Web 文書配置のための制約プログラミング手法

A Constraint Programming Technique for Web Document Layout

細部 博史
Hiroshi HOSOBE

国立情報学研究所
National Institute of Informatics
hosobe@nii.ac.jp

近年，オープンな標準仕様に基づいて情報システムを開発する例が増えてきている．このような背景のもと，Web ブラウザの仕様を適切に記述できるようにするために，制約プログラミングを用いて，Web ブラウザにおける文書の配置を形式化し実現するというアプローチが提案されている．本論文では，このような応用に向けた，Web 文書の配置のための制約プログラミング手法について述べる．

1 はじめに

近年，オープンな標準仕様に基づいて情報システムを開発する例が増えてきている．その代表例は Web ブラウザであり，HTML や Cascading Style Sheets などの標準仕様に基づいている．しかし現実には，Web ブラウザはその実装に依存した振舞いの差が大きく，ある実装で正常に表示できるコンテンツが，他の実装ではうまく表示されないという問題が生じることが多い．このような問題が生じる主な原因として，視覚化処理の複雑さのために，それを仕様として形式化することが困難である点が挙げられる．この問題を解決するために，**制約プログラミング**を用いて，Web ブラウザにおける文書の配置を形式化し実現するというアプローチが提案されている．

このアプローチに基づく従来研究では，制約プログラミングの基盤技術として，**Cassowary** [2] アルゴリズムに基づく制約解消系を採用し，階層的優先度を備えた**線形制約**を扱っているものが多い [1, 5, 10]. Cassowary は，シンプレックス法を拡張することで構築された制約解消法である．

一方，Cassowary と同様に，階層的優先度を備えた線形制約を扱う解消系として，著者はこれまでに，制約の**階層独立性**の解析に基づく制約解消法を提案している [6, 8, 9] ．この方法は，等式制約が多い状況では Cassowary よりも高速であるが，不等式制約が多くなると速度が低下するという欠点を持つ．

本論文では，両者の利点を同時に引き出すことを目的として，階層独立性の解析を組み入れることで Cassowary を改訂した，階層的優先度を備えた線形制約の系の新しい解消法を提案する．本手法では，Cassowary の場合に比べて，制約解消の際に内部的に構成されるタブローが小型化されるため，速度の向上が可能である．本論文では，単純なボックス配置の問題を例題として，本手法の処理を具体的に示す．

2 対象とする問題

本論文で提案する制約解消法が対象とするのは，locally-error-better (LEB) [3] または locally-metric-better [4] と呼ばれる解の基準に基づいて，実数領域上の線形制約からなる**制約階層** [4] の解の 1 つを求める問題である．より具体的には，線形制約として，通常の線形等式および線形不等式，edit (変数の値を繰り返し更新する)，stay (変数の値を一定にする) を扱う．

制約階層を表現するために，各制約に対して，**強さ**またはレベルと呼ばれる階層的な優先度を与える．強さは有限個の段階からなり，常に満たすべき必須制約のレベル required を最上位とし，その下には，例えば strong，medium，weak という，必要に応じて緩和される選好制約のレベルがある．

制約階層の LEB 解集合は，直観的に言えば，同一レベルにある制約を任意の順序で最大限に充足していくことで得られる解と，それらを補間するような中間的な解からなる．LEB 解の基準は weighted-sum-better (WSB) 解の基準を弱めたものであり [4]，LEB は WSB とともに，不等式を含む制約階層を解く場合に適当な基準として知られている [2, 3].

本論文では，制約階層における制約の間に，強さに矛盾しない全順序が与えられていると仮定する．すなわち，強い制約は弱い制約よりも先行し，同じ強さの制約の間に適当な順序が与えられているとする．

3 提案する制約解消法

本節では，前節に述べた問題を解く新しい制約解消法を提案する．

3.1 概要

本論文で提案する制約解消法は，階層独立性 [8] を利用することで，線形制約解消アルゴリズム Cassowary [2] を改訂したものである．シンプレックス法を基礎とする Cassowary と同様に，本手法もタブローを構成して，その更新を繰り返すことで，制約解消を行う．Cassowary との主な相異点は，本手法では，タブローを構成する際に，階層独立性を用いて制約の有効性を判定し，無効な制約を除外する点である．これによって，構成されるタブローの行数が少なくなるため，その後のタブローの更新処理が高速化されることが期待できる．

本手法では，制約の有効性の判定のために，Cassowary に適合する新しい方法を導入する．その基本的な考え方は，[6] で提案した線形制約解消法におけるものと同様であり，以下の通りである．

- 有効な等式制約は，不等式制約を除外した制約階層において階層独立なものである．
- 有効な不等式制約は，それ以外の不等式制約を除外した制約階層において階層独立なものである．

ここで，等式制約 $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = c_i$ または不等式制約 $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq c_i$ が階層独立であるとは，その係数ベクトル $(a_{i1}, a_{i2}, \ldots, a_{in})$ が，制約階層で先行する全ての等式制約 $a_{k1}x_1 + a_{k2}x_2 + \cdots + a_{kn}x_n = c_k$ の係数ベクトル $(a_{k1}, a_{k2}, \ldots, a_{kn})$ に対して線形独立であることをいう．

3.2 新規の制約階層の処理

最初に，新規に与えられた制約階層の解消について述べる．この処理は以下の手順からなる．

1. 有効な制約を用いてタブローを構成する．
2. 目的関数を構成した後，タブローの更新を繰り返して，目的関数を最小化する．

手順 1 では，制約をその全順序に従って 1 つずつ処理する．各ステップでは，階層独立性に基づいて制約の有効性を判定した後，有効な制約に対して以下の操作を行う．

- 等式制約の場合：誤差変数 (制約の緩和を表現するための変数) を補い，1 つの非制限変数 (外部から与えられた変数) を基底変数として左辺に持つ等式に書き換えた後，タブローに追加する．
- 不等式制約の場合：スラック変数 (不等式を等式に変換するための変数) と誤差変数を補い，それらの一方を基底変数として左辺に持つ等式に書き換えた後，タブローに追加する．

ただし，対象となる制約が必須等式制約である場合には，上記の処理における誤差変数の追加は不要である．また，不等式制約の基底変数は，手順 1 の進行中に，スラック変数と誤差変数の間で交換する必要が生じる場合がある．

階層独立性に基づく制約の有効性の判定は，以下のように行う．まず，タブローの情報を用いて，処理の対象となっている制約の非制限変数を可能な限り消去する．その上で，非零の係数を持つ非制限変数が残っていれば，対象となる制約は階層独立であり，従って有効である．

手順 2 は，基本的に Cassowary と同様である．すなわち，目的関数を誤差変数の重み付きの和として構成し，その際，制約階層における階層的な強さを表現するために，重みをベクトルで表す (目的関数の値はベクトル値となり，その比較には辞書式順序が用いられる)．ただし，Cassowary の場合と異なり，無効な制約に対応する誤差変数を目的関数に含める必要はない．その後のタブローの更新処理は，Cassowary と同様のピボット操作を繰り返すことで行う．

3.3 制約の追加

次に，すでに解かれている制約階層に対して，新しい制約を追加する方法を述べる．このとき，追加される制約は，その強さに基づいて，制約の全順序における適当な位置に挿入される．以下では，同じ強さの制約の並びの最後尾に挿入されると仮定する．

追加される制約が等式制約である場合，有効な等式制約をその全順序の逆順に選択し，以下を行う．

- 選択された制約が，追加される制約よりも弱い場合：まず，選択された制約に関する情報がタブローの複数の行に分散されている場合には，ピボット操作で 1 つの行にまとめる．次に，タブローの情報を用いて，追加される制約の非制限変数を可能な限り消去する．その上で，非零の係

数を持つ非制限変数またはスラック変数が残っていれば，選択された制約を無効としてタブローから削除し，追加される制約を有効として，誤差変数を補い，タブローに追加した後，目的関数を再構成して，その最小化を行う．そうでない場合，選択された制約を元の状態に戻し，次の等式制約の処理に進む．

- 選択された制約が，追加される制約と同じ強さであるか，より強い場合：追加された制約を無効とする．

多くの場合，以上の処理を繰り返す必要はなく，最初に選択された制約についてのみ実行すればよい．

追加される制約が不等式制約である場合，Cassowary と同様の方法で処理を行う．

3.4 制約の削除

すでに解かれている制約階層から，既存の制約を削除する方法を述べる．まず，削除される制約が無効である場合には，タブローを修正する必要はない．

削除される制約が有効な等式制約である場合，まず，その制約に関する情報を必要に応じてピボット操作で 1 つの行にまとめた上で削除する．その後，制約の全順序において後方にある無効な制約を正順に選択し，以下を行う．

- 選択された制約が等式である場合：前項で述べた制約の追加と類似の処理を行う．この制約を有効化できない場合，次の制約の処理に進む．
- 選択された制約が不等式である場合：制約を有効化した後，次の制約の処理に進む．

削除される制約が有効な不等式制約である場合，Cassowary と同様の方法で処理を行う．

3.5 変数値の更新

edit 制約による変数値の更新を実現するための処理は，その際の stay 制約の扱いも含めて，Cassowary と同様である．この処理では，各制約の階層独立性が変化しないためである．

4 制約解消の例

本節では，前節で提案した制約解消法の処理を具体的に示す．例題として，Web 文書配置において最も基本的な処理の 1 つであるボックス配置を単純化した問題を用いる．ここでは，図 1 に示されるような包含関係にある 2 つの矩形を考える．簡単のため，矩形の幅のみに注目し，外側と内側の矩形の幅がそれぞれ $w_{\mathrm{out}}, w_{\mathrm{in}}$ によって表されるとする．これらの矩形の間の上下左右のマージン幅は 10 であり，内側の矩形の幅 w_{in} は 50 以上であることが強く望まれ，それぞれの矩形のデフォルトの幅が 0 であるとする．

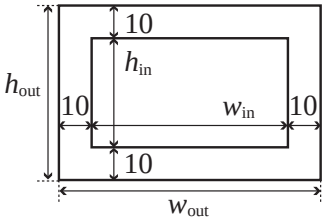

図 1: ボックス配置を単純化した例題

この例題を以下に制約階層として示す．

$$\begin{array}{ll} \mathsf{required} & w_{\mathrm{out}} = w_{\mathrm{in}} + 20 \\ \mathsf{strong} & w_{\mathrm{in}} \geq 50 \\ \mathsf{weak} & w_{\mathrm{out}} = 0 \\ \mathsf{weak} & w_{\mathrm{in}} = 0 \end{array}$$

前節で提案した制約解消法を用いて，この制約階層を解く例を示す．まず，前節で述べた手順 1 により，有効な制約を 1 つずつ追加することで，タブローを構成する．最初に，$\mathsf{required}\ w_{\mathrm{out}} = w_{\mathrm{in}} + 20$ に対応し，w_{out} を基底変数とする等式を加えることで，以下のタブローを得る．

$$w_{\mathrm{out}} = 20 + w_{\mathrm{in}}$$

次に，$\mathsf{strong}\ w_{\mathrm{in}} \geq 50$ にスラック変数 s_2 と誤差変数 δ_2^- を補い，δ_2^- を基底変数として得られた等式を追加して，以下のタブローを得る (横線は，非制限変数を基底変数とする行とそうでないものを分ける)．

$$\begin{array}{r} w_{\mathrm{out}} = 20 + w_{\mathrm{in}} \\ \hline \delta_2^- = 50 - w_{\mathrm{in}} + s_2 \end{array}$$

さらに，$\mathsf{weak}\ w_{\mathrm{out}} = 0$ に誤差変数 δ_3^+, δ_3^- を補った上で，w_{out} を消去し，w_{in} を基底変数とすることで得られた等式をタブローに加えて，以下を得る．

$$\begin{array}{rrr} w_{\mathrm{out}} = & & \delta_3^+ - \delta_3^- \\ w_{\mathrm{in}} = & -20 & + \delta_3^+ - \delta_3^- \\ \hline \delta_2^- = & 70 + s_2 & - \delta_3^+ + \delta_3^- \end{array}$$

最後に，$\mathsf{weak}\ w_{\mathrm{in}} = 0$ が処理されるが，これは階層独立でないために有効化されず，タブローには加えられない．

次に手順 2 に移り，目的関数 z を構成する．

$$\begin{array}{rrl} \text{minimize} & z = & (70,0,0) + (1,0,0)s_2 \\ & & +(-1,0,1)\delta_3^+ + (1,0,1)\delta_3^- \\ \text{subject to} & w_{\text{out}} = & \delta_3^+ - \delta_3^- \\ & w_{\text{in}} = & -20 \quad +\delta_3^+ - \delta_3^- \\ \hline & \delta_2^- = & 70 + s_2 - \delta_3^+ + \delta_3^- \end{array}$$

最後に z の最小化を行う．z を減らすことのできる非基底変数は δ_3^+ のみであり，ピボット操作によりこれを基底変数 δ_2^- と交換することで，以下を得る．

$$\begin{array}{rrl} \text{minimize} & z = & (0,0,70) + (0,0,1)s_2 \\ & & +(1,0,-1)\delta_2^- + (0,0,2)\delta_3^- \\ \text{subject to} & w_{\text{out}} = & 70 + s_2 - \delta_2^- \\ & w_{\text{in}} = & 50 + s_2 - \delta_2^- \\ \hline & \delta_3^+ = & 70 + s_2 - \delta_2^- + \delta_3^- \end{array}$$

この時点で，z は最適となる．これによって得られた解は $w_{\text{out}} = 70$, $w_{\text{in}} = 50$ である．この解は required と strong の制約を完全に充足しながら，weak $w_{\text{out}} = 0$ を最大限に満たすものである．

次の例題として，w_{out} の具体的な値が指定され，以下の制約が新たに追加された状況を考える．

$$\text{medium } w_{\text{out}} = 100$$

直前のタブローにおいて，最も弱い有効な制約は weak $w_{\text{out}} = 0$ であり，この制約に関する情報はタブローの最下位の行にのみ存在している．このとき，medium $w_{\text{out}} = 100$ を有効化できるため，誤差変数を補った上で，最下位の行を置き換える．

$$\begin{array}{rl} w_{\text{out}} = & 70 + s_2 - \delta_2^- \\ w_{\text{in}} = & 50 + s_2 - \delta_2^- \\ \hline \delta^- = & 30 - s_2 + \delta_2^- + \delta^+ \end{array}$$

次に目的関数 z を再構成する．

$$\begin{array}{rrl} \text{minimize} & z = & (0,30,0) + (0,-1,0)s_2 \\ & & +(1,1,0)\delta_2^- + (0,2,0)\delta^+ \\ \text{subject to} & w_{\text{out}} = & 70 + s_2 - \delta_2^- \\ & w_{\text{in}} = & 50 + s_2 - \delta_2^- \\ \hline & \delta^- = & 30 - s_2 + \delta_2^- + \delta^+ \end{array}$$

最後に z の最小化のため，非基底変数 s_2 と基底変数 δ^- をピボット操作により交換する．

$$\begin{array}{rrl} \text{minimize} & z = & (0,1,0)\delta^- \\ & & +(1,0,0)\delta_2^- + (0,1,0)\delta^+ \\ \text{subject to} & w_{\text{out}} = & 100 - \delta^- \quad +\delta^+ \\ & w_{\text{in}} = & 80 - \delta^- \quad +\delta^+ \\ \hline & s_2 = & 30 - \delta^- + \delta_2^- + \delta^+ \end{array}$$

この時点で z は最適であり，解として $w_{\text{out}} = 100$, $w_{\text{in}} = 80$ が得られる．この解は required, strong, medium の制約を完全に満たすものである．

5 おわりに

本論文では，階層的優先度を備えた線形制約の系の解消法として，階層独立性の解析を組み入れることで Cassowary を改訂した手法を提案した．

今後の課題としては，まず，本提案手法を組み込んだ制約解消系の開発と，Cassowary との速度比較に基づく評価実験が挙げられる．その後に，著者が Web 文書の配置を目的として提案した，単方向制約をサポートする制約解消法 DuPlex [7] と，本論文の提案手法を融合することで，表現力と速度の両方を同時に改善する制約解消法の構築を試みる予定である．

参考文献

[1] Badros, G., Borning, A., Marriott, K., and Stuckey, P.: Constraint Cascading Style Sheets for the Web, *Proc. ACM UIST*, 1999, pp. 73–82.

[2] Badros, G. J., Borning, A., and Stuckey, P. J.: The Cassowary Linear Arithmetic Constraint Solving Algorithm, *ACM Trans. Comput.-Human Interact.*, Vol. 8, No. 4 (2001), pp. 267–306.

[3] Borning, A., Anderson, R., and Freeman-Benson, B.: Indigo: A Local Propagation Algorithm for Inequality Constraints, *Proc. ACM UIST*, 1996, pp. 129–136.

[4] Borning, A., Freeman-Benson, B., and Wilson, M.: Constraint Hierarchies, *Lisp Symbolic Comput.*, Vol. 5, No. 3 (1992), pp. 223–270.

[5] Borning, A., Lin, R. K.-H., and Marriott, K.: Constraint-Based Document Layout for the Web, *Multimedia Systems*, Vol. 8, No. 3 (2000), pp. 177–189.

[6] 細部博史: ユーザインタフェースのための線形等式・不等式制約解消系, コンピュータソフトウェア, Vol. 19, No. 6 (2002), pp. 13–20.

[7] Hosobe, H.: Solving Linear and One-Way Constraints for Web Document Layout, *Proc. ACM SAC*, 2005, pp. 1252–1253.

[8] 細部博史, 松岡聡, 米澤明憲: 階層線形系を用いた効率的な制約階層解消法, インタラクティブシステムとソフトウェア V—日本ソフトウェア科学会 WISS'97, レクチャーノート/ソフトウェア学 18, 近代科学社, 1997, pp. 129–134.

[9] 細部博史, 松岡聡, 米澤明憲: HiRise: GUI 構築のためのインクリメンタルな制約解消系, コンピュータソフトウェア, Vol. 16, No. 6 (1999), pp. 33–45.

[10] Michalowski, B.: A Constraint-Based Specification for Box Layout in CSS2, Tech. Rep. 98-06-03, Dept. Comput. Sci. Eng., Univ. Washington, 1998.