特集●プログラミング及びプログラミング言語

最大重み和問題の線形時間アルゴリズムの導出

篠埜 功　胡 振江　武市 正人　小川 瑞史

入力として与えられる再帰データ中の, ある性質を満たす要素集合の中で, 重み和が最大のものを求める, 最大重み和問題の線形時間アルゴリズムの導出法に関する研究が行われているが, 計算量の定数係数が実用にならないほど大きくなるという問題点があった. 本論文においては, 係数の小さい実用的線形時間アルゴリズムの導出法を提案し, その有効性を示すため, 木の最大連結部分和問題, 制限付きナップサック問題の線形時間アルゴリズムの導出を行う.

1　はじめに

本論文では, リスト, 木等の再帰データ上の最大重み和問題の実用的線形時間アルゴリズムの導出法を提案する. 最大重み和問題とは, 「要素に重みの与えられたある再帰データ x が与えられたときに, 性質 p を満たす x 中の要素の部分集合のうちで重みの和が最大のものを求める」という問題で, ナップサック問題等の現実の問題を例として持っており, この問題の効率のよいアルゴリズムを求めることは重要である.

しかし, 最大重み和問題の線形時間アルゴリズムの形式的な導出法を考えるのは簡単ではなく, 具体的な最大重み和問題を解くアルゴリズムを考えることもそれほど易しくはない. 例えば, リスト上の最大重み和問題に最大部分列和問題 (maximum segment sum) [16] という問題がある. この問題の線形時間アルゴリズムを考えるのは一見簡単そうではあるが, Carnegie Mellon 大学の統計学者 Joseph B. Kadane によって発見されるまでは線形時間アルゴリズムは知られていなかった [5].

これまでには, 最大重み和問題の線形時間アルゴリズムの研究は再帰構成グラフ (decomposable graph) という制限されたグラフ上で行われていた [1] [6] [11] [26]. 再帰構成グラフを入力とする最大重み和問題は, 性質 p が正則な述語であれば線形時間で解けることが Bern らによって発見された [6]. さらに, 正則な述語を基本的な述語と $\forall$, $\exists$, $\wedge$, $\vee$, $\neg$ を用いて作り上げられる単項二階論理式によって記述し, これから線形時間アルゴリズムを機械的に導出する方法も Borie らによって考案された [11]. これは理論的には興味深い結果であったが, 素朴に線形時間アルゴリズムを導出するため非常にサイズの大きなテーブルが生成され, 計算量の定数係数が非常に大きくなり全く実用にならないという問題点があった.

本論文では, この問題点を解決し係数の小さい実用的線形時間アルゴリズムを導出する方法を提案する. 具体的には, 例えば最大部分列和問題においては, 上記の Borie らの方法によるアルゴリズムとわれわれの提示する方法によるアルゴリズムを同一の基準で計算量評価すると, 計算量の係数が Borie らの方法では $(4^{(2^{(2^{619})})})^2$ となるのに対し, われわれの提示する

Calculating Linear Time Algorithms for Solving Maximum Weightsum Problems

Isao SASANO, Zhenjiang HU, Masato TAKEICHI, 東京大学大学院工学系研究科情報工学専攻, Department of Information Engineering, University of Tokyo.

Mizuhito OGAWA, NTT コミュニケーション科学基礎研究所・科学技術振興事業団 さきがけ２１, NTT Communication Science Laboratories.

コンピュータソフトウェア, Vol.18, No.5 (2001), pp.1–17.
[論文] 2000 年 5 月 31 日受付.

方法では 8^2 となる.

われわれのアプローチのポイントは, 性質 p を再帰関数で記述するところにある. これにより, 性質 p が相互構造再帰に相当する mutumorphisms という再帰関数群で表されそれらがすべて有限の値域を持つならば, 最大重み和問題を解く線形時間アルゴリズムが機械的に導出できる. 多くの性質は通常論理値を値域とする関数からなる mutumorphisms によって自然に記述可能であり, 生成されるテーブルのサイズは mutumorphisms の値域サイズの積により決まるので, 導出されるアルゴリズムは実用的線形時間アルゴリズムとなる.

本論文では, 最大重み和問題を解くための一般的な最適化定理 (4 章) を与え, その証明を詳しく記述している. また, 最適化定理の有効性を示すため, 木の最大連結部分和問題, ナップサック問題の線形時間アルゴリズムが簡単に導出できることを示す.

本論文の構成は以下のようになっている. まず 2 章で, 再帰的データの表記法等について準備をしたのち, 3 章で, 最大重み和問題を形式的に定義する. 4 章で, 拡張された最適化定理, それを用いた実用的線形時間アルゴリズムの導出戦略を提示する. 5 章, 6 章で, 最適化定理の適用例として, 木に関する最大連結部分和問題, ナップサック問題の線形時間アルゴリズムをそれぞれ導出する. 7 章で関連研究, 8 章で結論を述べる.

2 予備知識

この章では, 再帰的データその他の表記法, また, catamorphism, mutumorphisms, 融合, 組化という基本的な概念 [7] [9] [19] について述べる.

2.1 再帰的データ型

本論文で対象とするデータ型は, 次の形で定義される再帰的データ型とする.

$$\begin{aligned} D\ \alpha &= C_1\ (\alpha, D_1, \ldots, D_{n_1}) \\ &\mid\ C_2\ (\alpha, D_1, \ldots, D_{n_2}) \\ &\mid\ \cdots \\ &\mid\ C_k\ (\alpha, D_1, \ldots, D_{n_k}) \end{aligned}$$

$D\ \alpha$ は定義される型, α は型変数を表す. 重みの与えられる要素は, パラメータ化された α 型の部分であるとする. D_i は $D\,\alpha$ を表す. C_i の引数の $D\,\alpha$ の個数が n_i 個であることを表すためにこのように表記する. C_i はデータ構成子と呼ばれ, α 型の要素と, ある決まった個数の再帰データ $D_1\ \ldots\ D_{n_i}$ から $D\ \alpha$ 型のデータを構成する. この形で定義されるデータ型は polynomial データ型であり, リスト, 2 分木, 根つき木 (rooted tree) [6], 直列並列グラフ (series-parallel graph) [22], 木幅 (tree width) に上限を設けたグラフ [3] などを含む.

最大重み和問題の入力データの型が正則 (regular) データ型の場合, 例えば

$$\mathit{Tree}\ \alpha = \mathit{Node}\ (\alpha, [\mathit{Tree}\ \alpha])$$

によって定義される Rose tree などの場合には, polynomial データ型へ変換する. データ型の変換は 5 章で述べるように容易に行うことができる.

2.2 Catamorphism

再帰データを入力とする関数の中で, 基本的再帰関数のクラスに catamorphism がある. これは, プログラム変換において最も重要な概念の 1 つでもある [9] [19]. 例えば, リスト上の catamorphism は以下のように定義される関数を表す.

$$\begin{aligned} \mathit{cata}\ [\,] &= e \\ \mathit{cata}\ (x : xs) &= x \oplus (\mathit{cata}\ xs) \end{aligned}$$

e, $\oplus$ を変えることにより, さまざまな関数を表すことができる. 関数 cata は, リストを入力として受けとり, その入力リスト中の [ ] を e で, :を $\oplus$ を置き換えて評価したものを結果として返す. 関数 cata は, e, $\oplus$ によって唯一に定まるので, 通常, $\mathit{cata} = (\!|e \triangledown \oplus|\!)$ と記述する.

定義 1 (Catamorphism)
次の等式により定義される関数 f を再帰データ $D\ \alpha$ 上の catamorphism と呼び, $(\!|\phi_1, \ldots, \phi_k|\!)_{D\,\alpha}$ と表す.

$$f\ (C_i\ (e,\ x_1, \ldots, x_{n_i})) = \phi_i\ (e,\ f\ x_1, \ldots, f\ x_{n_i})$$
$$(i = 1, \ldots, k)$$

$(\!|\phi_1, \ldots, \phi_k|\!)_{D\,\alpha}$ における添字の $D\ \alpha$ は, 文脈から明らかな場合には省略することもある. □

Catamorphism は, プログラム変換において重要な役割を果たし, 次の融合定理等に現れる重要な概念である.

定理 1 (融合定理)

$i = 1 \ldots k$ について,

$$f\ (\phi_i\ (e,\ r_1, \ldots, r_{n_i})) = \psi_i\ (e,\ f\ r_1, \ldots, f\ r_{n_i})$$

を満たすとき,

$$f \circ (\![\phi_1, \ldots, \phi_k]\!)_{D\,\alpha} = (\![\psi_1, \ldots, \psi_k]\!)_{D\,\alpha}$$

が成り立つ. □

この融合定理は, ある関数を catamorphism に融合し, 新たな catamorphism を得るための十分条件を与えている. 関数プログラミングにおいては, 小さな関数を組み合わせてプログラムを書くのが一般的であり, これから 1 つの大きな catamorphism を導出するための, 強力な機構を与える.

2.3 Mutumorphisms

Mutumorphisms は, catamorphism を相互再帰的に定義された関数群に一般化したものであり, 次のように定義される [14] [15] [17].

定義 2 (Mutumorphisms)

再帰的データ $D\ \alpha$ 上の mutumorphisms とは, 次のような形で相互再帰的に定義された関数群 $f_1, f_2, \ldots, f_n$ を指す. $i \in \{1, 2, \ldots, n\}$ に対して,

$$\begin{aligned} f_i\ (C_1\ (e, x_1, \ldots, x_{n_1})) &= \\ &\phi_{i1}\ (e,\ h\ x_1, \ldots, h\ x_{n_1}) \\ f_i\ (C_2\ (e, x_1, \ldots, x_{n_2})) &= \\ &\phi_{i2}\ (e,\ h\ x_1, \ldots, h\ x_{n_2}) \\ &\vdots \\ f_i\ (C_k\ (e, x_1, \ldots, x_{n_k})) &= \\ &\phi_{ik}\ (e,\ h\ x_1, \ldots, h\ x_{n_k}) \end{aligned}$$

但し

$$h = f_1 \vartriangle f_2 \vartriangle \ldots \vartriangle f_n. \qquad \square$$

ここで, $f_1 \vartriangle f_2 \vartriangle \ldots \vartriangle f_n$ は次のように定義される関数を表す.

$$(f_1 \vartriangle f_2 \vartriangle \ldots \vartriangle f_n)\ x = (f_1\ x, f_2\ x, \ldots, f_n\ x)$$

Mutumorphisms は組化 (tupling) [14] [15] [17] と呼ばれる変換により, 1 つの catamorphism に変換することができる.

定理 2 (組化定理)

定義 2 における mutumorphisms $f_1, f_2, \ldots, f_n$ について,

$$f_1 \vartriangle f_2 \vartriangle \ldots \vartriangle f_n = (\![\phi_1, \phi_2, \ldots, \phi_k]\!)_{D\,\alpha}$$

$$where\ \phi_i = \phi_{1i} \vartriangle \ldots \vartriangle \phi_{ni} \quad (i = 1, \ldots, k)$$

が成り立つ. □

3 最大重み和問題

この章では, 最大重み和問題を形式的に定義し, これまでの Borie による解法について述べる.

3.1 最大重み和問題の定義

最大重み和問題は次のように定義することができる. 入力としては, 要素に重みの与えられたある再帰データ x が与えられる. その入力データ x 中のいくつかの要素にマークをつけるという操作を考え, これをマーク付けと呼ぶ. 最大重み和問題は, 性質 p を満たすマーク付けの中で, マークの付けられた要素の重みの和が最も大きなものを 1 つ見つける問題と定義できる. 最大重み和問題の解は, すべてのマーク付けの中から性質 p を満たすものを取り出し, そのなかで和が最も大きなものを 1 つ取り出すことによって得られる.

$$mws\ :\ (D\ \alpha^* \to Bool) \to D\ \alpha \to D\ \alpha^*$$

$$mws\ p\ x\ =\ \uparrow wsum\ /\ [x^*\,|\,x^* \leftarrow gen\ x,\ p\ x^*]$$

性質を p, 入力データを x とするとき, $mws\ p\ x$ が 1 つの解を与える. 関数 gen は, 入力データを引数にとり, すべてのマーク付きデータからなるリストを返す. 関数 $wsum$ は重み和を求める関数であり, $\uparrow wsum\ /$ により, 1 つの重み和最大のマーク付きデータが得られる. 演算子 $\uparrow_f$ は次のように定義される.

$$\begin{aligned} a \uparrow_f b &= a, \quad if\ f\ a > f\ b \\ &= b, \quad otherwise \end{aligned}$$

また, 演算子 / は

$$\oplus\ /\ [a_1, a_2, \ldots, a_n] = a_1 \oplus a_2 \oplus \cdots \oplus a_n$$

のように定義される.

関数 gen, $wsum$ を定義する前に, マーク付けに関する記法を定める. あるデータ型 α に対して, α^* は α 型を, 各要素についてマークが付けられているかどうかの情報を持つように拡張したデータ型を表すも

のとする. 具体的には, α^* は

$$\alpha^* = (\alpha, Bool)$$

と定義し, 真偽値により, 要素がマーク付けされているかどうかを表すようにする. $(\alpha, Bool)$ は, α, $Bool$ の直積を表す. また, α^* 型あるいは $D\ \alpha^*$ 型の変数を表すとき, マーク付きの型の変数であることを見た目に分かりやすくするために, $a^*, b^*, \ldots, x^*$ のように変数名に $*$ を付けて表すことがある.

関数 gen は, データ x を引数としてとり, データ x の各要素のマークの付け方を全て列挙する.

$$\begin{aligned}
&gen\ :\ D\ \alpha \to [D\ \alpha^*]\\
&gen = (\!|[\eta_1, \ldots, \eta_k]|\!)\\
&\quad where\\
&\qquad \eta_i\ (e, xs_1^*, \ldots, xs_{n_i}^*) =\\
&\qquad\quad [C_i\ (e^*, x_1^*, \ldots, x_{n_i}^*)\ |\\
&\qquad\qquad e^* \leftarrow [mark\ e, unmark\ e],\\
&\qquad\qquad x_1^* \leftarrow xs_1^*,\\
&\qquad\qquad x_2^* \leftarrow xs_2^*,\\
&\qquad\qquad\ \vdots\\
&\qquad\qquad x_{n_i}^* \leftarrow xs_{n_i}^*]\quad (i = 1 \ldots k)
\end{aligned}$$

ここで, 関数 $mark$, $unmark$ は, マークを付ける関数, マークを付けない関数を表す.

$$\begin{aligned}
mark\ x &= (x, True)\\
unmark\ x &= (x, False)
\end{aligned}$$

関数 $wsum$ は, マーク付きデータを引数にとり, マークの付いた要素の重みの和を返す.

$$\begin{aligned}
&wsum : D\ \alpha^* \to Weight\\
&wsum = (\!|[\phi_1, \ldots, \phi_k]|\!)\\
&\quad where\\
&\qquad \phi_i\ (e^*, w_1, \ldots, w_{n_i}) =\\
&\qquad\quad (if\ marked\ e^*\ then\ weight\ e^*\ else\ 0) +\\
&\qquad\quad w_1 + \cdots + w_{n_i}
\end{aligned}$$

関数 $marked$ は, マーク付き要素 e^* を引数にとり, e^* がマークされているとき $True$, されていないとき $False$ を返す.

$$\begin{aligned}
marked &: D\ \alpha^* \to Bool\\
marked\ (e, m) &= m
\end{aligned}$$

また, $Weight$ は重みを表す型であり, $weight$ はマーク付き要素を引数にとり, その重みを返す関数である.

以上で定義された関数 mws は最大重み和問題の仕様と見ることもでき, これを用いることにより, 最大独立部分列和問題 (4 章), 木の最大連結部分和問題 (5 章), ナップサック問題 (6 章) などのさまざまな具体的最大重み和問題を記述することができる. 例えば, 最大独立部分列和問題は

$$\begin{aligned}
&mis = mws\ p\\
&p\ :\ [\alpha^*] \to Bool\\
&p\ [x] = True\\
&p\ (x : xs) = if\ marked\ x\ then\\
&\qquad\qquad not\ (marked\ (hd\ xs))\ \wedge\ p\ xs\\
&\qquad\quad else\ p\ xs
\end{aligned}$$

のように記述することができ, p は, マークの付いた要素がリスト中に隣接して現れないという性質を表す. この問題の入力はリストであるので $D\ \alpha = [\alpha]$ であり, p の引数の型は, $D\ \alpha^* = [\alpha^*]$ である.

関数 mws の計算量は, このままでは入力データのサイズに関して指数オーダであり, 実用には適さない. 次章で, mws からの実用的線形時間プログラムの導出法を提示する.

3.2 Borie らの解法

最大重み和問題は, グラフアルゴリズムにおいて多くの例を持つ. Bern らは, 多くのグラフに関する問題, 例えば最小頂点被覆問題 (minimum vertex cover problem), 最大独立集合問題 (maximum independent set problem), 最大マッチング問題 (maximum matching problem), 巡回セールスマン問題 (traveling salesman problem) などが最大重み和問題の例であることを示し, さらに, 再帰構成グラフ (decomposable graph) 上では線形時間で解けることを示した[6]. Bern らの研究成果や Courcelle's らの研究成果[13] に基づいて, Borie らは, 最大重み和問題の線形時間アルゴリズムを機械的に導く方法を提案した [11]. Borie らの方法のポイントは, 性質 p を, 頂点 v が枝 e の端点であるという性質を表す述語 $Inc\ (v, e)$ 等の少数の基本的述語を, 論理演算子 $\wedge$, $\vee$,$\neg$ と (1 階または 2 階の) 限量子 $\forall$, $\exists$ を用いて組み合わされる論理式に限定するというところにある. 例えば, 最大独立部分列和問題の

性質 p は,

$$p = \forall v_1\ \forall v_2\ \neg\ Adj\ (v_1, v_2)$$
$$Adj\ (v_1, v_2) = \exists e_1\ (Inc\ (v_1, e_1)\ \wedge\ Inc\ (v_2, e_1)) \wedge\ \neg\ (v_1 = v_2)$$

のように記述される. 性質 p をこのような論理式に制限することにより, $mws\ p$ に対する線形時間アルゴリズムを, 論理式の構造から機械的に導くことができるようになる. これは理論的にはよい成果ではあるが, 導き出される線形時間アルゴリズムは巨大なテーブルを持つので, 実用には適さない [4] [6] [11]. 例えば, 最大独立部分列和問題については, テーブルサイズは $2^{(2^{142})}$ 以上となる. これに対し, われわれの提案する方法ではテーブルサイズは 8 となる (図 3).

4 最適化定理および導出戦略

この章では, 最大重み和問題を解く線形時間アルゴリズムの再帰関数による性質記述からの導出法を提案する. まず導出の際に適用する最適化定理を提示し, それを用いていくつかの最大重み和問題の例の効率のよい線形時間アルゴリズムを導出する.

4.1 最適化定理

ここでは, 最大重み和問題を解く線形時間アルゴリズムを導出する際に用いる最適化定理を, 次の最適化補題をもとにして mutumorphisms, 組化というプログラム変換を利用することにより構築する.

補題 1 (最適化補題)

$$spec\ :\ D\ \alpha \to D\ \alpha^*$$
$$spec\ =\ mws\ (accept \circ (\![\phi_1, \ldots, \phi_k]\!))$$

によって表される最大重み和問題は, 述語 $accept$ の定義域が有限ならば線形時間で解ける.

証明

最大重み和問題が上記の $spec$ のように記述されたとき, この 1 つの解は入力データを x とするとき, 図 1 の関数 opt を用いて

$$opt\ accept\ \phi_1 \ldots \phi_k\ x$$

により与えられる. 図 1 中の関数 $getdata, eachmax$ は, 図 3 中の関数 `getdata`, `eachmax` と同じ関数を表す. また, 述語 $accept$ の定義域の型を $Class$ とすると, $accept$ の型は $Class \to Bool$, $(\![\phi_1, \ldots, \phi_k]\!)$ の型は $D\ \alpha^* \to Class$, 図 1 中の関数 $(\![\psi_1, \ldots, \psi_k]\!)_{D\ \alpha}$ の型は $D\ \alpha \to [(Class, Weight, D\ \alpha^*)]$ である.

以下では, 関数 opt が $spec$ と同じ最適解を返し, 計算量が線形時間であることを示す.

opt の正しさ

まず, 関数 opt が $spec$ と同じ最適解を返すことを, 図 2 で

$$spec\ x = opt\ accept\ \phi_1\ \ldots\ \phi_k\ x$$

を右辺から左辺へ変換することによって示す.

1 つ目の変換は, 関数 opt の展開である.

2 つ目の変換規則は,

$$(\![\psi_1, \ldots, \psi_k]\!) = eachmax \circ (\![\psi_1', \ldots, \psi_k']\!)$$

である. 但し, ψ_i' は次のように定義される関数である.

$$\begin{aligned}&\psi_i'\ (e, cand_1, \ldots, cand_{n_i}) = \\ &\quad [(\phi_i\ (e^*, c_1, \ldots, c_{n_i}),\\ &\qquad (if\ marked\ e^*\ then\ weight\ e^*\ else\ 0)\\ &\qquad + w_1 + \ldots + w_{n_i},\\ &\qquad C_i\ (e^*, r_1^*, \ldots, r_{n_i}^*))\ |\\ &\qquad\qquad e^* \leftarrow [mark\ e, unmark\ e],\\ &\qquad\qquad (c_1, w_1, r_1^*) \leftarrow cand_1,\\ &\qquad\qquad \cdots,\\ &\qquad\qquad (c_{n_i}, w_{n_i}, r_{n_i}^*) \leftarrow cand_{n_i}]\end{aligned}$$

関数 $eachmax$ は $(Class, Weight, D\ \alpha^*)$ 型の 3 つ組のリストを引数にとり, そのリスト中において, 第一要素 (クラス) が同じ 3 つ組が複数ある場合には, 第二要素 (重み和) が最大の 3 つ組の中で最も末尾側にあるもののみを残す. また, 関数 ψ_i' はリスト中の要素の順番を保つ. すなわち, 3 つ組リスト $cand_j$ 中において任意の 2 つの 3 つ組 (c_u, w_u, r_u^*), (c_v, w_v, r_v^*) をとってきたとき, (c_v, w_v, r_v^*) が末尾側にあった場合には, リスト $\psi_i'\ (e, cand_1, \ldots, cand_j, \ldots, cand_{n_i})$ 中において, $C_i\ (e^*, r_1^*, \ldots, r_{j-1}^*, r_v^*, r_{j+1}^*, \ldots, r_{n_i})$ を第三要素とする 3 つ組は $C_i\ (e^*, r_1^*, \ldots, r_{j-1}^*, r_u^*, r_{j+1}^*, \ldots, r_{n_i})$ を第三要素とする 3 つ組より末尾側にある.

したがって,

$$\psi_i\ (e, cand_1, \ldots, cand_{n_i}) = \psi_i\ (e, eachmax\ cand_1, \ldots, eachmax\ cand_{n_i})$$

が成り立つ. 関数 ψ_i, ψ_i' は

$$\psi_i = eachmax \circ \psi_i'$$

$$
\begin{array}{l}
opt\ accept\ \phi_1 \ldots \phi_k\ x = getdata\ (\uparrow_{snd} /\ [\,(c,w,r^*)\,|\,(c,w,r^*) \leftarrow (\!|\psi_1,\ldots,\psi_k|\!)_D\ x,\ accept\ c\,]\,) \\
\quad where\ \psi_i\ (e, cand_1, \ldots, cand_{n_i}) = \\
\quad\quad eachmax\,[\,(\phi_i\ (e^*, c_1, \ldots, c_{n_i}), \\
\quad\quad\quad\quad (if\ marked\ e^*\ then\ weight\ e^*\ else\ 0) + w_1 + \ldots + w_{n_i}, \\
\quad\quad\quad\quad C_i\ (e^*, r_1^*, \ldots, r_{n_i}^*))\ | \\
\quad\quad\quad\quad\quad\quad e^* \leftarrow [mark\ e, unmark\ e], \\
\quad\quad\quad\quad\quad\quad (c_1, w_1, r_1^*) \leftarrow cand_1, \cdots, (c_{n_i}, w_{n_i}, r_{n_i}^*) \leftarrow cand_{n_i}] \qquad (i = 1, \ldots, k)
\end{array}
$$

図 1　最適化関数 *opt*

を満たすので

$$eachmax\ (\psi_i^{'}\ (e, cand_1, \ldots, cand_{n_i})) =$$
$$\psi_i\ (e, eachmax\ cand_1, \ldots, eachmax\ cand_{n_i})$$

となる．よって融合定理 (定理 1) より，

$$(\!|\psi_1, \ldots, \psi_k|\!) = eachmax \circ (\!|\psi_1^{'}, \ldots, \psi_k^{'}|\!)$$

が成り立つ．

3 つ目の変換規則

$$filter\ (accept \circ fst) \circ eachmax =$$
$$eachmax \circ filter\ (accept \circ fst)$$

は, 関数 *filter* と関数 *eachmax* が可換であることを示す．関数 *filter* は

$$
\begin{array}{lcl}
filter & : & (\alpha \to Bool) \to [\alpha] \to [\alpha] \\
filter\ p\ [\,] & = & [\,] \\
filter\ p\ (x : xs) & = & if\ p\ x\ then\ x : filter\ p\ xs \\
 & & else\ filter\ p\ xs
\end{array}
$$

と定義され,

$$\lambda p.\ \lambda xs.\ [\,x\,|\,x \leftarrow xs,\ p\ x\,]$$

と等しい．関数 $filter\ (accept \circ fst) \circ eachmax$ は, 関数 *eachmax* を適用したのちフィルタリングする．関数 $eachmax \circ filter\ (accept \circ fst)$ はフィルタリングしたのち関数 *eachmax* を適用する．述語 $(accept \circ fst)$ はクラス情報のみに関係しており, 関数 *filter* と関数 *eachmax* は, 引数のリストの要素順序を保存するので, これら 2 つの関数が等しいことがいえる．

4 つ目の変換規則は,

$$\uparrow_{snd} / \circ eachmax = \uparrow_{snd} /$$

である．関数 $\uparrow_{snd} /$ がリスト中において最も末尾側の最適解を返し, 関数 *eachmax* が各クラスについて最も末尾側の最適要素を順序を保って返すことから, この等式が成り立つ．

5 つ目の変換規則

$$c = (\!|\phi_1, \ldots, \phi_k|\!)\ r^*$$

は, r^* が属するクラスが c であることを意味している．x に関する帰納法により, リスト $(\!|\psi_1^{'}, \ldots, \psi_k^{'}|\!)\ x$ の各要素 (c, w, r^*) がこの等式を満たすことを示すことができる．

6 つ目の変換規則は

$$getdata \circ \uparrow_{snd} / = \uparrow wsum\ / \circ map\ getdata$$

である．この変換規則は一般には成り立たないが, 引数を xs とした時,

$$map\ snd\ xs = map\ (wsum \circ getdata)\ xs$$

を満たせば, xs に関する帰納法により上記変換規則が成り立つ．実際, リスト

$$xs' = [\,(c, w, r^*)\,|\,(c, w, r^*) \leftarrow (\!|\psi_1^{'}, \ldots, \psi_k^{'}|\!)\ x\,]$$

の各要素 (c, w, r^*) について $w = wsum\ r^*$ が成り立つことが x に関する帰納法で示され, したがって, リスト

$$
\begin{array}{l}
xs = [\,(c, w, r^*)\,|\,(c, w, r^*) \leftarrow (\!|\psi_1^{'}, \ldots, \psi_k^{'}|\!)\ x, \\
\qquad\qquad accept\ ((\!|\phi_1, \ldots, \phi_k|\!)\ r^*)\,]
\end{array}
$$

の各要素 (c, w, r^*) についても $w = wsum\ r^*$ が成り立つ．これから

$$map\ snd\ xs = map\ (wsum \circ getdata)\ xs$$

が導かれる．

7 つ目の変換規則は

$$r^* = getdata\ (c, w, r^*)$$

である．これは, 関数 *getdata* の定義より成り立つ．

8 つ目の変換規則は

$$map\ f \circ filter\ (p \circ f) = filter\ p \circ map\ f$$

である．これは, map-filter 交換則とよばれる [7].

$$
\begin{array}{ll}
 & opt\ accept\ \phi_1 \ldots \phi_k\ x \\
= & \quad \{\ opt \text{ の定義より } \} \\
 & getdata\ (\uparrow_{snd} /\ [\,(c,w,r^*) \\
 & \qquad\qquad |\,(c,w,r^*) \leftarrow (\![\psi_1,\ldots,\psi_k]\!)\ x, \\
 & \qquad\qquad accept\ c]\,) \\
= & \quad \{\ (\![\psi_1,\ldots,\psi_k]\!) = eachmax \circ (\![\psi_1^{'},\ldots,\psi_k^{'}]\!)\ \} \\
 & getdata\ (\uparrow_{snd} / \\
 & \quad [\,(c,w,r^*)\,|\,(c,w,r^*) \leftarrow eachmax\ (\![\psi_1^{'},\ldots,\psi_k^{'}]\!)\ x), \\
 & \qquad accept\ c]\,) \\
= & \quad \{\ filter\ (accept \circ fst) \circ eachmax = \\
 & \qquad eachmax \circ filter\ (accept \circ fst)\ \} \\
 & getdata\ (\uparrow_{snd} /\ (eachmax \\
 & \quad [\,(c,w,r^*)\,|\,(c,w,r^*) \leftarrow (\![\psi_1^{'},\ldots,\psi_k^{'}]\!)\ x, \\
 & \qquad accept\ c]\,)) \\
= & \quad \{\ \uparrow_{snd} / \circ eachmax = \uparrow_{snd} /\ \} \\
 & getdata\ (\uparrow_{snd} / \\
 & \quad [\,(c,w,r^*)\,|\,(c,w,r^*) \leftarrow (\![\psi_1^{'},\ldots,\psi_k^{'}]\!)\ x, \\
 & \qquad accept\ c]\,) \\
= & \quad \{\ c = (\![\phi_1,\ldots,\phi_k]\!)\ r^*\ \} \\
 & getdata\ (\uparrow_{snd} / \\
 & \quad [\,(c,w,r^*)\,|\,(c,w,r^*) \leftarrow (\![\psi_1^{'},\ldots,\psi_k^{'}]\!)\ x, \\
 & \qquad accept\ ((\![\phi_1,\ldots,\phi_k]\!)\ r^*)\,]\,) \\
= & \quad \{\ getdata \circ \uparrow_{snd} / = \uparrow wsum\ / \circ map\ getdata\ \} \\
 & \uparrow wsum\ /\ (map\ getdata \\
 & \quad [\,(c,w,r^*)\,|\,(c,w,r^*) \leftarrow (\![\psi_1^{'},\ldots,\psi_k^{'}]\!)\ x, \\
 & \qquad accept\ ((\![\phi_1,\ldots,\phi_k]\!)\ r^*)\,]\,) \\
= & \quad \{\ r^* = getdata\ (c,w,r^*)\ \} \\
 & \uparrow wsum\ /\ (map\ getdata \\
 & \quad [\,(c,w,r^*)\,|\,(c,w,r^*) \leftarrow (\![\psi_1^{'},\ldots,\psi_k^{'}]\!)\ x, \\
 & \qquad (accept \circ (\![\phi_1,\ldots,\phi_k]\!))\ (getdata\ (c,w,r^*))\,]\,) \\
= & \quad \{\ map\ f \circ filter\ (p \circ f) = filter\ p \circ map\ f\ \} \\
 & \uparrow wsum\ /\ [r^*\,|\,r^* \leftarrow map\ getdata\ (\![\psi_1^{'},\ldots,\psi_k^{'}]\!)\ x), \\
 & \qquad accept\ ((\![\phi_1,\ldots,\phi_k]\!)\ r^*)\,] \\
= & \quad \{\ map\ getdata \circ (\![\psi_1^{'},\ldots,\psi_k^{'}]\!) = (\![\eta_1,\ldots,\eta_k]\!)\ \} \\
 & \uparrow wsum\ /\ [r^*\,|\,r^* \leftarrow (\![\eta_1,\ldots,\eta_k]\!)\ x, \\
 & \qquad accept\ ((\![\phi_1,\ldots,\phi_k]\!)\ r^*)\,] \\
= & \quad \{\ mws \text{ の定義より } \} \\
 & mws\ (accept \circ (\![\phi_1,\ldots,\phi_k]\!))\ x \\
= & \quad \{\ spec \text{ の定義より } \} \\
 & spec\ x
\end{array}
$$

図 2 最適化関数 opt の証明

9 つ目の変換規則

$$map\ getdata \circ (\![\psi_1^{'},\ldots,\psi_k^{'}]\!) = (\![\eta_1,\ldots,\eta_k]\!)$$

は融合定理 (定理 1) より成り立つ. ただし, 関数 $\eta_1,\ldots,\eta_k$ は, 関数 gen の定義中の $\eta_1,\ldots,\eta_k$ であるとする. 融合定理の前提条件は, $i = 1,\ldots,k$ について

$$map\ getdata\ (\psi_i^{'}\ (e, cand_1,\ldots,cand_{n_i})) = \eta_i\ (e,\ map\ getdata\ cand_1,\ldots,\ map\ getdata\ cand_{n_i})$$

が成り立つことであるが, これを以下に示す. まず, $\psi_i^{'}$, $getdata$ の定義より,

$$
\begin{array}{l}
\text{左辺} = [\ C_i\ (e^*, r_1^*,\ldots,r_{n_i}^*)\ | \\
\qquad e^* \leftarrow [mark\ e, unmark\ e], \\
\qquad (c_1, w_1, r_1^*) \leftarrow cand_1, \\
\qquad \cdots, \\
\qquad (c_{n_i}, w_{n_i}, r_{n_i}^*) \leftarrow cand_{n_i}]
\end{array}
$$

となる. 次に η_i の定義より,

$$\begin{aligned}\text{右辺} = [\ C_i\ (e^*, r_1^*, \ldots, r_{n_i}^*)\ \mid \\ e^* \leftarrow [mark\ e, unmark\ e], \\ r_1^* \leftarrow map\ getdata\ cand_1, \\ \cdots, \\ r_{n_i}^* \leftarrow map\ getdata\ cand_{n_i}]\end{aligned}$$

となる. *getdata* の定義より, 左辺 = 右辺 が成り立つ.

10, 11 番目の変換は, 関数 *mws*, *spec* の定義の畳み込みである.

opt の線形性

ここでは, 関数 *opt* の時間計算量が入力データのサイズに関して線形であること, すなわち, 入力データのサイズが n のとき, 関数 *opt* の時間計算量が $O(n)$ であることを示す. ただし, 入力データのサイズは, 入力データ中の要素数と定義する. 関数 *eachmax* はクラス数 (受理関数 *accept* の定義域サイズ) 以下の要素からなるリストを返すので, リスト $(\!|\psi_1, \ldots, \psi_k|\!)_{D\ \alpha}\ x$ の要素数はクラス数以下となる. 関数 *accept*, *getdata* の計算量は $O(1)$ であり, 関数 $\uparrow_{snd}$ / の入力サイズは $O(1)$ であるので, 関数 $(\!|\psi_1, \ldots, \psi_k|\!)_{D\ \alpha}$ の計算量が $O(n)$ であれば, 関数 *opt* の計算量も $O(n)$ となる.

関数 $(\!|\psi_1, \ldots, \psi_k|\!)_{D\ \alpha}$ の計算量が $O(n)$ であることを示すには, $\psi_i\ (e,\ cand_1, \ldots, cand_{n_i})$ が $O(1)$ で計算できることを示せば十分である. ここで, $\psi_i\ (e,\ cand_1, \ldots, cand_{n_i})$ の計算量が $O(1)$ のとき $(\!|\psi_1, \ldots, \psi_k|\!)_{D\ \alpha}\ x$ の計算量が $O(|x|)$ であることを入力データ x に関する帰納法により以下で示す. ただし, $|x|$ はデータ x のサイズを表すものとする. まず, x が再帰部分を含まないデータの場合は明らかである. 次に, x が再帰部分を含む場合を考える. $x = C_i\ (e, x_1, \ldots, x_{n_i})$ とおくと,

$$\begin{aligned}&(\!|\psi_1, \ldots, \psi_k|\!)_{D\ \alpha}\ x = \\ &\quad \psi_i\ (e, (\!|\psi_1, \ldots, \psi_k|\!)_{D\ \alpha}\ x_1, \ldots, \\ &\qquad (\!|\psi_1, \ldots, \psi_k|\!)_{D\ \alpha}\ x_{n_i})\end{aligned}$$

であるが, 帰納法の仮定より

$$(\!|\psi_1, \ldots, \psi_k|\!)_{D\ \alpha}\ x_j\ (1 \le j \le n_i)$$

の計算量は $O(|x_j|)$ である. よって,

$$(\!|\psi_1, \ldots, \psi_k|\!)_{D\ \alpha}\ x$$

の計算量は, $\psi_i\ (e,\ cand_1, \ldots, cand_{n_i})$ の計算量が $O(1)$ であること, および

$$|x| = 1 + \sum_{j=1}^{n_i} |x_j|$$

であることより,

$$O(1) + \sum_{j=1}^{n_i} O(|x_j|) = O(|x|)$$

となる. 以上で, $\psi_i\ (e,\ cand_1, \ldots, cand_{n_i})$ の計算量が $O(1)$ のとき $(\!|\psi_1, \ldots, \psi_k|\!)_{D\ \alpha}\ x$ の計算量が $O(|x|)$, すなわち $O(n)$ であることが示された.

以下で, $\psi_i\ (e,\ cand_1, \ldots, cand_{n_i})$ が $O(1)$ で計算できることを示す. まず, *eachmax* の引数リストの各 3 つ組 (c, w, r^*) が $O(1)$ で求まることが以下のことから分かる. c は ϕ_i の計算量が $O(1)$ であることより $O(1)$ で求まる. r^* は e^*, $r_1^*, \ldots,\ r_{n_i}^*$ を構成子 C_i によって組み合わせることによって得られるので, $O(1)$ で求まる. w は n_i 回の $+$ 演算で求まるが, n_i は定数

$$N = \max\ \{n_i \mid 1 \le i \le k\}$$

以下であるので, $O(1)$ である. よって, *eachmax* の引数リストの各 3 つ組 (c, w, r^*) は $O(1)$ で求まる.

eachmax の引数リストの長さは, 丁度 $(2 \times |cand_1| \times \ldots \times |cand_{n_i}|)$ であり, これは定数 $2C^N$ 以下である. ここで, C はクラス数であり, $|cand|$ はリスト $cand$ の長さを表す. よって, $\psi_i\ (e,\ cand_1, \ldots, cand_{n_i})$ の計算量は $O(1)$ であるといえる. N に関しては指数オーダーであるが, リスト, 2 分木等のよく使われるデータ型に対しては N は小さく, 1 または 2 程度である. □

この最適化補題 1 は, Bern らによって与えられた, 再帰構成グラフ上の最大重み和問題を解く線形時間アルゴリズムが存在するための十分条件 [6] を, 再帰関数によって表現したものである.

補題 2 (分解補題)

性質 $p_0 : D\ \alpha^* \to Bool$ が, 関数 $f_i : D\ \alpha^* \to S_i\ (i = 1, \ldots, n)$ を用いて定義されているとき, 関数群 $p_0, f_1, f_2, \ldots, f_n$ が mutumorphisms をなしており, かつ各集合 S_i の要素数が有限であるならば, 性質 p_0 は, 有限の定義域を持つ述語 *accept* を用いて

$$p_0 = accept \circ (\!|\phi_1, \ldots, \phi_k|\!)$$

と分解することができる.

証明

mutumorphisms p_0, $f_1, \ldots,$ f_n は次のように定義されていたとする.

$$p_0\ (C_1\ (e,\ x_1, \ldots, x_{n_1})) =$$
$$\phi_{01}\ (e,\ h\ x_1, \ldots, h\ x_{n_1})$$
$$p_0\ (C_2\ (e,\ x_1, \ldots, x_{n_2})) =$$
$$\phi_{02}\ (e,\ h\ x_1, \ldots, h\ x_{n_2})$$
$$\vdots$$
$$p_0\ (C_k\ (e,\ x_1, \ldots, x_{n_k})) =$$
$$\phi_{0k}\ (e,\ h\ x_1, \ldots, h\ x_{n_k})$$

また, 各 $i \in \{1, \ldots, n\}$ について,

$$f_i\ (C_1\ (e,\ x_1, \ldots, x_{n_1})) =$$
$$\phi_{i1}\ (e,\ h\ x_1, \ldots, h\ x_{n_1})$$
$$f_i\ (C_2\ (e,\ x_1, \ldots, x_{n_2})) =$$
$$\phi_{i2}\ (e,\ h\ x_1, \ldots, h\ x_{n_2})$$
$$\vdots$$
$$f_i\ (C_k\ (e,\ x_1, \ldots, x_{n_k})) =$$
$$\phi_{ik}\ (e,\ h\ x_1, \ldots, h\ x_{n_k})$$

但し

$$h = p_0 \vartriangle f_1 \vartriangle \cdots \vartriangle f_n.$$

組化定理より,

$$p_0 \vartriangle f_1 \vartriangle \ldots \vartriangle f_n = (\!|\phi_1, \ldots, \phi_k|\!)_{D\,\alpha} \qquad (1)$$

但し

$$\phi_i = \phi_{0i} \vartriangle \phi_{1i} \vartriangle \ldots \vartriangle \phi_{ni} \quad (i = 1, \ldots, k)$$

となる.

$$accept\ (x_0, x_1, \ldots, x_n) = x_0$$

と受理関数を定義することにより,

$$p_0 = accept \circ (\!|\phi_1, \ldots, \phi_k|\!)_{D\,\alpha}$$

が得られる.

次に, 受理関数 *accept* の定義域, すなわち $(\!|\phi_1, \ldots, \phi_k|\!)_{D\,\alpha}$ の値域が有限であることを示す. 式 (1) より, p_0 の値域サイズは 2, 各 $f_i (i \in \{1, \ldots, n\})$ の値域サイズは $|S_i|$ で有限であるので, $(\!|\phi_1, \ldots, \phi_k|\!)_{D\,\alpha}$ の値域サイズ, すなわち *accept* の定義域サイズは $2 \times \Pi_{i=1}^{n} |S_i|$ となり, 有限である. □

以下では, mutumorphisms $f_1, f_2, \ldots, f_n$ について, 各関数 $f_i (i = 1, 2, \ldots, n)$ の値域のサイズが有限のとき, これを有限 mutumorphisms と呼ぶ.

定理 3 (最適化定理)

性質 $p_0 : D\ \alpha^* \rightarrow Bool$ が有限の値域をもつ関数 $f_i : D\ \alpha^* \rightarrow S_i\ (i = 1, \ldots, n)$ と有限 mutumorphisms を形成しているならば,

$$spec \ : \ D\ \alpha \rightarrow D\ \alpha^*$$
$$spec \ = \ mws\ p_0$$

によって表される最大重み和問題は, 線形時間で解ける.

証明

補題 1 および補題 2 より成り立つ. □

以上に提示した最適化定理の特別の場合として, 次の系が成り立つ.

系 1

性質 $p_0 : D\ \alpha^* \rightarrow Bool$ が性質 $p_i : D\ \alpha^* \rightarrow Bool\ (i = 1, \ldots, n)$ と mutumorphisms を形成しているならば,

$$spec \ : \ D\ \alpha \rightarrow D\ \alpha^*$$
$$spec \ = \ mws\ p_0$$

によって表される最大重み和問題は, 線形時間で解ける.

証明

$p_0, p_1, \ldots, p_n$ の値域は *True,False* からなる集合であるので, 関数群 $p_0, p_1, \ldots, p_n$ は有限 mutumorphisms をなす. よって最適化定理 3 より, この系が成り立つ. □

最適化定理は, 最大重み和問題を解く効率のよい線形時間アルゴリズムを導出するときの 1 つのインターフェースを提供する. 性質 p_0 が有限 mutumorphisms で記述されると, 係数の小さな線形時間アルゴリズムが機械的に導出できる. 線形時間アルゴリズムの計算量の係数は, mutumorphisms 中の各値域サイズの積によって決まり, 多くの場合においてこれは小さいので, 実用的線形時間アルゴリズムが得られる.

4.2 導出戦略

ここでは, 最適化定理に基づいて最大重み和問題を解く実用的線形時間アルゴリズムを導出する方法を提示する.

仕様 入力データの構造を表すデータ型 R を定義し, 性質 p を R 上の再帰関数として記述する.

Step 1 データ型 R が, 2.1 節で述べた polynomial データ型ではない場合には, データ型 R に対応する polynomial データ型 D を見つけ, R 上の性質 p を, D 上の性質 p' に変換する. データ

型 R が, polynomial データ型の場合には, 何もしない, すなわち $p' = p$, $D = R$ とする.

Step 2 性質 p' の適当な部分に融合変換を施すことにより, p' を D 上の有限 mutumorphisms を形成する関数群 $p'_0, f'_1, \ldots, f'_n$ に変換する. 融合定理を適用するためには catamorphism が得られていなければならないが, catamorphism を得るために組化が行われることもある.

Step 3 最適化定理を適用し, 線形時間アルゴリズムを得る.

以下で, この導出ステップを説明するため, リストの最大独立部分列和問題の線形時間アルゴリズムを以上の方法に従って導出する.

4.2.1 最大独立部分列和問題

最大独立部分列和問題とは, 重みを持つ要素からなるリストを入力として受け取り, どの 2 つの要素も隣接していないような要素集合のうち, 重み和が最大のものを求めるという問題である. この問題を例として, 線形時間アルゴリズムの導出過程を示す.

仕様 最大独立部分列和問題を解く関数を mis とすると, 関数 mis は,

$$\begin{aligned} mis &: List\ \alpha \to \alpha^* \\ mis &= mws\ p \end{aligned}$$

のように書き表すことができる. 入力データはリストなので, データ型としてはリスト型 $List\ \alpha$ を用いている. ただし, 性質記述を容易にするため, $List\ \alpha$ は空リストを含まないリスト型であるとし, 次のように定義する.

$$List\ \alpha = Single\ \alpha \mid Cons\ (\alpha, List\ \alpha)$$

性質 p は, マークの付けられた要素中には隣接している要素はないという性質であり, これを $List\ \alpha$ 上の再帰関数として記述すればよい.

$$\begin{aligned} &p\ :\ List\ \alpha^* \to Bool \\ &p\ (Single\ x) = True \\ &p\ (Cons\ (x, xs)) = if\ marked\ x\ then \\ &\qquad\qquad not\ (marked\ (hd\ xs)) \\ &\qquad\qquad \wedge\ p\ xs \\ &\qquad else\ p\ xs \\ &hd\ :\ List\ \alpha^* \to \alpha^* \\ &hd\ (Single\ x) = x \\ &hd\ (Cons\ (x, xs)) = x \end{aligned}$$

入力リストが 1 つの要素からなる場合は性質 p を満たす. 入力リストが 2 つ以上の要素からなる場合には, 先頭要素 x にマークが付けられている場合には, 残りのリスト xs が性質 p を満たし, かつ先頭要素 $hd\ xs$ にマークが付けられていないということをチェックする. 先頭要素 x にマークが付けられていない場合には, 残りのリスト xs が性質 p を満たすことをチェックする.

Step 1 $List\ \alpha$ は polynomial データ型 (2.1 節) であるので, データ型の変換はしなくてよい.

Step 2 上記の性質 p は, 関数 hd とともに mutumorphisms をなしているが, hd の値域は α^* 型であるので, α が無限集合の場合には有限 mutumorphisms ではない. 性質 p を $List\ \alpha^*$ 上の有限 mutumorphisms で表すために, $not\ (marked\ (hd\ xs))$ を $p_1\ xs$ とおく.

$$p_1\ xs = not\ (marked\ (hd\ xs))$$

融合変換により,

$$\begin{aligned} p_1 &: List\ \alpha^* \to Bool \\ p_1\ (Single\ x) &= not\ (marked\ x) \\ p_1\ (Cons\ (x, xs)) &= not\ (marked\ x) \end{aligned}$$

が得られ, p_1 の値域は $Bool$ 型で, 有限となる. 性質 p は次のようになり, p, p_1 は $List\ \alpha$ 上の有限 mutumorphisms をなす.

$$\begin{aligned} p\ (Single\ x) &= True \\ p\ (Cons\ (x, xs)) &= if\ marked\ x\ then \\ &\qquad p_1\ xs\ \wedge\ p\ xs \\ &\quad else\ p\ xs \end{aligned}$$

Step 3 最適化定理を適用することにより, 次の

線形時間アルゴリズムを得る.

$$
\begin{aligned}
&mis = opt\ accept\ \phi_1\ \phi_2\\
&\quad where\\
&\qquad accept\ (x,y) = x\\
&\qquad \phi_1\ x = (True, not\ (marked\ x))\\
&\qquad \phi_2\ (x,(y_0,y_1)) = (if\ marked\ x\ then\\
&\qquad\qquad\qquad\qquad y_1\ \wedge\ y_0\\
&\qquad\qquad\qquad else\ y_0,\\
&\qquad\qquad\qquad not\ (marked\ x))
\end{aligned}
$$

関数 *opt* をこの問題に特化して関数型言語 Haskell で記述すると, 図 3 の関数 `mis` のようになる. 例えば, 入力リストが `[1,2,3,4]` のときの最適解は `mis [1,2,3,4]` により与えられ, `[(1,False),(2,True),(3,False),(4,True)]` が結果となる. 上記の関数 *opt* の第一引数 *accept* の引数の型は $(Bool, Bool)$ であるが, 図 3 の関数 `mis` においては, これを`type Class = Int` でコーディングしている. 具体的には, クラス `0` が $(False, False)$ に, クラス `1` が $(False, True)$ に, クラス `2` が $(True, False)$ に, クラス `3` が $(True, True)$ に対応している. また, 図 3 の関数 `foldr1` は演算子 / に対応し, 関数 `bmax second` は演算子 $\uparrow_{snd}$ に対応し, 関数`mis'` は関数 $(\!|\psi_1, \psi_2|\!)_{D\,\alpha}$ に対応する. 関数`table` は ϕ_2 に対応しているが, 第一引数は, マーク付けされているかどうかを判定してから与えられる.

5 木における最大連結部分和問題

この章では, 最適化定理の適用例として, 木の最大連結部分和問題の線形時間アルゴリズムを導出する. この問題は, 最大部分列和問題 [16] の木版である. 最大部分列和問題とは, 入力として数のリストを受け取り, 連続した部分リストの中で最も和の大きなものを求めるという問題である. 木の最大連結部分和問題は, 入力として重み付き要素を頂点とする木を受け取り, 連結している頂点集合の中で最も重み和の大きなものを求めるという問題である. この問題の線形時間アルゴリズムを導出法を用いずに, あるいは系統立てて考えることをしないで見つけるのはそれほど易しくはない.

5.1 仕様

木における最大連結部分和問題の入力は木なので, 次のようなデータ型を定義する.

$$Tree\ \alpha = Node\ (\alpha, [Tree\ \alpha])$$

最大連結部分和問題を解く関数を mcs とすると関数 mcs は

$$
\begin{aligned}
mcs &: Tree\ \alpha \to Tree\ \alpha^*\\
mcs &= mws\ p
\end{aligned}
$$

のように書き表すことができる. 性質 p は, マークの付けられた要素は連結しているという性質であり, これを $Tree\ \alpha$ 上の再帰関数として記述すればよい.

$$
\begin{aligned}
&p : Tree\ \alpha^* \to Bool\\
&p\ (Node\ (v,[\,])) = True\\
&p\ (Node\ (v, t : ts)) =\\
&\qquad if\ nomark\ t\ then\ p\ (Node\ (v,ts))\\
&\qquad else\ p\ t\ \wedge\\
&\qquad\qquad if\ not\ (nomark\ (Node\ (v,ts)))\ then\\
&\qquad\qquad\quad p\ (Node\ (v,ts))\ \wedge\\
&\qquad\qquad\quad marked\ (getRoot\ t)\ \wedge\\
&\qquad\qquad\quad marked\ v\\
&\qquad\qquad else\ True
\end{aligned}
$$

マーク付き木が $Node\ (v,[\,])$ の場合, すなわち 1 つのノード v のみからなる場合には, v がマーク付けられているかどうかにかかわらず連結しているので, $True$ を返す. マーク付き木が $Node\ (v, t : ts)$ である場合には, まず子 t 中にマーク付けられた要素があるかどうかを関数 $nomark$ により判定する. 子 t 中にマーク付けられた要素が 1 つもない場合には, 残りのマーク付き木のマーク付き要素が連結していればよいので, $Node\ (v,ts)$ について性質 p が成り立つかどうかを再帰的に調べる. マーク付き木 t 中にマーク付けられた要素がある場合には, まず, マーク付き木 t について $p\ t$ が成り立つ, すなわちマーク付き要素が連結していることが必要である. また, 残りのマーク付き木 $Node\ (v,ts)$ 中にマークの付けられた要素がある場合には, 連結性を満たすため, マーク付き木 t の根 $getRoot\ t$ およびノード v がマーク付けられていなければならず, また, 残りのマーク付き木 $Node\ (v,ts)$ は性質 p を満たさなければならない.

関数 $nomark$ は, マーク付き木を引数にとり, どの

```
mis :: [Elem] -> [MElem]
mis xs = let opts = mis' xs
         in getdata (foldr1 (bmax second)
                     [ (c,w,r)
                     | (c,w,r) <- opts,
                       accept c])

mis' :: [Elem] -> [(Class,Weight,[MElem])]
mis' [x] = [(2,x,[(x,True)]), (3,0,[(x,False)])]
mis' (x:xs) =
  let opts = mis' xs
  in eachmax [ (table (marked mx) c,
                (if marked mx then weight mx else 0)
                 + w,
                mx:r)
             | mx <- [mark x, unmark x],
               (c,w,r) <- opts]

bmax :: Ord w => (a -> w) -> a -> a -> a
bmax f a b = if f a > f b then a else b

eachmax :: (Eq c, Ord w) => [(c,w,a)] -> [(c,w,a)]
eachmax xs = foldl f [] xs
  where f [] (c,w,r) = [(c,w,r)]
        f ((c,w,r) : opts) (c',w',r') =
            if c==c' then
               if w>w' then (c,w,r) : opts
               else opts ++ [(c',w',r')]
            else (c,w,r) : f opts (c',w',r')

type Weight = Int
type Elem = Weight
type MElem = (Elem,Bool)
type Class = Int

weight :: MElem -> Weight
weight (w,_) = w

marked :: MElem -> Bool
marked (_,m) = m

mark :: Elem -> MElem
mark x = (x,True)

unmark :: Elem -> MElem
unmark x = (x,False)

table :: Bool -> Class -> Class
table True 0 = 0
table True 1 = 0
table True 2 = 0
table True 3 = 2
table False 0 = 1
table False 1 = 1
table False 2 = 3
table False 3 = 3

accept c = c==2 || c==3
second (_,x,_) = x
getdata (_,_,x) = x
```

図 3　最大独立部分列和問題の線形時間アルゴリズム

要素にもマークが付けられていなければ *True*, そうでなければ *False* を返す.

$$nomark : Tree\ \alpha^* \to Bool$$
$$nomark\ (Node\ (v, [\,])) = not\ (marked\ v)$$
$$nomark\ (Node\ (v, t : ts)) =$$
$$\qquad nomark\ t\ \wedge\ nomark\ (Node\ (v, ts))$$

関数 *getRoot* は, マーク付き木を引数にとり, その根を返す.

$$getRoot : Tree\ \alpha^* \to \alpha^*$$
$$getRoot\ (Node\ (v, ts)) = v$$

5.2　導出

Step 1　データ型 *Tree* α は, ノードが任意の個数の子を持つことができるので, polynomial データ型 (2.1 節) にはなっていない. そこで, rooted tree [6] という 2 分木の構造で木を表す.

$$RTree\ \alpha \quad ::= \quad Root\ \alpha$$
$$\qquad\qquad\quad |\quad Join\ (RTree\ \alpha, RTree\ \alpha)$$

この 2 つのデータ型間の変換関数は, 次の 2 つの関数とする.

$$r2t : RTree\ \alpha \to Tree\ \alpha$$
$$r2t\ (Root\ v) = Node\ (v, [\,])$$
$$r2t\ (Join\ (t_1, t_2)) =$$
$$\qquad let\ Node\ (v, ts) = r2t\ t_2$$
$$\qquad in\ Node\ (v,\ (r2t\ t_1) : ts)$$
$$t2r : Tree\ \alpha \to RTree\ \alpha$$
$$t2r\ (Node\ (v, [\,])) = Root\ v$$
$$t2r\ (Node\ (v, t : ts)) =$$
$$\qquad Join\ (t2r\ t, t2r\ (Node\ (v, ts)))$$

これら 2 つの関数は, 線形時間でデータ型を変換する.

次に, *Tree* α 上の性質 p を *RTree* α 上の性質に変換する. p', $getRoot'$ を, p, $getRoot$ に対応する, *RTree* α 上の関数とする. これらの関数は, 次の関係式を満たさなければならない.

$$\begin{aligned} p' &: RTree\ \alpha^* \to Bool \\ p'\ t &= p\ (r2t\ t) \\ getRoot' &: RTree\ \alpha^* \to \alpha^* \\ getRoot'\ t &= getRoot\ (r2t\ t) \end{aligned}$$

単純な変換により,

$$\begin{aligned} &p'\ (Root\ v) = True \\ &p'\ (Join\ (t_1, t_2)) = \\ &\quad if\ nomark'\ t_1\ then\ p'\ t_2 \\ &\quad else\ p'\ t_1\ \wedge \\ &\qquad if\ not\ (nomark'\ t_2)\ then \\ &\qquad\quad p'\ t_2\ \wedge \\ &\qquad\quad marked\ (getRoot'\ t_1)\ \wedge \\ &\qquad\quad marked\ (getRoot'\ t_2) \\ &\qquad else\ True \\ &getRoot'\ (Root\ v) = v \\ &getRoot'\ (Join\ (t_1, t_2)) = getRoot'\ t_2 \\ &nomark'\ (Root\ v) = not\ (marked\ v) \\ &nomark'\ (Join\ (t_1, t_2)) = nomark'\ t_1\ \wedge \\ &\qquad\qquad\qquad\qquad nomark'\ t_2 \end{aligned}$$

が得られる.

Step 2 Step1 でえられた関数群 p',$getRoot'$, $nomark'$ は *RTree* α^* 上の mutumorphisms ではあるが, 関数 $getRoot'$ の値域が α^* 型であることより α の要素数が無限の場合には有限 mutumorphisms ではないので, 最適化定理が適用できない. そこで, $marked \circ getRoot'$ を rm' とおく.

$$\begin{aligned} rm' &: RTree\ \alpha^* \to Bool \\ rm' &= marked \circ getRoot' \end{aligned}$$

関数 *marked* と関数 $getRoot'$ を融合すると,

$$\begin{aligned} rm'\ (Root\ v) &= marked\ v \\ rm'\ (Join\ (t_1, t_2)) &= rm'\ t_2 \end{aligned}$$

が得られる. よって p' は,

$$\begin{aligned} &p'\ (Root\ v) = True \\ &p'\ (Join\ (t_1, t_2)) = \\ &\quad if\ nomark'\ t_1\ then\ p'\ t_2 \\ &\quad else\ p'\ t_1\ \wedge \\ &\qquad if\ not\ (nomark'\ t_2)\ then \\ &\qquad\quad p'\ t_2\ \wedge\ rm'\ t_1\ \wedge\ rm'\ t_2 \\ &rm'\ (Root\ v) = marked\ v \\ &rm'\ (Join\ (t_1, t_2)) = rm'\ t_2 \end{aligned}$$

となり, 有限 mutumorphisms で表される.

Step 3 最適化定理を適用することにより, 次の線形時間アルゴリズムを得る.

$$\begin{aligned} &mcs = r2t \circ (opt\ accept\ \phi_1\ \phi_2) \circ t2r \\ &\quad where \\ &\qquad accept\ (x, y, z) = x \\ &\qquad \phi_1\ v = (True, marked\ v, \\ &\qquad\qquad not\ (marked\ v)) \\ &\qquad \phi_2\ ((x_1, y_1, z_1), (x_2, y_2, z_2)) = \\ &\qquad\qquad (if\ z_1\ then\ x_2 \\ &\qquad\qquad\ else\ x_1\ \wedge \\ &\qquad\qquad\qquad if\ not\ z_2\ then \\ &\qquad\qquad\qquad\quad x_2\ \wedge\ y_1\ \wedge\ y_2 \\ &\qquad\qquad\qquad else\ True, \\ &\qquad\qquad y_2, \\ &\qquad\qquad z_1\ \wedge\ z_2) \end{aligned}$$

accept への入力 (x, y, z) は, 関数 $opt\ accept\ \phi_1\ \phi_2$ への入力が t であった場合には, $(p'\ t, rm'\ t, nomark'\ t)$ である.

以上で線形時間アルゴリズムが導出されたが, ϕ_i をあらかじめ計算してテーブルとして保持しておくこともできる. そのためには以下のように 8 つのクラスを定義する.

$$\begin{aligned} c_0 &= (False, False, False) \\ c_1 &= (False, False, True) \\ c_2 &= (False, True, False) \\ c_3 &= (False, True, True) \\ c_4 &= (True, False, False) \\ c_5 &= (True, False, True) \\ c_6 &= (True, True, False) \\ c_7 &= (True, True, True) \end{aligned}$$

関数 *accept*, ϕ_1 は以下のようになり, 関数 ϕ_2 は

表 1 関数 ϕ_2

ϕ_2	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7
c_0	c_0	c_0	c_2	c_2	c_0	c_0	c_2	c_2
c_1	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7
c_2	c_0	c_0	c_2	c_2	c_0	c_0	c_2	c_2
c_3	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7
c_4	c_0	c_4	c_2	c_6	c_0	c_4	c_2	c_6
c_5	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7
c_6	c_0	c_4	c_2	c_6	c_0	c_4	c_6	c_6
c_7	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7

表 1 のようになる.

$$\begin{aligned} accept\ c &= (c == c_4)\ \vee\ (c == c_5)\ \vee \\ &\quad (c == c_6)\ \vee\ (c == c_7) \\ \phi_1\ v &= if\ marked\ v\ then\ c_6\ else\ c_5 \end{aligned}$$

6 ナップサック問題

この章では, 最適化定理の適用例として重みが正の整数のナップサック問題の線形時間アルゴリズムを導出する. 一般のナップサック問題は NP-完全問題であるが, 重みが正の整数に制限された問題は, 線形時間で解ける. また, 前章までに挙げた最大重み和問題の例においては, 性質 p が述語のみからなる mutumorphisms に変換され, 系 1 の範囲内で線形時間アルゴリズムが得られたが, 本章のナップサック問題においては, 性質 p が述語以外の有限の値域を持つ関数も含む有限 mutumorphisms に変換され, 最適化定理 3 により線形時間アルゴリズムが得られる.

6.1 仕様

ナップサック問題とは, 重み, 価値を持つ荷物の集合が与えられたとき, 重み和が C 以下であるような部分集合のうち価値が最大のものを求めるという問題である. 荷物の集合は, 重みと価値の対を要素とするリスト型 $[(Weight, Value)]$ で表すこととする. ナップサック問題を解く関数を $knap$ とすると, 関数 $knap$ は

$$\begin{aligned} &knap : [(Weight, Value)] \to [(Weight, Value)^*] \\ &knap = mws\ p \end{aligned}$$

のように書き表すことができる. 性質 p は, 選ばれた荷物の重み和が C 以下という性質であり, これをリスト $[(Weight, Value)]$ 上の再帰関数として記述すればよい.

$$\begin{aligned} p &: [(Weight, Value)] \to Bool \\ p\ xs &= weightsum\ xs \le C \end{aligned}$$

ナップサック問題における荷物の価値は, 最大重み和問題に定式化した場合には重みであり, 荷物の重みは, 性質 p に関する値となる.

6.2 導出

Step 1 リストは, polynomial データ型 (2.1 節) であるので, データ型の変換はしなくてよい.

Step 2 性質 p をリスト上の有限 mutumorphisms で表す. そのために, 以下のような関数 cut を導入する.

$$\begin{aligned} cut &: Weight \to Weight \\ cut\ w &= if\ w \le C\ then\ w\ else\ C+1 \end{aligned}$$

この関数 cut は重みを引数にとり, 重みが C より大きければ $C+1$ を返し, そうでなければその重みをそのまま返す. この関数 cut を用いることにより, 以下のように性質 p をリスト上の有限 mutumorphisms で表すことができる.

$$\begin{aligned} &p\ xs = (sumw\ xs) \le C \\ &sumw\ [\,] = 0 \\ &sumw\ (x : xs) = \\ &\qquad cut\ ((if\ marked\ x\ then\ weight\ x \\ &\qquad\qquad else\ 0) + sumw\ xs) \end{aligned}$$

p の取り得る値は $True$,$False$, $sumw$ の取り得る値は $0, 1, \ldots, C+1$ であるので, 関数群 $p, sumw$ はリスト上の有限 mutumorphisms をなす.

Step 3 最適化定理を適用することにより次の線

形時間アルゴリズムを得る.

$$
\begin{aligned}
&knap = opt\ accept\ \phi_1\ \phi_2\\
&\quad where\\
&\qquad accept\ (x,y) = x\\
&\qquad \phi_1 = (True, 0)\\
&\qquad \phi_2\ (x,(y_1,y_2)) =\\
&\qquad\quad (cut\ ((if\ marked\ x\ then\ weight\ x\\
&\qquad\qquad else\ 0) + y_2) \le C,\\
&\qquad\quad cut\ ((if\ marked\ x\ then\ weight\ x\\
&\qquad\qquad else\ 0) + y_2))
\end{aligned}
$$

7 関連研究

1980 年台のなかば頃から, 入力グラフを再帰構成グラフに制限し, グラフを入力とする多くの NP-完全問題の線形時間アルゴリズムに関して研究が行われてきた [2] [10]. その先駆的研究として, Takamizawa らによる直列並列グラフ上での研究 [22] がある.

背景にある概念は木幅 (tree width) [20] というグラフの複雑さの尺度である. 本論文で対象とするグラフは木幅が上界 k 以下のグラフのクラスであるが, これは従来 partial k-tree [1] や k-端点グラフ (k-terminal graph) として提案されてきたものと等価であり, separator [23] や clique [18] などに関連して同様な概念は現れてきた. ハミルトン閉路問題等のグラフに関する多くの NP-完全問題はグラフのサイズに関して線形で木幅に関して指数オーダー以上となるアルゴリズムが構成可能であり [11], アルゴリズムの観点からは木幅がグラフの複雑さを適切に表現しているといえよう. Courcelle は, 木幅に上限のある (超 (hyper)) グラフが閉じた (グラフの) 単項二階論理式 (monadic second order logic) を満たすかどうかは線形時間で判定できることを示した [12]. Borie らは, さらに, 拡張された単項二階論理式によって記述された判定, 列挙, 最適化問題が線形時間で解けることを示した [11]. グラフ版の単項二階論理においては, 通常の単項二階論理における後続関数 (successor function) [24] のかわりに, 頂点 v が枝 e の端点であるかどうかを判定する述語 $Inc\ (v,e)$ が用いられる.

これらの手法により生成されるアルゴリズムは線形ではあるが, 巨大なテーブルを扱うため, 非実用的なほど巨大な係数を持つ. これは, 次の 2 つの理由による.

- テーブルは, 仕様をいくつかの核となる述語に分解した場合の各述語の真偽値のすべての組合せとして構成される. 彼らの方法では, ある固定した基本述語を組み合わせて述語を記述しなければならないため, テーブルサイズは増大する. これに対し, われわれの方法は, mutumorphism により直接核となる述語を記述するため, テーブルサイズは小さくなる.
- 彼らのテーブル構成法は, 限量子 $\forall$,$\exists$ が現れるたびにテーブルサイズが指数関数的爆発を起こして増大する. それに対し, われわれの方法では限量子のかわりに再帰を用いるため, テーブルサイズが小さくなる.

Aspvall らは, 同時に必要とされるテーブルを少なくすることにより, メモリ使用量を押さえる方法を提案した [4]. われわれの方法はテーブルサイズを直接小さくするものであり, 相互に補いあう.

Bird は, リスト上の最大重み和問題の一例である, 最大部分列和問題の線形時間アルゴリズムを導出した [8]. また, Bird と de Moor によって最大部分列和問題等の最適化問題の抽象化された関係の変形に基づく線形時間アルゴリズムの導出法も示されている [9]. 彼らはあるクラスの最適化問題の導出法を非常に一般的な枠組で研究しており, 非常に広い範囲に適用可能である [9]. しかしながら, 彼らの理論を専門家以外が導出に用いるのは難しい. これに対しわれわれは, 最大重み和問題という最適化問題に制限することにより, 非常に単純な線形時間アルゴリズム導出法を得た. 導出における重要な鍵は, 値域サイズが有限の catamorphism, mutumorphism の重要性を認識したところにあり, この点は, [9] においてはほとんど注目されていない. また, de Moor は, ナップサック問題, 木上のナップサック問題等の動的計画法で解ける問題を統一的に扱い, 効率のよいプログラムを導出しているが, 計算量に関する議論はほとんどなく, 線形時間アルゴリズムが導出されるための条件等については明らかにされていない. それに対してわれわれの方法は, 線形時間アルゴリズムが機械的に導出される

条件を明確にしている.

筆者らによる論文 [21] においては, 最大重み和問題を解く実用的線形時間アルゴリズムの導出法およびその際に適用する最適化定理を提案しており, 本論文は, それをさらに発展させたものである. 本論文においては, 最適化定理を, 性質を記述する mutumorphisms に関する制限を緩めることによって拡張し, 拡張した最適化定理の適用例としてナップサック問題の線形時間アルゴリズムを導出した.

8 結論

本論文においては, 再帰データ上の最大重み和問題の実用的線形時間アルゴリズムの導出法およびその際に適用する最適化定理を提案した. 例として, 木の最大連結部分和問題, およびナップサック問題の線形時間アルゴリズムを導出した. われわれの方法は, 再帰関数で性質を記述するというところがポイントであり, それを有限 mutumorphisms へ変換し最適化定理を適用することにより, 多くの場合において実用的線形時間アルゴリズムを得る.

現段階においては, われわれはよく使われるリストや木といった再帰データ型を扱っているが, 木幅がある上限以下のグラフは polynomial データ型 (2.1 節) で表すことができるので [3], 木幅がある上限以下のグラフを対象とすることもできる. 例えば構造的プログラムの制御フローグラフは木幅に上限があるので [25], 線形時間制御フロー解析アルゴリズムの導出もわれわれの方法の対象となり得る.

逆に, われわれの方法の限界は入力データの木幅に上限がなければならないところにある. 例えば 2 次元版の最大部分列和問題は, 2 次元格子構造の木幅には上限がないので, われわれの方法では直接扱うことができない.

参考文献

[1] Arnborg, A. and Proskurowski, A.: Linear Time Algorithms for NP-hard Problems on Graphs Embedded in k-Trees, Technical Report TRITA-NA-8404, Royal Institute of Technology, Sweden, 1984.

[2] Arnborg, S.: Decomposable Structures, Boolean Function, Representations, and Optimization, *Mathematical Foundations of Computer Science 1995*(Wiedermann, J. and Hájek, P.(eds.)), Lecture Notes in Computer Science, Vol. 969, Springer-Verlag, 1995, pp. 21–36.

[3] Arnborg, S., Courcelle, B., Proskurowski, A., and Seese, D.: An Algebraic Theory of Graph Reduction, *Journal of the ACM*, Vol. 40,No. 5(1993), pp. 1134–1164.

[4] Aspvall, B., Proskurowski, A., and Telle, J. A.: Memory Requirements for Table Computations in Partial k-Tree Algorithms, *Algorithmica*, Vol. 27,No. 3(2000), pp. 382–394.

[5] Bentley, J. L.: Programming Pearls: Algorithm Design Techniques, *Communications of the ACM*, Vol. 27,No. 9(1984), pp. 865–871.

[6] Bern, M. W., Lawler, E. L., and Wong, A. L.: Linear-Time Computation of Optimal Subgraphs of Decomposable Graphs, *Journal of Algorithms*, Vol. 8(1987), pp. 216–235.

[7] Bird, R.: An Introduction to the Theory of Lists, *Logic of Programming and Calculi of Discrete Design*(Broy, M.(ed.)), NATO ASI Series, Vol. F36, Springer-Verlag, 1987, pp. 5–42.

[8] Bird, R.: Algebraic Identities for Program Calculation, *The Computer Journal*, Vol. 32, No. 2 (1989), pp. 122–126.

[9] Bird, R. and de Moor, O.: *Algebra of Programming*, Prentice Hall, 1996.

[10] Bodlaender, H. L.: A Tourist Guide through Treewidth, *Acta Cybernetica*, Vol. 11(1993), pp. 1–21.

[11] Borie, R. B., Parker, R. G., and Tovey, C. A.: Automatic Generation of Linear-Time Algorithms from Predicate Calculus Descriptions of Problems on Recursively Constructed Graph Families, *Algorithmica*, Vol. 7(1992), pp. 555–581.

[12] Courcelle, B.: Graph Rewriting: An Algebraic and Logic Approach, *Handbook of Theoretical Computer Science*(van Leeuwen, J.(ed.)), Vol. B, Elsevier Science Publishers, 1990, chapter 5, pp. 194–242.

[13] Courcelle, B.: The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs, *Information and Computation*, Vol. 85,No. 1(1990), pp. 12–75.

[14] Fokkinga, M. M.: Tupling and Mutumorphisms, *Squiggolist*, Vol. 1,No. 4(1989).

[15] Fokkinga, M. M.: *Law and Order in Algorithmics*, PhD Thesis, University of Twente, Dept INF, Enschede, The Netherlands, 1992.

[16] Gries, D.: The Maximum-Segment Sum Problem, *Formal Development of Programs and Proofs*(Dijkstra, E. W.(ed.)), Addison-Wesley, 1990.

[17] Hu, Z., Iwasaki, H., Takeichi, M., and Takano, A.: Tupling Calculation Eliminates Multiple Data Traversals, *Proceedings of the 2nd ACM SIGPLAN International Conference on Functional Programming (ICFP'97)*, Amsterdam, The Netherlands, ACM Press, June 1997, pp. 164–175.

[18] Kříž, I. and Thomas, R.: Clique-Sums, Tree-Decompositions and Compactness, *Discrete Mathematics*, Vol. 81(1990), pp. 177–185.

[19] Meijer, E., Fokkinga, M., and Paterson, R.: Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire, *Proceedings of the 5th International Conference on Functional Programming Languages and Computer Architecture (FPCA'91)*, Lecture Notes in Computer Science, Vol. 523, Cambridge, Massachusetts, ACM Press, August 1991, pp. 124–144.

[20] Robertson, N. and Seymour, P. D.: Graph Minors. II. Algorithmic Aspects of Tree-Width, *Journal of Algorithms*, Vol. 7,No. 3(1986), pp. 309–322.

[21] Sasano, I., Hu, Z., Takeichi, M., and Ogawa, M.: Make it Practical: A Generic Linear-Time Algorithm for Solving Maximum-Weightsum Problems, *Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, Montreal, Canada, ACM Press, September 2000, pp. 137–149.

[22] Takamizawa, K., Nishizeki, T., and Saito, N.: Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs, *Journal of the Association for Computing Machinery*, Vol. 29(1982), pp. 623–641.

[23] Thomas, R.: A Menger-like Property of Tree-Width: The Finite Case, *Journal of Combinatorial Theory, Series B*, Vol. 48(1990), pp. 67–76.

[24] Thomas, W.: Automata on Infinite Objects, *Handbook of Theoretical Computer Science*(van Leeuwen, J.(ed.)), Vol. B, Elsevier Science Publishers, 1990, chapter 4, pp. 133–192.

[25] Thorup, M.: All Structured Programs have Small Tree Width and Good Register Allocation, *Information and Computation*, Vol. 142(1998), pp. 159–181.

[26] Wimer, T. V.: *Linear Algorithms on k-Terminal Graphs*, PhD Thesis, Clemson University, 1987. Report No. URI-030.