フロンティア法プログラム version 1.0.1

2017 年 9 月 28 日

1　はじめに

フロンティア法は、与えられたグラフ上において、指定された 2 点間のパスや全域木、連結成分などを表現する ZDD を構築するアルゴリズムである。構築した ZDD を基に、全解の列挙や数え上げ、ランダムサンプリングなどを行える。

以下では、パスや全域木など、ZDD で表現したい部分グラフを構築対象と呼ぶ。

2　できること

- 入力グラフに対して、以下の部分グラフ（構築対象）の全てを表現する ZDD の構築
  - 全域森
  - 全域木
  - 無向 s-t パス（サイクル）
  - 有向 s-t パス（サイクル）
  - （無向）辺素 s-t パス（サイクル）
  - 根付き森
  - 連結成分
  - k-カット （高々 k 本のカット）
  - 源を指定したカット
- 入力ハイパーグラフに対して、以下の部分ハイパーグラフ（構築対象）の全てを表現する ZDD の構築
  - 集合分割
  - 集合被覆
  - 集合パッキング
- 構築した ZDD の既約化（reduced and ordered ZDD を得る）
- 構築した ZDD の解の個数のカウント
- 構築した ZDD から全解を列挙
- 構築した ZDD から解の一様ランダムサンプリング
- 入力グラフの頂点番号を幅優先順に付け替え

- 入力グラフ、出力 ZDD の graphviz 形式 [*1]による出力

3 ビルドと実行

3.1 準備

64 ビット整数（32 ビット整数）で表現できない大きな数を扱うときは、GNU MP ライブラリ（http://gmplib.org/）をあらかじめインストールしておくと良い。ただし、インストールしなくても差し支えは無い（5.4 節参照）。

3.2 ビルド

./configure と make によって導入する。

```
./configure
make
```

オプションを指定しない場合、64 ビット版がビルドされる。32 ビット版をビルドする場合は以下で示すようにオプションを与える。

```
./configure --enable-32bit
make
```

64 ビット CPU 環境が一般的になってきたため、32 ビット版の動作はサポートしない。

src/frontier ディレクトリに frontier、 enumpath プログラムと libfrontier.a ライブラリファイルが生成される。また、utility ディレクトリに makegrid プログラムが生成される。make install コマンドでこれらのプログラムをインストールすることができる。

説明の都合上、プログラムを展開したディレクトリの直下にシンボリックリンクを張る。

```
ln -s src/frontier/frontier frontier
ln -s src/utility/makegrid makegrid
```

makegrid プログラムは $m \times n$ グリッドグラフを生成する。frontier プログラムの入力として用いることができる。

実行例（2×3 グリッドグラフの生成）

```
./makegrid 2 3
```

出力

```
2 4
1 3 5
2 6
```

[*1] グラフ描画プログラム。http://graphviz.org/

```
1 5
2 4 6
3 5
```

3.3 実行例

3×3 グリッドの全域森 ZDD を構築するには、以下のコマンドを実行する。

```
./makegrid 3 3 | ./frontier -t sforest
```

標準出力の結果

```
#1:
2:3,4
#2:
3:5,5
4:5,6
#3:
5:7,8
...
```

上記の出力は ZDD を表す。出力の見方は 5.1 節を参照。

標準エラー出力の結果

```
# of nodes of ZDD = 45
# of solutions = 3102
```

構築した ZDD の既約化前のノード数は 45、解の数（ZDD が表現する集合族の要素の個数）は 3,102 であることがわかる。

ZDD を既約化するには `-r` オプションを用いる。

```
./makegrid -d 3 3 | ./frontier -t sforest -r
```

出力結果（ZDD 本体は省略）

```
# of nodes of ZDD = 48
# of nodes of reduced ZDD = 41
# of solutions = 3102
```

s-t パス ZDD を構築するには `-t stpath` オプションを用いる。以下のコマンド例は、3×3 グリッドの 1 つの角から対角の角までの全パスの ZDD を構築する。始点を指定しない場合は頂点番号 1、終点を指定しない場合は最も大きな頂点番号を指定したことになる。3×3 グリッドの場合は始点は頂点 1、終点は頂点 9 となる。

```
./makegrid -d 3 3 | ./frontier -t stpath
```

s-t パスの始点番号は `-s` オプションで、終点番号は `-e` オプションで指定する。

```
./makegrid -d 3 3 | ./frontier -t stpath -s 2 -e 8
```

4　入力書式

`frontier` プログラムに与える入力は、無向グラフ、有向グラフ、ハイパーグラフのいずれかであり、`-t` オプションで指定した構築対象によって異なる。いずれの形式でも、頂点番号、辺の番号はともに 1 から始まる。

4.1　無向グラフの入力書式

無向グラフを入力する場合、隣接リスト形式または辺リスト形式のテキストで表されたグラフを入力する。隣接リスト形式を指定する場合はオプション無しであり、辺リスト形式を用いる場合は `-c` オプションを指定する。

隣接リスト形式では i 行目に頂点 i に隣接する頂点の番号を並べる。隣接リスト形式では、並列辺（parallel edge）は無視される。すなわち、頂点 i から j に張られた辺が複数存在する入力を与えた場合も、1 本しか存在しないグラフに変換される。したがって、i 行目に j が記述され、j 行目に i が記述されているとき、入力を読み込んで作成されるグラフでは、i, j 間には辺が 1 本だけ存在する（j 行目の i の記述は無視されると考える）。辺の番号は、無視される辺を除いて、隣接リストでの番号の出現順に $1, 2, 3, \ldots$ と振られる（フロンティア法ではこの順で辺が処理される）。隣接リスト形式では、辺の番号（順番）を自由に指定することはできない。辺の番号を指定する場合は辺リスト形式を用いる必要がある。

辺リスト形式では、1 行目に頂点の個数を書き、2 行目以降の各行に辺の始点と終点番号を並べる。1 行目の頂点の個数は省略することができる。この場合は、辺に現れる頂点番号の最大値が頂点の個数となる。辺リスト形式では、並列辺は無視されない。

図 1 のグラフを例に考える。

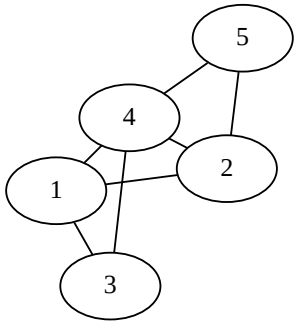

図 1　入力グラフの例

図 1 のグラフを表現する隣接リストは以下の通りである。

```
2 3 4
4 5
4
```

```
5
```

このとき、i 番の辺 e_i は、$e_1 = \{1,2\}$, $e_2 = \{1,3\}$, $e_3 = \{1,4\}$, $e_4 = \{2,4\}$, $e_5 = \{2,5\}$, $e_6 = \{3,4\}$, $e_7 = \{4,5\}$ となる。

図 1 のグラフを表現する辺リストは以下の通りである。

```
5
1 2
1 3
1 4
2 4
2 5
3 4
4 5
```

辺に重みを付けることも可能である。3 番目の数字が辺の重みとなる。3 番目の数字が省略された場合、重みは 1 とみなされる。

```
5
1 2 3.2
1 3 4.5
1 4 1.7
2 4 9.6
2 5 12.0
3 4 23.1
4 5 32.0
```

辺に重みを付ける場合、隣接リストは用いることができない。

4.2 有向グラフの入力書式

入力が有向グラフの場合、... （未稿）

4.3 ハイパーグラフの入力書式

辺リスト形式でハイパーグラフを表現する。それ以外の形式は現在のところ対応していない。`-c` オプションを用いる。

```
4
1 2 3
2
2 4
```

1 行目は頂点の数を表す。2 行目以降は 1 つの行が 1 つのハイパー辺を表す。ハイパー辺が含む頂点の番号を空白区切りで並べる。

4.4 入力グラフの確認

用意した入力が意図した通りに `frontier` プログラムに読み込まれるかを確認するために、`--print-graph-al`, `--print-graph-el`, `--print-graphviz` オプションが用意されている。`--print-graph-al` は入力グラフを隣接リスト形式で、`--print-graph-el` は入力グラフを辺リスト形式で出力する。このオプションを用いることで、隣接リスト形式と辺リスト形式の相互変換も可能である。6 節のオプションリストも参照のこと。

例：入力グラフを辺リスト形式で標準出力に出力する。

```
./makegrid 3 | ./frontier --print-graph-el -
```

例：入力グラフを隣接リスト形式でファイル xxx.txt に出力する。

```
./makegrid 3 | ./frontier --print-graph-al xxx.txt
```

`--print-graphviz` オプションは入力グラフを graphviz 形式で出力する。graphviz をインストールしていれば、グラフを描画することができる。辺の番号を確認するときなどに使うことができる。

例：入力グラフを graphviz で描画（png フォーマットで出力）

```
./makegrid 3 | ./frontier --print-graphviz - | neato -Tpng -o xxx.png
```

出力結果は図 2 のようになる。

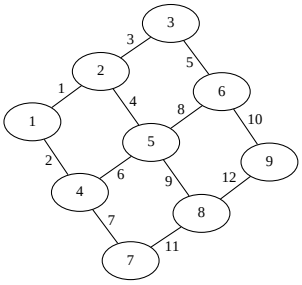

図 2　graphviz 形式による出力例

5　出力書式

5.1 ZDD の出力

`frontier` プログラムは、標準出力に ZDD を出力する。

出力 ZDD の例

```
#1:
```

```
2:3,4
#2:
3:5,5
4:5,6
#3:
5:7,8
...
```

`#i:` の行は、変数（辺）番号 `i` の変数が始まることを意味する。`p:q,r` の行は、ノード番号 `p` の 0-枝の先がノード番号 `q` に、1-枝の先がノード番号 `r` に接続していることを意味する。0-終端は番号 0、1-終端は番号 1 であり、終端以外のノードは 2 以上の番号をもつ。ノード番号はデフォルトでは 10 進数であるが、`--hex` オプションを与えると 16 進数（Knuth の形式）にすることができる。

プログラムに `-r` オプションを与えると既約な ZDD が出力される。

5.2 graphviz による ZDD の描画

graphviz プログラムを用いて ZDD を描画することができる。`--print-zdd-graphviz` オプションで graphviz 形式の ZDD を出力できる。

例：*s*-*t* パスを表す ZDD を graphviz で描画（png フォーマットで出力）

```
./makegrid 3 | ./frontier -t stpath -n --print-zdd-graphviz - | dot -Tpng -o xxx.png
```

出力結果は図 3 のようになる。

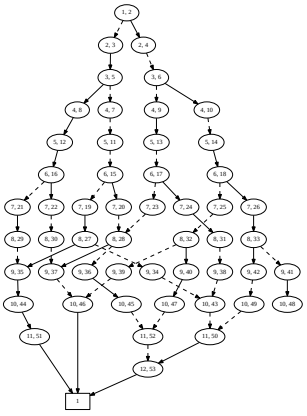

図 3　graphviz 形式による ZDD の描画例

5.3 要素の列挙、一様ランダムサンプリング

`--enum` オプションで ZDD が表現する集合族の全集合の列挙、`--sample` オプションで全集合から一様にランダムサンプリングができる。

例：*s*-*t* パス（の辺集合）を全列挙

```
./makegrid 3 | ./frontier -t stpath -n --enum -
```

出力例

```
1 3 5 6 7 8 11 12
1 3 5 8 9 12
1 3 5 10
1 4 6 7 11 12
1 4 8 10
1 4 9 12
2 3 4 5 6 10
2 3 4 5 7 9 10 11
2 6 8 10
2 6 9 12
2 7 8 9 10 11
2 7 11 12
```

1 行が 1 つの集合に対応する。上記の出力は $\{1,3,5,6,7,8,11,12\},\{1,3,5,8,9,12\},\ldots$ を表す。

例：*s-t* パスの一様ランダムサンプリング（10 個）

```
./makegrid 3 | ./frontier -t stpath -n --sample - 10
```

5.4 計算精度

ZDD の解の個数が 64 ビットまたは 32 ビットで表現できる数（$2^{64}-1$ または $2^{32}-1$）を超える数を扱うには、GNU MP ライブラリ、または、同梱の BigInteger を用いる。

本プログラムのビルド時（./configure 実行時）GMP ライブラリがインストールされていると、自動的にリンクされ、使用可能となる。特別な設定を何もしなくても（オプションを何もつけなくても）GMP ライブラリが使用される。

GMP ライブラリがインストールされていない場合、BigInteger が使用される。

以下で説明するオプションは、特殊な事情が無い限り付加する必要は無いと思われる。`--sm` オプションを用いると GMP ライブラリが、`--sb` オプションを用いると BigInteger が強制使用される。`--sd` オプションを用いると、double 型による計算を行う。`--si` オプションを用いると、64 ビットまたは 32 ビット整数型による計算を行う。`--si` オプションは、メモリの使用量は最も少ないが、計算結果のオーバーフローが起こる可能性があり、大きな数を正しく扱えないことに注意。

旧バージョンで使用していた apfloat ライブラリは廃止された。代わりに GMP ライブラリを用いる。

6 オプション

6.1 構築対象

構築対象は `-t` *kind* の形式で指定する。*kind* に指定可能な対象を表 1 に示す。

表 1　構築対象の指定

kind	入力	構築対象
`sforest`	無向グラフ	全域森 (spanning forest)
`stree`	無向グラフ	全域木 (spanning tree)
`stpath`	無向グラフ	無向 *s*-*t* パス (*s*-*t* path)
`pathmatching` or `pm`	無向グラフ	パスマッチング（複数パス）
`mtpath` or `numberlink`	無向グラフ	複数終端対パス (multi terminal paths)
`dstpath`	有向グラフ	有向 *s*-*t* パス (directed *s*-*t* path) 現バージョンでは未対応
`stedpath`	無向グラフ	（無向）辺素 *s*-*t* パス (edge-disjoint *s*-*t* path) 現バージョンでは未対応
`rforest`	無向グラフ	根付き森 (rooted forest)
`comp`	無向グラフ	連結成分 (component)
`kcut`	無向グラフ	*k*-カット (*k*-cut)
`rcut`	無向グラフ	源を指定した *k*-カット (terminal *k*-cut)
`setpt`	ハイパーグラフ	集合分割 (set partition)
`setc`	ハイパーグラフ	集合被覆 (set cover)
`setpk`	ハイパーグラフ	集合パッキング (set packing)

6.2　オプション

プログラムのオプションを表 2, 3 に示す。表の「使用可能構築対象」は、そのオプションが使用可能な構築対象である。例えば、`--cycle` オプションは無向パス、有向パス列挙時（*kind* が `stpath` または `dstpath`）にのみ使用可能である。一部のオプションでは引数に *filename* （ファイル名）を指定することがあるが、`-` を指定することで標準出力に出力することができる。

7　更新履歴

- ver 1.0.0 2016/3/19 新版公開
- ver 1.0.1 2016/9/27 有向 s-t パスに対応

表 2　プログラムのオプション

オプション	デフォルト	使用可能構築対象	効果
`-t` *kind*	`sforest`	全て	列挙の種類を指定する。*kind* に指定する内容は表 1 を参照。
`-b`	off	入力がグラフである構築対象全て	入力されたグラフについて、始点からの幅優先探索によって頂点番号を付け替える。
`--input-al`	on	入力がグラフである構築対象全て	入力として、隣接リスト形式を用いる。
`-c` or `--input-el`	off	入力がグラフである構築対象全て	入力として、辺リスト形式を用いる。
`-w` or `--weight` *filename*	指定しない	全て	辺に重みを設定する。ファイル *filename* には数字（double 型）の列を記入する。前から順に辺に重みが設定される。書かれた数字が辺の個数より少ない場合、残りの辺には 1 の重みが設定される。入力グラフに記述された重みより、本オプションで指定した重みが優先される。
`-s` N	1	`stpath`, `dstpath`	始点の頂点番号を N に設定する。
`-e` N	最大頂点番号	`stpath`, `dstpath`	終点の頂点番号を N に設定する。
`-h` or `--hamilton`	off	`stpath`, `dstpath`	*s-t* パス、サイクル列挙時にハミルトンパス、サイクルを列挙する。
`--cycle`	off	`stpath`, `dstpath`	サイクルを列挙する。
`--any`	off	`stpath`, `dstpath`	始点から任意の頂点へのパスを列挙する。
`-k` or `--upper` D	∞	`comp`, `kcut`, `rcut`, `stpath`	カット列挙において、カットの数を指定。連結成分列挙において、連結成分の数を指定。また、パス列挙において、パスの長さの上限を指定。`--le` オプションを指定すると重さの総和が高々 D の対象を列挙。`--me` オプションを指定すると重さの総和が少なくとも D の対象を列挙。`--le --me` のいずれも指定しない場合は重さの総和がちょうど D の対象を列挙。
`--le`	off	`comp`, `kcut`, `rcut`, `stpath`	`-k` オプションと同時に指定することで、重さの総和が高々 D の対象を列挙。
`--me`	off	`comp`, `kcut`, `rcut`, `stpath`	`-k` オプションと同時に指定することで、重さの総和が少なくとも D の対象を列挙。

表 3　プログラムのオプション（続き）

オプション	デフォルト	使用可能構築対象	効果
`-f` r_1 r_2 $\cdots$	指定なし	`rforest`, `rcut`	根付き木、源を指定したカットの列挙において、根や源の頂点番号 $r_1, r_2, \ldots$ を指定。
`--root` *filename*	指定なし	`rforest`, `rcut`	根付き木、源を指定したカットの列挙において、根や源の頂点番号を指定。`-f` オプションとは異なり、ファイルから読み込む。ファイル *filename* には数字（double 型）の列を記入する。
`-r`	off	全て	構築した ZDD を既約化する。
`-n` or `--no-print-zdd`	off（出力する）	全て	ZDD を標準出力に出力しないようにする。`>/dev/null` と同様だが、本オプションの方が高速。
`--terminal` *filename*	off	`mtpath`	複数終端対パスの列挙において、終端対を記述したファイルを指定する。ファイル *filename* には数字（int 型）の列を、始点と終点を交互に記入する。
`--print-zdd-graphviz` *filename* [`0`]	off（出力しない）	全て	ZDD を graphviz 形式でファイル（標準出力）に出力する。第 2 引数に `0` を指定すると 0-終端も印字する（指定しない場合は 0-終端は印字されない）。
`--print-zdd-sbdd` *filename* [`0`]	off（出力しない）	全て	ZDD を SapporoBDD library の Import 関数で読み込める形式でファイル（標準出力）に出力する。
`--no-solution`	off（計算する）	全て	ZDD の解の個数を計算しない。
`--print-graph-al` *filename*	off	全て	入力グラフを隣接リスト形式でファイル（標準出力）に出力して、プログラムを終了する。デバッグ用。
`--print-graph-el` *filename*	off	全て	入力グラフを辺リスト形式でファイル（標準出力）に出力して、プログラムを終了する。デバッグ用。
`--print-graphviz` *filename*	off	全て	入力グラフを graphviz 形式でファイル（標準出力）に出力して、プログラムを終了する。

表 4　プログラムのオプション（続き）

オプション	デフォルト	使用可能構築対象	効果
`--enum` *filename*	off	全て	ZDD の全解をファイル（標準出力）に出力する。出力形式は下記参照。
`--sample` *filename* [N]	off	全て	ZDD の全解から一様ランダムサンプリングを行い、N 個の解をファイル（標準出力）に出力する。N のデフォルト値は 100。出力形式は下記参照。
`--hex`	off	全て	ZDD の出力時、ID 番号を 16 進数にする（Knuth の形式）。
`--am`	off	全て	ZDD をオートマトンに変換する。
`--print-am`	off	全て	オートマトンを出力する。
`-v`	off	全て	計算の途中経過を印字する。
`--si`	off	全て	解の数の計算において、64 ビットまたは 32 ビット型整数を用いる。オーバーフローの検知は行わない。
`--sd`	off	全て	解の数の計算において、double 型を用いる。
`--sb`	off	全て	解の数の計算において、BigInteger 型（任意長整数型）を用いる。
`--sm`	off	全て	解の数の計算において、GMP ライブラリを用いる。
`--sa`	off	全て	解の数の計算において、apfloat ライブラリを用いる（廃止）。