |
| CCDR-PAID:TH6 | 8.228E+08 | 8.241E+08 | 8.265E+08 | 8.266E+08 | 8.270E+08 | 8.276E+08 | 4.955E+09 |

図 6.26 データセット $D_5$ のスレッド数 2,4,6 の時の L3 キャッシュミス

表 6.5 メモリ消費量 (MB)

| | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ |
|---|---|---|---|---|---|
| PAID | 270 | 540 | 350 | 410 | 690 |
| CC-PAID | 430 | 820 | 470 | 660 | 900 |
| CCDR-PAID | 540 | 1030 | 580 | 790 | 1080 |

向上により処理時間を短縮し，CCDR-PAID に関しても CC-PAID に対して計算量と L3 キャッシュミスの削減により処理時間が短縮できると考えられる．

## 2.7 メモリ使用量

データセット $D_1$，$D_2$，$D_3$，$D_4$，$D_5$ を用い，アルゴリズム毎のメモリ使用量を計測した．最小支持度は 6 章 2.2 節の変更幅を 0.02 とした結果から，それぞれのデータセットで PAID，CC-PAID，CCDR-PAID の三つのアルゴリズムが計測された最も小さな最小支持度を用い，データセット $D_1$ では 0.52，$D_2$ では 0.56，

表 6.6 データセット $D_1$ 処理時のデータ構造毎のサイズ (MB)

| | position list-DB | ILP-DB | 再構築された ILP-DB |
|---|---|---|---|
| PAID | 130 | 37 | |
| CC-PAID | 130 | 37 | |
| CCDR-PAID | 130 | 37 | 107 |

$D_3$ では 0.86，$D_4$ では 0.3，$D_5$ では 0.88 となっている．表 6.5 に結果を示す.

また，データセット $D_1$ と最小支持度 0.52 を用いて PAID, CC-PAID に関しては position list-DB と ILP-DB のデータサイズ，CCDR-PAID ではそれに加えて再構築された ILP-DB のデータサイズも示す (表 6.6)．なお，再構築された ILP-DB のサイズは再構築されるたびにデータサイズに加算し，メモリ領域を解放したときはデータサイズから差し引き，最もデータサイズが大きくなった値を示す.

**考察**

基準となるデータセット $D_1$ では，PAID に対して CC-PAID は約 59%，CCDR-PAID は CC-PAID に対して約 26%，メモリ使用量が増加した.

基準に対して時系列データ数が 2 倍のデータセット $D_2$ では，PAID に対して CC-PAID は約 52%，CCDR-PAID は CC-PAID に対して約 26%，メモリ使用量が増加した.

基準に対して平均トランザクション数が 2 倍のデータセット $D_3$ では，PAID に対して CC-PAID は約 34%，CCDR-PAID は CC-PAID に対して約 23%，メモリ使用量が増加した.

基準に対してアイテムの種類数が 2 倍のデータセット $D_4$ では，PAID に対して CC-PAID は約 61%，CCDR-PAID は CC-PAID に対して約 20%，メモリ使用量が増加した.

基準に対して時系列データ数と平均トランザクション数が 2 倍のデータセット $D_5$ では，PAID に対して CC-PAID は約 30%，CCDR-PAID は CC-PAID に対して約 20%，メモリ使用量が増加した.

PAID に対して CC-PAID のメモリ使用量が増加した原因は Common-Prefix-At-A-Time により，$common\ prefix$ を持つ長さ $k$ の時系列パターン毎の処理に必要な支持度のデータ等を保持しなくてはならないためと考えられる．

また，Common-Prefix-At-A-Time では CPU キャッシュにフェッチされたデータを最も効率良く利用するために共通する接頭辞を持つ時系列パターン全てを処理対象としているが，SPADE [21] のクラス分割のように，処理対象とする時系列パターン数を調整することにより，メモリ使用量を抑えることも可能と考えられる．しかし，処理対象とする時系列パターン数を調整すると時間的局所性を効率よく向上させることができなくなり，キャッシュミスが増加する可能性が考えられるため，処理速度の観点からは，共通する接頭辞を持つ時系列パターン全てを処理対象とするのが望ましいと考えられる．

CC-PAID に対して，CCDR-PAID のメモリ使用量が増加した主な原因は，表 6.6 の結果から再構築された ILP-DB の影響が大きいと考えられる．この問題に関しては，6 章 2.3 節の考察と同様に閾値を設けて ILP-DB の再構築を抑えることで，メモリ使用量の増加を抑えることが可能と考えられる．

また，PC クラスタを用いた並列処理により，全体的なメインメモリの容量を増加させることで CC-PAID と CCDR-PAID のメモリ使用量の増加に対して対処することも可能と考えられる．

# 第7章　結論

## 1. まとめ

時系列パターンマイニングの分野では，大規模な時系列データベースを対象に処理を行った場合，その処理時間が膨大になるといった問題があり，処理時間を短縮させるアプローチとして3章で紹介した時系列パターンマイニングアルゴリズムの提案や並列化が行われてきた．

これに対し，相関ルールマイニングの分野ではデータ構造の局所性等に着目し，CPU キャッシュの利用効率を向上させることにより処理速度を向上させるアプローチが行われ，アルゴリズムの性質が似ている時系列パターンマイニングでも同じようなアプローチが有効である可能性が考えられた．また，CPU キャッシュに関しては1章に示したようにメモリの壁の問題と処理するデータに対する CPU キャッシュの容量の少なさといった問題があり，時系列パターンマイニングにおいて，CPU のキャッシュミスによるレイテンシがボトルネックとなる可能性に着目した．この他にも，時系列パターンマイニングの計算量の削減とマルチコア CPU を用いた並列処理による処理時間の短縮も重要と考えた．

上記のような背景から時系列パターンマイニングのアルゴリズムの中核を変更することなく CPU キャッシュの利用効率の向上及び計算量とキャッシュミスを削減し，処理速度を向上させることを目的とし，2章で紹介した既存の時系列パターンマイニングアルゴリズムの中でも処理効率の良い PAID アルゴリズムを対象に，研究を行った．

4章では，PAID アルゴリズムにおいて支持度の更新処理に用いられる item-last-position list といったデータ構造の CPU キャッシュの利用効率に着目し，キャッシュミスが生じる可能性について考察を行った．この結果，共通する接頭辞を持つ

時系列パターンが存在する場合，item-last-position list の処理範囲が同じになることに着目し，CPU キャッシュにフェッチされた ILP-list の時間的局所性を向上させるために，共通する接頭辞を持つ時系列パターンで CPU キャッシュにフェッチされた ILP-list の再利用を行う Common-Prefix-At-A-Time の提案を行った．これにより，CPU キャッシュにフェッチされたデータが他の共通する接頭辞を持つ複数の長さ $k$ の時系列パターン間で直ぐに利用されやすくなる等，時間的局所性を向上させることが可能となり，この手法を PAID に適用した CC-PAID アルゴリズムを提案した．

5 章では，CC-PAID のデータアクセスに着目し，データアクセスの効率化により計算量と CPU のキャッシュミスの削減を行う CCDR-PAID アルゴリズムを提案した．これは，ILP-list から長さ $k+1$ の時系列パターンの長さ $k+1$ 番目のアイテムになる可能性のあるアイテムを抜き出し，ILP-list の再構築を行う．再構築された ILP-list を用いることにより，共通する接頭辞を持つ長さ $k$ の時系列パターンが対応する時系列データに含まれているか判断し，含まれていない場合，position list の処理を省略する．支持度の更新に関連した処理では，再構築された ILP-list は処理に不要なデータが取り除かれているため，処理量が少なくなると同時に CPU キャッシュにフェッチされるデータサイズが小さくなる．また，再構築の処理は必要なデータのコピーにより行われ，メインメモリ上のデータ間の距離が近くなるため空間的局所性の向上にも繋がる可能性がある．更に，再構築された ILP-list は次の長さ $k+1$ の時系列パターンの発見処理で動的に再構築が行われる．上記のように，動的に再構築された ILP-list を有効的に利用することにより，CCDR-PAID では，省略可能なデータ構造へのアクセスの省略や処理に不要なデータへのアクセスを極力避けることにより，計算量と CPU のキャッシュミスを削減させる．

CC-PAID と CCDR-PAID の両者のアルゴリズムは，アルゴリズムのコアを変更することなく処理速度を向上させることを目的としたものであり，性能評価からそれらの有効性を確認した．処理時間の結果としては，PAID に対して CC-PAID は最大で約 55%，処理時間の短縮が確認され，CC-PAID に対して CCDR-PAID は最大で約 22%，処理時間が短縮された．実行命令数とキャッシュミスの評価で

は，CC-PAID は，PAID に対して実行命令数が最大で約 8%，増加し，L3 キャッシュミスは最大で約 64%，削減された．CCDR-PAID は CC-PAID に対して実行命令数が最大で約 45%，削減され，L3 キャッシュミスは最大で約 22%，削減された．処理時間と実行命令数とキャッシュミスから，PAID に対する CC-PAID の処理時間が短縮される要因を CPU キャッシュの利用効率の向上とし，CC-PAID に対して CCDR-PAID が処理時間を短縮する要因は計算量とキャッシュミスの削減によるものと考察した．

また，処理時間では，PAID に対して CCDR-PAID は最大で約 64%，処理時間の短縮が確認され，実行命令数では最大で約 43%，実行命令数が削減され，L3 キャッシュミスでは最大で約 72%，L3 キャッシュミスが削減された．更に処理時間を効率的に短縮させるために，PAID，CC-PAID，CCDR-PAID の並列化も行っており，スレッド数 1 対して，6 の時，PAID，CC-PAID，CCDR-PAID はそれぞれ 4 倍以上の速度向上が確認された．

## 2. 今後の課題

6 章の考察から，プログラムの最適化や発見される時系列パターンの特徴等の調査及びより詳細な評価が本研究の今後の課題と考えられる．

また，時系列パターンマイニングに限らず他の分野でも CPU キャッシュの利用効率を向上させる手法の提案は処理速度の向上において重要なアプローチであると考え，本論文で行われたような手法を他の分野で応用可能か検討を行っていく必要がある．

更に，時系列パターンマイニングでは，非常に多くの時系列データと時系列パターンの処理が行われる．そのため，並列処理による処理時間の短縮方法は有効なアプローチと考えられる．今後は多数のコアを持つプロセッサが登場するメニーコアの時代が予測されている [25]．Cell プロセッサ [13] や GPU もまた多くのコアを有しており，このような他のアーキテクチャの特性を活かし，効率的に時系列パターンを発見する手法の提案は処理時間の短縮に有効的である可能性があり，同時に情報爆発の時代においても積極的に取り組むべきであると思われる．

# 謝辞

本研究の主指導教官であるインタラクティブメディア設計学研究室 加藤 博一教授には，講座内の研究会におきまして貴重な御意見を数多く頂きました．また，博士後期課程の学生として私自身に不甲斐無い部分が多くありましたが，寛大な御心で見守っていただき，様々な面で配慮していただけたため，滞り無く本研究を遂行することができました．ここに深く感謝の意を表します．

本研究の副指導教官であるインタラクティブメディア設計学研究室 宮崎 純准教授には，非常に丁寧な研究のご指導をしていただきました．特に本研究に関して，数多くの専門的なアドバイスや御意見を頂きました．また，研究に関しての様々な不安や悩みについても親切に聞いていただき，研究を進める上での視点や考え方，研究者としての心構え等交えながら御意見を頂き，研究者としての考え方や在り方を学ばせて頂けた貴重な体験でした．宮崎 純准教授の的確なご指導が無ければ本研究を遂行することは出来ませんでした．ここに深く感謝の意を表します．

本研究の副指導教員であるコンピューティング・アーキテクチャ研究室 中島 康彦教授には，博士中間発表や公聴会等で貴重なご意見等，親切なご指導受け賜わりました．ここに深く感謝の意を表します．

熊本大学 有次 正義教授には，公聴会等で親切なご指導受け賜わりました．ここに深く感謝の意を表します．

インタラクティブメディア設計学研究室 天野 敏之助教 (現 山形大学工学部准教授)，藤澤 誠助教 (現 筑波大学図書館情報メディア研究科助教) にも講座内の研究会等で様々な意見等を頂きました．感謝の意を表します．

その他，インタラクティブメディア設計学研究室に関わる教員，秘書，先輩，学生の方々にも様々な面でお世話になりました．感謝の意を表します．

岩手県立大学 基盤ソフトウェア学講座 澤本 潤教授，瀬川 典久講師をはじめとする教員，学生の方々にも帰省した際に様々な面でお世話になりました．感謝の意を表します．

瀬川 典久講師には帰省した際に研究の場を与えていただき，研究者とはどういったものかといったこと等を教えていただき，進路についても相談に乗って頂

きました．心より感謝申し上げます．

また，岩手県立大学 分散システム学講座 高田 豊雄教授，王 家宏准教授，児玉 英一郎講師にも帰省した際，研究や学生生活等の様々なお話を聞いていただき，その都度いろいろなアドバイスを頂きました．感謝の意を表します．

王 家宏准教授には，学部時代に時系列パターンマイニングについて教えて頂き，卒業後も帰省した際に研究や進路について様々なアドバイスを頂きました．心より感謝申し上げます．

多くの方々のご協力の元で本研究および本稿の執筆が遂行できたことを，この場をお借りして心より感謝申し上げます．

最後に，父，母，祖母，愛猫達には研究や学生生活などで不安や心配なことがある度に，いつも励まして頂き，私の心の支えとなって頂きました．心より感謝いたします．ありがとうございました．

# 参考文献


[1] Intel vtune amplifier xe 2011. http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, pp. 487–499, Santiago,Chile, 1994.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pp. 3–14, Washington, DC, USA, 1995. IEEE Computer Society.

[4] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pp. 429–435, New York, NY, USA, 2002. ACM.

[5] Clay Breshears. 並行コンピューティング技法実践マルチコア/マルチスレッドプログラミング, p. 46. オライリー・ジャパン, 2009.

[6] Guozhu Dong and Jian Pei. *Sequence Data Mining*, p. 2. Springer, 2007.

[7] Douglas E.Comer. コンピュータアーキテクチャのエッセンス, pp. 222–223. 翔泳社, 2007.

[8] Amol Ghoting, Gregory Buehrer, Srinivasan Parthasarathy, Daehyun Kim, Anthony Nguyen, Yen-Kuang Chen, and Pradeep Dubey. Cache-conscious frequent pattern mining on a modern processor. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pp. 577–588. VLDB Endowment, 2005.

[9] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, Second Edition*, p. 239. Morgan Kaufmann, 2006.

[10] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pp. 1–12, New York, NY, USA, 2000. ACM.

[11] Yuki Matsubara, Jun Miyazaki, Goshiro Yamamoto, Yuki Uranishi, Sei Ikeda, and Hirokazu Kato. Ccdr-paid: More efficient cache-conscious paid algorithm by data reconstruction. *ACM Symposium on Applied Computing (ACM SAC2012)*, March 2012 (Accepted).

[12] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Trans. on Knowl. and Data Eng.*, Vol. 16, pp. 1424–1440, November 2004.

[13] Matthew Scarpino. *Programming the Cell Processor: For Games, Graphics, and Computation*. Prentice Hall, 2008.

[14] Takahiko Shintani and Masaru Kitsuregawa. Mining algorithms for sequential patterns in parallel: Hash based approach. In *Proceedings of the Second Pacific Asia Conference on Knowledge Discovery and Data mining*, pp. 283–294, 1998.

[15] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '96, pp. 3–17, London, UK, 1996. Springer-Verlag.

[16] Jiahong Wang, Yoshiaki Asanuma, Eiichiro Kodama, Toyoo Takata, and Jie Li. Mining sequential patterns more efficiently by reducing the cost of scanning sequence databases. *IPSJ Digital Courier*, Vol. 2, pp. 768–782, 2006.

[17] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News*, Vol. 23, pp. 20–24, March 1995.

[18] Zhenglu Yang, Masaru Kitsuregawa, and Yitong Wang. Paid: Mining sequential patterns by passed item deduction in large databases. In *Proceedings of the 10th International Database Engineering and Applications Symposium*, pp. 113–120, Washington, DC, USA, 2006. IEEE Computer Society.

[19] Zhenglu Yang, Yitong Wang, and Masaru Kitsuregawa. Effective sequential pattern mining algorithms for dense database. In *Proceedings of Data Engineering Workshop(DEWS2006)*, 2006.

[20] Mohammed J. Zaki. Parallel sequence mining on shared-memory machines. In *Journal of Parallel and Distributed Computing*, pp. 401–426. Springer-Verlag, 2001.

[21] Mohammed J. Zaki. Spade: an efficient algorithm for mining frequent sequences. In *Machine Learning Journal, special issue on Unsupervised Learning*, pp. 31–60, 2001.

[22] D. M. クロエンケ. データベース処理 基礎・設計・実装, p. 548. 株式会社トッパン, 1996.

[23] 株式会社フィックスターズ. OpenCL 入門　マルチコア CPU・GPU のための並列プログラミング, p. 18. インプレスジャパン, 2010.

[24] 高木允, 田村慶一, 周藤俊秀, 北上始. Modified prefixspan 法の並列化と動的負荷分散手法. 情報処理学会論文誌. 数理モデル化と応用, Vol. 46, No. 10, pp. 138–152, 2005.

[25] 安田絹子, 飯塚博道, 青柳信吾, 小林林広, 阿部貴之. マルチコア CPU のための並列プログラミング―並列処理&マルチスレッド入門, p. 36. 秀和システム, 2006.

[26] 奥川峻史. Bit ： コンピュータ・サイエンス誌. 高速化技術のキー「RISC」はやわかり, 第 24 巻, pp. 51–65. 共立出版, 1992.

[27] 松原裕貴, 宮崎純, 藤澤誠, 天野敏之, 加藤博一. Cc-paid：cpu キャッシュを有効利用した並列時系列パターンマイニングアルゴリズム. 情報処理学会論文誌データベース（TOD）, Vol. 4, No. 2, pp. 88–100, 2011.

# 研究業績

A. 査読付き論文誌

[A1] 松原 裕貴，宮崎 純，藤澤 誠，天野 敏之，加藤 博一, CC-PAID：CPU キャッシュを有効利用した並列時系列パターンマイニングアルゴリズム，情報処理学会論文誌データベース (TOD),Vol.4, No.2, pp.88-100 (2011).

B. 査読付き国際会議

[B1] Yuki Matsubara, Jun Miyazaki, Goshiro Yamamoto, Yuki Uranishi, Sei Ikeda, and Hirokazu Kato, CCDR-PAID: More Efficient Cache-Conscious PAID Algorithm by Data Reconstruction, ACM Symposium on Applied Computing (ACM SAC2012), March 2012 (Accepted).

C. 査読なし国内発表

[C1] 松原 裕貴, 宮崎 純, 加藤 博一, CPU キャッシュを有効利用した並列時系列パターンマイニングアルゴリズム Cache-conscious parallel PAID の提案, 第 151 回 データベースシステム研究発表会, 2010 年.

[C2] 松原 裕貴, 宮崎 純, 藤澤 誠, 天野 敏之, 加藤 博一, CC-PAID におけるデータベースサイズと CPU キャッシュ利用効率の関係について, 第 3 回データ工学と情報マネジメントに関するフォーラム, 2011 年.

[C3] 松原 裕貴, 宮崎 純, 山本 豪志朗, 浦西 友樹, 池田 聖, 加藤 博一, データアクセスの改良による時系列パターンマイニングアルゴリズムの高速化, 第 153 回データベースシステム研究発表会, 2011 年.