VSOPのruby 拡張ライブラリについて

２０１０年5月２８日

ERATO湊離散構造処理系プロジェクト
統計・マイニング応用グループ 技術員 中元政一

背景と目的

背景

情報大公開プロジェクトにおいて、データマイニングツールであるIGVminerをRubyで開発。このIGVminer内部でVSOP(ZDD)を使用する必要があった

目的

RubyでVSOP(ZDD)を使えるようにすることによって、VSOP(ZDD)をシームレスにruby上で利用可能になる

Rubyを使用した理由

日本で普及しているRubyで仕様を固め、 次の段階でPerlやRythonの他言語への移植を考えた。

記述例

VSOP

```
A= a b c
B= 4
C = A * B
print C

D=Lcm("FQ","tra.txt",2)
D.restrict(x1 x2).show
```

RUBY拡張　ruby 1.8.6 を使用

```
require "rubygems"
require "vsop"
A=VSOP.itemset("a b c")
B=VSOP.constant(4)
C = A * B
C.show

D=VSOP.lcm("FQ","tra.txt",2)
D.restrict("x1 x2")
```

- vsopオブジェクトを作成および出力をのぞけば、ほぼ同じ記述で処理を行える
- vsopオブジェクトをrubyオブジェクトとして作成しているので、rubyの制御構造(while,ifなど)使用できる
- vsopではITEM名を[a-z][a-zA-z0-9_]*にする必要があったがruby拡張では意識する必要がない
- rubyの文字列や数値あるいはCSVファイルを相互に変換可能である

VSOPの命令とruby拡張ライブラリのメソッドとの対応表

VSOP	RUBY拡張
Maxval	maxval
Minval	minval
TotalVal	totalval
Items	items
TermsEQ	termsEQ
TermsNE	termsNE
TermsGT	termsGT
TermsGE	termsGE
TermsLT	termsLT
TermsLE	termsLE
Restrict	restrict
Permit	permit
Permitsym	permitsym
FreqPatA	freqpatA
FreqPatM	freqpatM
FreqPatC	freqpatC
SymGrp	symgrp

VSOP	RUBY拡張
*	*
/	/
%	%
+	+
+(単項演算子)	+(単項演算子)
-	-
-(単項演算子)	-(単項演算子)
==	==
>	>
>=	>=
<	<
<=	<=
!=	ne?
?:	iif
Import	import
Lcm	lcm

VSOP	RUBY拡張
print	show
print /bit	bit
print /hex	hex
print /map	map
print /rmap	rmap
print /case	case
print /size	size
print /count	count
print /density	density
print /value	value
print /maxcover	maxcover
print /maxcost	maxcost
print /mincover	mincover
print /mincost	mincost
print /decomp	decomp
print /nodesize	nodesize
print /export	export

VSOP	RUBY拡張
print /plot	未対応
print /plot0	未対応
symbol	未対応
RUBY拡張固有	itemset
RUBY拡張固有	constant
RUBY拡張固有	lcm_ub
RUBY拡張固有	csvout
RUBY拡張固有	hashout
RUBY拡張固有	coerce
RUBY拡張固有	to_i

RUBY拡張固有メソッド

itemset
機能:引数で指定したアイテムから新しいVSOPオブジェクトを作成
書式:s = VSOP.itemset("a b c")

constant
機能: 引数で指定した整数から新しいVSOPオブジェクトを作成
書式:c = VSOP.constant(10)

corece
機能: rubyオブジェクトからVSOPオブジェクトを作成
書式:s = VSOP.itemset(rubyobj)

to_i
機能:VSOPオブジェクトから数値を生成（VSOPオブジェクトが数値の場合のみ）
書式:val = c.to_i

lcm_ub
機能:パターン長制約を指定できるlcm over zdd
書式:VSOP. lcm_ub("[FMC]",tra_filename,minimum support,unper bound,order_filename)

csvout
機能:VSOPオブジェクトを指定したファイルにCSV出力
書式:s.csvout(outfilename)

出力例(重み,itemset)

```
2,a b
3,c d
2,e
```

← 2 a b + 3 c d +2e

hashout
機能:VSOPオブジェクトをrubyのハッシュオブジェクトへ変換
書式:s.hashout

使用例

rubyのハッシュオブジェクトに
変換する事により
h["a"]とする事で
"a"に対する重みを取得できるようになる

使用例(1)

- taxonomyの冗長パターンの削除

前述のIGVminerで使用した記述の抜粋であり、
lcmで計算した頻出パターンから
親子関係のあるitemのセットを含むパターンを取り除いている

```
pat = VSOP.lcm("FQ","tradata",4)
taxo.each{|fldVal|
  oyako1=VSOP.constant(0)
  items=fldVal.split
  items.each{|item1|
    items.each{|item2|
      if item1!=item2 then
        oyako=oyako+VSOP.itemset("#{item1} #{item2}")
      end
    }
  }
}
pat=pat.restrict(oyako).iif(0,pat)
```

親子関係表

子	親
a	X
b	Y

上記の親子関係表を使用して a b x y の組み合わせを考えるととなる
冗長である部分を取り除く必要が有る

a	b	X	Y
a X	a Y	b X	b Y
a b X	a b Y	a X Y	b X Y
a b X Y			

青色の部分が冗長な部分

使用例（2）

- n-queen問題

NXNのマス上にN個のQUEENを互いにとら
れないように配置するパターンを求める

```
require "rubygems"
require "vsop"
n = 8
n = ARGV[0].to_i if ARGV[0]
s_base=VSOP.constant(0)
s_new=VSOP.constant(0)
(1..n).each{|i|
    (1..n).each{|j|
    tmp = s_base
    (1...i).each{|k|
      tmp %= ("x#{i-k}y#{j}") if i-k > 0
      tmp %= ("x#{i-k}y#{j-k}") if i-k > 0 && j-k>0
      tmp %= ("x#{i-k}y#{j+k}") if i-k > 0 && j+k<=n
    }
    if i>1 then
      s_new += (tmp*"x#{i}y#{j}")
    else
      s_new += VSOP.itemset("x#{i}y#{j}")
    end
  }
  s_base=s_new
  s_new=VSOP.constant(0)
}
s_base.show
s_base.csvout("x.csv")
```

参考文献：Minato, Shin-ichi,Zero-suppressed BDDs and their applications,2001

今後について

問題点（開発上の課題）

- ソース整理
- データの前処理
- 異なるデータのZDDでの組み合わせ
- VSOPで重みに小数をつかえるようにする

応用課題

- OLAP (online analytical processing) への応用
- ZDDのstream

ソースの整理

現状、sed変換により元々あるソースの変換をおこない実装している。今後の事を考えると、この変換処理をなくし、ソースを整理していく必要が有る

symbol名の変換処理

使用者がデータを [a-z][a-zA-z0-9_]*の形へ変換する必要がある
(LCM over zddを使用する場合には[0-9]|[1-9][0-9]*)

ruby拡張の場合、現在は
LCM over zddを使用するときのみ変換必要

変換テーブル	
リンゴ	a
もも	b
梨	c
葡萄	d
いちご	e

リンゴ	もも	梨
葡萄	梨	
もも	いちご	
いちご		
リンゴ	葡萄	

a	b	c
d	c	
b	e	
e		
a	d	

変換テーブル	
リンゴ	1
もも	2
梨	3
葡萄	4
いちご	5

1	2	3
4	3	
2	5	
5		
1	4	

LCM over ZDDの使用時の

異なるデータのZDDでの組合せ

異なるデータでLCM over ZDDを使用する際、アイテムの読み替えが発生する

1 2 3 4 5 6 7 8 9
1 2 3 4 5 7 8 9
1 2 3 6
1 2 4 5 6 8
1 2 5 8
1 2 9
1 2 4 8 9

最小サポート4で頻出集合を求める →

4 x5 x8 x2 x1 + 4 x5 x8 x2 + 4 x5 x8 x1 + 4 x5 x8 + 4 x5 x2 x1 + 4 x5 x2 +
4 x5 x1 + 4 x5 + 4 x4 x8 x2 x1 + 4 x4 x8 x2 + 4 x4 x8 x1 + 4 x4 x8 +
4 x4 x2 x1 + 4 x4 x2 + 4 x4 x1 + 4 x4 + 4 x9 x2 x1 + 4 x9 x2 + 4 x9 x1 +
4 x9 + 5 x8 x2 x1 + 5 x8 x2 + 5 x8 x1 + 5 x8 + 7 x2 x1 + 7 x2 + 7 x1 + 7

1 2 3 4 5 6 7 8 9
1 5 7 9
1 3 5 7 9
1 3 4 10
1 3 4 5 6 10
1 10
1 10
1 3 4 5

4 x4 x1 + 4 x4 + 4 x9 x8 x1 + 4 x9 x8 + 4 x9 x2 x1 + 4 x9 x2 + 5 x9 x1+
5 x9 + 5 x8 x1 + 5 x8 + 4 x2 x1 + 4 x2 + 8 x1 + 8

x4→x10,x8→x5,x9→x3,x2→x4と読み替える

4 x10 x1 + 4 x10 + 4 x3 x5 x1 + 4 x3 x5 + 4 x3 x4 x1 + 4 x3 x2 + 5 x3 x1+
5 x3 + 5 x5 x1 + 5 x5 + 4 x4 x1 + 4 x4 + 8 x1 + 8

ZDDで小数をデフォルトで対応

小数桁数を固定すれば現状でも対応可能だが、前処理、後処理が必要となる。またOLAPでは小数の処理が多くなると考えられ、デフォルトで小数に対応した方がよいと考える。

OLAP(online analytical processing)への応用

多次元データをZDDで作成

key1	key2	key3	val
a	A	X	5
a	A	Y	2
a	A	Z	5
a	B	X	6
a	B	Y	8
a	B	Z	10
b	A	X	3
b	A	Y	14
b	A	Z	13
b	B	X	6
b	B	Y	30
b	B	Z	31

DB =5 a A X + 2 a A Y + 5 a A Z + 12 a A + 6 a B X + 8 a B Y + 10 a B Z + 24 a B + 11 a X + 10 a Y + 15 a Z + 36 a + 3 b A X + 14 b A Y + 13 b A Z + 30 b A + 6 b B X + 30 b B Y + 31 b B Z + 67 b B + 9 b X + 44 b Y + 44 b Z + 97 b + 8 A X + 16 A Y + 18 A Z + 42 A + 12 B X + 38 B Y + 41 B Z + 91 B + 20 X + 54 Y + 59 Z + 133

keyによる分析が可能
複数のZDDを作成する事で複数valに対応可能
小数桁を固定すれば実数対応可能

ITEMに属性を持たせることで、より詳細な分析が可能
例）ドリルダウン

参考論文：Minato, Shin-ichi,Zero-suppressed BDDs and their applications,2001

ZDDのstream

ZDDをstream化する事で並列処理することが可能となり、効率よく処理する事が可能。
例えば、ある企業の支店ごとのデータ並列処理し、集める事がファイル出力なしに実行可能になる

参考論文：湊真一、石原晋也　BDDの規模によらず一定の実記憶の範囲内で動作するストリーム形式BDD処理アルゴリズム.1999

今後についてのまとめと計画

以上の内容を実行していくにはZDD、LCMのアルゴリズムおよび実装方法について、深い理解が必要である。
まずは理解し、その後、整理、および応用へとつなげていく。