A-009

グラフウォークを列挙する系列二分決定グラフの高速な生成法に関する実験と評価

Experiments and Evaluations on Fast Methods of Generating Sequence BDDs for Graph Walk Enumeration

石丸　亮‡　　青木　洋士‡　　湊　真一‡
Ryo Ishimaru　Hiroshi Aoki　Shin-ichi Minato

1. はじめに

大規模なデータを計算機上で効率よく表現し，かつ高速に処理することは様々な問題を処理するための基盤となる重要な技術である。特に近年，グラフ上の経路の列挙及び索引化に注目が集まっている．列挙及び索引化ができれば，問題の解を具体的に目で見て分析することができ，必要に応じて解の取捨選択や解の数の見積もりを行うことができる．このような特徴から，グラフ上の経路を列挙し索引化する技術は，電力網，交通網，人物関係ネットワーク構造のような現実のグラフ表現で表すことができる実問題への応用が期待できる．ゼロサプレス型二分決定グラフ(Zero suppressed Binary Deci-sion Diagram, ZDD) [1]と呼ばれる組合せ集合の圧縮に応用されているデータ構造がある．グラフ上での頂点の重複を許さない経路(パス)の列挙・索引化は、ZDD を用いたフロンティア法[2]と呼ばれるアルゴリズムによって効率よく表現することができる．

一方で，ZDD では同じ頂点や辺の重複をもつ経路を表現することは困難であるため，グラフ上での一般的な経路であるウォークを列挙・索引化することは難しい．そこで，系列二分決定グラフ(Sequence Binary Decision Diagram, SeqBDD)[3]と呼ばれる系列集合の圧縮に応用されているデータ構造がある．SeqBDD を用いることにより,グラフ上での頂点や辺の重複を許すウォークの列挙・索引化も効率的に行える．しかし，SeqBDD では構築法として，ZDD におけるフロンティア法のような高速な構築法はまだ知られていない．

本研究では，SeqBDD を高速に構築する手法を提案し，実験による比較と考察を行う．

2. 準備

2.1 グラフ

グラフにおける頂点集合をV，辺集合をEとし，それによって表されるグラフを$G=(V,E)$と表現する．頂点$u,v\in V$であるときに$(u,v)\in E$であるならば，頂点uと頂点vが辺(u,v)によってつながっていることを表す．

一般にグラフ上の隣接している辺をたどることによりつくられる辺の系列をウォークとして扱う．

2.2 系列集合

系列集合とは,「n個のアイテムの中から任意の個数だけ重複を許し選んだその並び順」を要素とする集合である．例えば,4 個のアイテムa,b,c,dについて，$\{aaa,dcba\}$, $\{aab,ccd,dd\}$などは系列集合である．系列集合の特徴として，出現順序の区別や文字の重複の区別を可能としており，先に示した ZDD では系列集合を表現できない．

2.3 系列二分決定グラフ

系列二分決定グラフとは系列集合を表現するデータ構造である．SeqBDD は図 1(右図)に示されているように，1 つの根節点と 0 または 1 の終端節点を持つ．それ以外の各節点は 2 値のラベル付けに対応した 0-枝，1-枝と呼ばれる 2 つの出力枝を持つ．SeqBDD のそれぞれの経路は，1 つの系列を表し，その系列は根から終端までのパス上で出現する文字を順に繋げたもののうち，1-枝を終端節点として持つものである．本研究ではグラフの辺を SeqBDD の節点に対応させ，グラフ上をたどることによりできる経路を SeqBDD の系列として表現する(図 1).

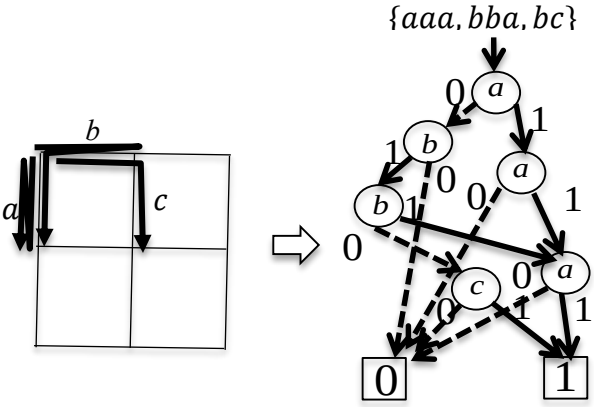

図 1　グラフウォークと系列集合を表す SeqBDD

また，SeqBDD にはいくつかの基本的な演算が行えることが知られている．例えば，2 つの SeqBDD に対する和集合や積集合を返す演算や，ある SeqBDD に対して先頭にアイテムを追加した SeqBDD を新たに返す演算($Push$演算)などが行える．

3. アルゴリズム

3.1 既存手法

グラフウォークにおける SeqBDD の最も愚直な構築方法は，深さ優先探索により該当のウォークに対応する系列をグラフ上から探索し和集合演算を用いて 1 つずつ組み合わせていく方法である．

また，グラフウォークは隣接行列の乗算により求めることが知られているため，隣接行列の非ゼロの要素を SeqBDD とした行列を使う手法も提案されている[4].

‡ 北海道大学大学院情報科学研究科

3.2 提案手法

先に述べた深さ優先を用いた方法では，似たような形の SeqBDD を何度も構築してしまっている．そこを解消するために SeqBDD の$Push$演算を利用した，メモ化を用いた探索アルゴリズムを提案する．基本は深さ優先探索を用いて終端からボトムアップに SeqBDD を構築していく．その過程で各ステップごとに逐一 SeqBDD を記憶していく．そして以前に探索したものと同じ形の SeqBDD が必要になった場合は記憶していた SeqBDD を$Push$演算によって繋げることで再利用(メモ化)することにより無駄な計算を省き高速化をはかる．

4. 評価実験

上記のメモ化によるアルゴリズムを C++で実装した．実行時間，処理の各ステップにおける再帰回数を計測し比較した．

実験は CPU が Intel Core i7 2.93GHz，OS が Ubuntu 14.04，メモリ容量が 8GB という環境下で行った．

4.1 使用したグラフとウォーク

本研究ではグラフの種類として，グリッドグラフと完全グラフの 2 種類を用いた．ウォークは始点を 1 か所に固定し(グリッドグラフは角の頂点，完全グラフはランダムに 1 点)，あるkステップに対しての全ウォークを列挙索引化した．またウォークの種類は，単純なkステップウォーク，辺重複を許さないkステップウォーク，m頂点被覆kステップウォークの 3 種類を扱った．

4.2 アルゴリズム

本研究では先に述べた，愚直な深さ優先による構築方法，メモ化を用いた構築方法に加え，隣接行列を用いた構築方法の 3 つのアルゴリズムで比較を行った．

4.3 各ウォークにおけるメモ化の条件

扱うウォークの種類ごとにメモ化ができる(SeqBDD の形が同じとなる)条件が異なってくるのでそれぞれを以下に示す．

kステップウォーク…今いる頂点，残りステップ数
辺重複を許さないkステップウォーク…今いる頂点，残りステップ数，すでに通った辺の種類
m頂点被覆kステップウォーク…今いる頂点，残りステップ数，すでに通った頂点の種類

5. 実験結果と考察

結果は表 1~3 の通りである．kステップウォークの場合，グリッドグラフにおいて，メモ化によるアルゴリズムが最も計算時間が速い結果となった．完全グラフでも，隣接行列を用いた方法よりはやや遅い結果となったが，愚直な深さ優先と比べるとかなり速くなったと言える．一方で，制約の付いた，辺重複を許さないkステップウォーク・m頂点被覆kステップウォークでは愚直な深さ優先による構築方法よりも遅くなる結果となった．制約のついたkステップウォークでは，メモ化をするための条件が単純なkステップウォークよりも厳しいためあまり効率的に働いていないと考えられる．それに対してメモ化を行うためにプログラム上で時間がかかってしまっているため，結果的に遅くなったと考察する．

表 1　kステップウォーク

グラフ	k	SeqBDD 節点数	計算時間(秒)		
			メモ化無	メモ化有	隣接行列
9×9 グリッドグラフ	16	1224	85.91	0.0045	0.0053
	17	1368	323.67	0.0046	0.0056
	18	1512	1218.77	0.0049	0.0057
15 頂点完全グラフ	7	1259	22.72	0.0042	0.0038
	8	1469	320.79	0.0044	0.0015
	9	1679	TimeOut	0.0047	0.0041

表 2　辺重複を許さない kステップウォーク

グラフ	k	SeqBDD 節点数	計算時間(秒)		
			メモ化無	メモ化有	隣接行列
9×9 グリッドグラフ	18	232890	8.521	52.59	TimeOut
	19	464307	20.499	131.86	TimeOut
	20	909625	49.349	325.08	TimeOut
15 頂点完全グラフ	6	264890	2.273	4.378	11.411
	7	2702531	29.13	55.02	160.19
	8	27020802	366.05	651.39	TimeOut

表 3　m頂点被覆 kステップウォーク

グラフ	m	k	SeqBDD 節点数	計算時間(秒)		
				メモ化無	メモ化有	隣接行列
9×9 グリッドグラフ	4	13	565	7.933	13.501	1.289
	8	13	25489	8.167	5.094	7.198
	12	13	44392	8.099	7.374	10.763
15 頂点完全グラフ	2	6	84	9.291	11.445	0.025
	4	6	21937	9.325	4.037	0.471
	6	6	124577	9.93	0.939	1.602

6. おわりに

今回の実験を通じ，メモ化を用いてグラフウォークを系列とする SeqBDD を列挙索引化することは，単純なkステップウォークに関しては効率的に働くことが分かった．

今後の課題としては，高速化を計ることのできなかったいくつかの制約付きのkステップウォークを高速にするため，アルゴリズムを改良していく必要がある．また，今回の手法はウォークを探索する過程で工夫を入れたものであり，実際に SeqBDD を構築する上での工夫に乏しかったので，その構築手法を考えていくことも必要と思われる．

謝辞

本研究の一部は JSPS 科研費 15H05711 の助成による．

参考文献

[1] Shin-ichi Minato, “Zero-suppressed BDDs and their applications”, STTT, Vol.3, No.2, pp 156-170 (2001).
[2] E. Loekito, J. Bailey, J.Pei “A binary decision diagram based approach for mining frequent subsequences”, Knowledge and Information Systems: An International Journal, Vol.24, No.2, pp 235-268(2010).
[3] 湊 真一, ERATO 湊離散構造処理プロジェクト “超高速グラフアルゴリズム<フカシギの数え方>が拓く,組合せ問題への新アプローチ”, pp 31-45(2015)
[4] 青木 洋士, 安田 宜仁, 湊 真一, “系列二分決定グラフを用いた全頂点ウォークの列挙と索引化”(2015)