LISP プログラミングによるグラフ理論の諸問題の効率的なアルゴリズム

寺林 杏理[1]　東海林 暁貴[1]　小島 正樹[1]

１．木とグラフ

木（tree）とは、計算機科学（computer science）および情報科学（informatics）で頻繁に登場するデータ構造であり[1]、階層的なファイル管理（directory）や、言語処理における構文解析（parse）などに使用される。生物学では進化系統樹に二分木（binary tree）が用いられるし、化学構造式は、分子のトポロジー情報（どの原子どうしがどんな結合で繋がっているか）を表すために、原子を頂点、共有結合を辺とみなすグラフとして歴史的に考案された[2]。またタンパク質のトポロジーが回転可能な単結合を辺とする木で表されること[3]に基づき、二面角を座標とするラグランジュ力学による分子動力学計算プログラム DYANA[4]が開発されている。糖鎖の一次構造も木で表すことができる[5]。

グラフとは、頂点（または節点）と、頂点どうしを接続する辺（または枝）の集合より構成される（厳密な定義は[6]参照）。このとき頂点と辺を辿ってできる軌跡を路（path）といい、始点と終点が一致する路を閉路（cycle）という。数学の一分野としてのグラフ理論は、1736 年に Euler が「ケーニヒスベルクの橋の問題」を、今日のオイラー閉路の考えを用いて解いたことに始まり[2]、その後の位相幾何学へと発展した。

任意の 2 頂点間に路が存在するグラフを連結であるといい、閉路のない連結グラフを木という[6]。木において、ある特別な頂点を根（root）として選ぶと、辺で結ばれた頂点どうしに上下関係（または親子関係）が定まる。このとき最も下流の（すなわち子を持たない）頂点を葉（leaf）という。

２．LISP について

LISP は人工知能（AI）の命名者である McCarthy が 1958 年に考案したプログラミング言語で、リスト（list）というデータ構造を取り扱う LISt Processor に由来する[7]。関数型のプログラミングパラダイムを主にサポートし、AI[8]、記号代数（computer algebra）[9]、論理プログラミング[10]の分野で長い間使用されてきた。MIT（マサチューセッツ工科大学）が、LISP の方言である Scheme を開発し、計算機科学の入門科目で採用して以来、その教科書[11]（略して SICP）を通じてよく知られている。以降では、Scheme[12]を例として説明する。

LISP が扱うリストには 2 つの特徴がある。第 1 に、数値や文字などのデータだけでなく、+演算子などの手続き（procedures）も、リストの要素にすることができる。例えば、(+ 5 7)というリストは、12 と評価される。第 2 に、リスト自体が別のリストの要素であるような入れ子（nested）構造でも構わない。例えば、図 2 の木 B は、((*a c g*) *b d*)というリストで表すことができる。

リスト構造を操作する手続きとしては、car, cdr, cons がある。car は引数に指定したリストの最初の要素を取り出し、cdr は残りのリストを取り出す。例えば（car '(1 2 (3 4))）は 1 と評価され、（cdr '(1 2 (3 4))）は(2 (3 4))と評価される（リストの最初の要素が手続きでなくデータの場合は、開きカ

[1]生命科学部生物情報科学研究室

ッコの前に'を付けてクォートする)。cons は、第二引数のリストの先頭に、第一引数を要素として挿入する。例えば、(cons 1 '(2 3))は(1 2 3)と評価される。なお()は、要素のない空リストを表す。

３．計算機科学におけるグラフ理論の諸問題

3-1. グラフの同型性問題

2 つのグラフの各辺と各頂点が 1 対 1 に対応するとき，両グラフは同型（isomorphic）であるという（正確な定義は[6]参照)。例えば図 1 の 2 つのグラフは同型である。同型なグラフでは、そのトポロジーが同じである。

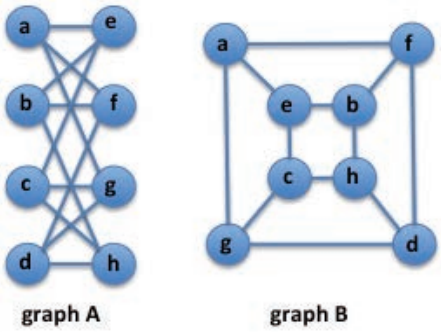

図 1. 同型なグラフ

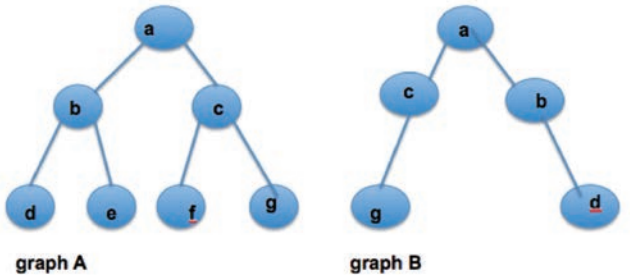

図 2. 部分グラフ同型

3-2. 部分グラフ同型性（subgraph isomorphism）問題

グラフ A がグラフ B と同型なグラフを含むかどうかを判定する問題である（図 2)。与えられた部分構造を含むデータの探索などに応用される。部分グラフ同型性の判定は計算量理論[13]では NP 完全（NP-complete）問題であるが、実用上高速に動作するアルゴリズムも知られている[14, 15]。

3-3. 最大共通部分グラフ（maximum common subgraph）問題

2 つのグラフの最大共通部分を出力する問題（図 3）であり、NP 困難（NP-hard）であることが知られている。実用上はグラフの類似性の評価に用いられ[16]、特別な場合については効率的なアルゴリズムも報告されている[17]。

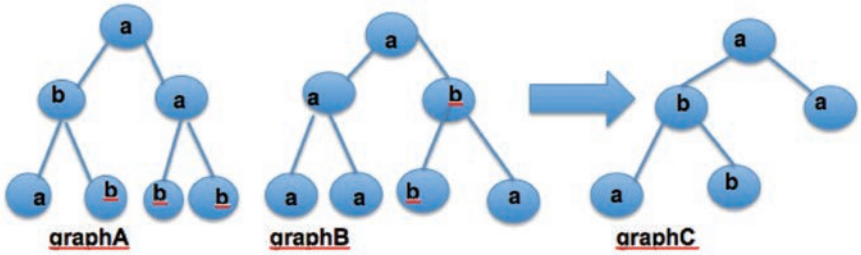

図 3. 最大共通部分グラフ

４．問題の設定

文献[3]に従って、タンパク質分子を、N 末端を根、側鎖を分枝、主鎖を幹とする二分木（C 末端と側鎖の先端が葉）で表し、入れ子のリストで実装する。さらに、単結合まわりの二面角（φ，ψ，ω，χ）を 6 値化した 0 から 5 までの 6 進数（ただし 5+1 は 10 でなく 0）を、リストの要素とする。

こうすることで、
・立体構造（立体配座）の自由度をデータの次元と等しくする
・分子のトポロジー情報をリスト構造自体で表す
ことができる。当研究室ではこの方法に基づくタンパク質解析システム VOLTES（Virtual Optimization of Local TErtiary Structure）を開発し、実際の構造解析に応用している[論文準備中]。

５．実装

木の根（N 末端）がリストの car で、子（根を取り除いた C 末端側）が cdr で表されることに注目して、再帰的（recursive）なアルゴリズム[5]により、上述したグラフ理論の問題を解く。全体のソースコードを図 4 に示す。

まず atom を、対（pair：リストも含まれる）でも空リストでもないものとして定義する。

isomorphic?は、引数の 2 つのリスト *x* と *y* が同型か否かを判定する述語（predicate：真または偽で評価される手続き）である。*x* と *y* がともに空のときは真、ともに atom のときは真とする。*x* と *y* がともに対のときは、(car *x*)と(car *y*)が同型で、かつ(cdr *x*)と(cdr *y*)が同型ならば真とする。それ以外の場合は全て偽と判定する。

nnodes は、木 *x* の節点（node：根や葉も含む）の個数を返す手続きで、(car *x*)の節点の個数と(cdr *x*) の節点の個数の和として再帰的に定義される。もし再帰の途中で、*x* が atom になったら 1 を、空になったら 0 を返す。

```
(define (atom? x)
  (and (not (pair? x)) (not (null? x))))

(define (isomorphic? x y)
  (or (and (null? x) (null? y))
      (and (atom? x) (atom? y))
      (and (pair? x) (pair? y)
           (isomorphic? (car x) (car y)) (isomorphic? (cdr x) (cdr y)))))

; number of nodes
(define (nnodes x)
  (cond ((null? x) 0)
        ((atom? x) 1)
        (else (+ (nnodes (car x)) (nnodes (cdr x))))))

; Common Subgraph
(define (cs x y)
  (if (and (pair? x) (pair? y) (isomorphic? (car x) (car y)))
    (cons (car x) (cs (cdr x) (cdr y)))
    '()))

(define (subsets s)
  (if (null? s) '()
      (cons s (subsets (cdr s)))))

(define (flatnest s)
  (if (null? s) '()
      (append (car s) (flatnest (cdr s)))))

(define (cs-list l m)
  (flatnest
    (map (lambda (x)
      (map (lambda (y) (cs x y)) (subsets m)))
         (subsets l))))

(define (foreach proc s)
  (if (null? s) '()
      (begin (proc (car s))
             (foreach proc (cdr s)))))
```

図 4. ソースコード

手続き cs は、木 *x* と *y* の根を含む共通な同型部分グラフ(cs)を表すリストを出力する。*x* と *y* が対で、かつ(car *x*)と(car *y*)が同型ならば、(cdr *x*)と(cdr *y*)の共通部分グラフの先頭に(car *x*)を挿入したリ

ストを返す。そうでないときは、空リストを返す。例えば、*x* が(1 2 (3 4) 5)、*y* が(5 4 3)のとき、両者の根を含む共通同型部分は(1 2)と(5 4)なので、(cs '(1 2 (3 4) 5) '(5 4 3))は(1 2)と評価される（*x* の部分の方が返される）。

リスト *s* の subsets は、木 *s* の根を順に取り除いていったときの子を要素とするリストを返す。例えば、(subsets '(1 2 3))は((1 2 3) (2 3) (3))と評価される。

手続き flatnest は、引数のリスト s の入れ子構造を平らにする。例えば、(flatnest '(((1 2) (1)) ((2) (3 4))))は、((1 2) (1) (2) (3 4))と評価される。

手続き cs-list は、引数のリスト *l* と *m* の cs を要素とするリストを返す。このとき *l* と *m* の根の位置を順にずらしながら、各根を含む共通同型部分のリストを要素とするリストを作成する。LISP らしく、他のプログラミング言語の 2 重反復ループの代わりに、リストを写像する手続きである map の入れ子で実装する。また手続き cs の 2 つの引数を変数として扱うため、λ 計算を用いる。なお、map の 2 重の入れ子は、2 重の入れ子のリストを生成するので、手続き flatnest で平らにする。

手続き foreach は、第 2 引数のリスト *s* の各要素に、第 1 引数の手続き proc を順に適用するが、map と異なりリスト化はしない。Perl の foreach ループと全く同様の振る舞いをする。

６．実行例

実行時は、あらかじめリスト *l*1 と *l*2 を定義し、*l*1 と *l*2 の各 cs に対して、nnodes（節点の個数）と cs（共通同型部分の *l*1 側）を出力する。*l*1 が(1 2 (3 4) 5)、*l*2 が(5 4 3)のときの実行例を、図 5 に示す。

nnodes の値が *l*1 または *l*2 の節点の個数に等しい場合は、部分同型性が成立していることになる。また nnodes の値が最も大きなものが最大共通部分グラフとなる。

実装したアルゴリズムから明らかな通り、全ての場合の cs を出力するため、最大共通部分グラフが複数存在する場合や、次点（proxime）の共通部分グラフについても求めることができる。

```
(define l1 '(1 2 (3 4) 5))
(define l2 '(5 4 3))
(foreach (lambda (x)
   (newline)
   (display (nnodes x))
   (display ",")
   (display x)
  ) (cs-list l1 l2))

2,(1 2)
2,(1 2)
1,(1)
1,(2)
1,(2)
1,(2)
0,()
0,()
0,()
1,(5)
1,(5)
1,(5)
```

図 5. 実行例

７．ユーザビリティ向上を目指して

もし cs の場合の数が多いときは、nnodes の値でフィルタリングして閾値以上の場合のみ出力するようにするとよい。

また各 cs について、共通同型部分の *l*1 側でなく、そのときの *l*1 と *l*2 の根の位置（リストの先頭から何番目の要素か）を出力しても、必要な情報は得られる。そのときは、cs-list の各要素をさらに入れ子にして *l*1 と *l*2 の根の位置も保存し、出力時は caddr と cadar で呼び出すようにする。写像の入れ子において、*l*1 と *l*2 の根の位置を表す変数は、特殊形式 set!を用いて値を代入すればよい。

８．まとめ

一般にプログラムはアルゴリズム（計算可能な手続き）とデータ構造から成っている[11]が、本稿では解く問題に適したデータ構造を実装し、それに特化したプログラミング言語とアルゴリズムを用いて解決した。今後は作成したプログラムを用いて、タンパク質のトポロジーに基づく構造解析・分子設計を進めていく予定である。

参考文献

1. Knuth 著，有澤・和田監訳，「The Art of Computer Programming 1」，アスキー　（2004）
2. Biggs, Lloyd, Wilson 著，一松・秋山・恵羅訳，「グラフ理論への道（Graph Theory 1736-1936）」，地人書館　（1986）
3. Abe, Braun, Noguti, Go, *Comput. Chem.* **8**, 239-247 (1984)
4. Güntert, Mumenthaler, Wüthrich, *J. Mol. Biol.* **273**, 283-298 (1997)
5. 東海林暁貴　東京薬科大学修士論文 (2014)
6. 浜田隆資・秋山仁著，「グラフ論要説」，槙書店　(1982)
7. Graham 著，野田訳，「On Lisp」，オーム社　（2007）
8. Nrving 著, 杉本訳,「実用 Common Lisp（Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp）」，翔泳社　（2010）
9. 横田博著，「はじめての Maxima」，工学社（2006）; 数式処理システム Maxima は Common Lisp で実装されている
10. Friedmann, Eastlund 著，中野訳，「定理証明手習い（The little Prover）」，ラムダノート（2017）
11. Sussman, Abelson, Sussman 著，和田訳，「計算機プログラムの構造と解釈（Structure and Interpretation of Computer Programs）第 2 版」，ピアソン　(2000)
12. Sperber, Dybvig, Flatt, Straaten (eds.),「Revised^6 Report on the Algorithmic Language Scheme」 (2007)
13. Sipser 著，太田・田中監訳，「計算理論の基礎（Introduction to the Theory of Computation）3」，共立出版　(2008)
14. Ullmann, *J. ACM* **23**, 31-42 (1976)
15. 藤芳, 数理解析研究所講究録 **2040**, 1-9 (2017)
16. Raymond, Gardiner, Willett, *Comput. J.* **45**, 631-644 (2002)
17. Yamaguchi, Aoki, Mamitsuka, *Information Processing Letters* **92**, 57-63 (2004)