Vol.2011-HCI-142 No.21
2011/3/18

ユーザインタフェースのための線形制約解消系

細　部　博　史[†1]

制約は多様な問題解決のための有力な手段であり，プログラミングや人工知能などの分野で利用されている．ユーザインタフェースのプログラミングにおける制約の主要な用途はグラフィカルオブジェクトの配置であり，制約によって配置の処理が自動化されて容易になるという利点がある．本研究では階層的優先度を伴った線形等式制約および線形不等式制約からなる問題を高速に処理するための制約解消系を構築する．

A Linear Constraint Solver for User Interfaces

Hiroshi Hosobe[†1]

Constraints are a powerful tool for solving various problems, and have been used in fields such as programming and artificial intelligence. A primary use of constraints in user interface programming is the layout of graphical objects since constraints automate and ease the layout process. This technical report presents a constraint solver that efficiently handles problems consisting of linear equality and inequality constraints with hierarchical preferences.

1. はじめに

制約は多様な問題解決のための有力な手段であり，プログラミングや人工知能などの分野で利用されている．特にドローイングエディタなどのユーザインタフェースアプリケーションのプログラミングでグラフィカルオブジェクトの配置を制約で表現することは自然であり，これによって配置の処理が自動化されて容易になるという利点がある．このため，ユーザインタフェース分野では古くから制約を用いたプログラミングの研究が行われてきた．

†1 国立情報学研究所
National Institute of Informatics, Japan

制約を用いたユーザインタフェースのプログラミングでは，制約の優先度を処理できるようにすることが重要である．グラフィカルオブジェクトには，それぞれの相対的な位置関係に関する制約に加えて，画面上での存在可能な範囲や，マウスによるオブジェクトの移動などを表す種々の制約を付与する必要があり，プログラマがそれらを矛盾なく指定することが困難なためである．制約の優先度を扱うための理論的枠組みとしては，**制約階層**[3] が広く利用されている．制約階層では個々の制約に**強さ**と呼ばれる数段階の階層的優先度が与えられ，強い制約を可能な限り多く満たすように解が定められる．

本研究では**線形制約**からなる制約階層を高速に処理するための**制約解消系**を構築する．線形制約としては線形等式だけでなく線形不等式も対象とする．本制約解消系は**シンプレックス法**を基礎とする新しい手法を採用している．同様にシンプレックス法を基礎とする制約解消系に Cassowary[1),4)] があるが，本手法は主に以下の 2 点で Cassowary と異なる．

- 問題を**順序制約階層**[16),17)] として扱う．
- 内部処理に **LU 分解**を用いる．

順序制約階層と LU 分解を利用するという点で本手法は HiRise[7),8),10),11)] に類似するが，HiRise ではこれらを等式処理のみに用いていた．本手法は不等式を含む全体の処理に順序制約階層と LU 分解を導入することで効率的な線形制約解消を実現するものであり，HiRise と Cassowary の利点を融合するものであると言える．また本手法は HiRise や Cassowary と同様に**インクリメンタル**な制約解消[6)] を行うことで解の再計算を効率化する．

本稿は以下の構成からなる．まず 2 節で関連研究について紹介し，3 節で制約階層について述べる．次に 4 節で本研究で新たに開発した制約解消法を提案し，5 節でその実装を述べる．最後に 6 節で本研究のまとめと今後の課題を述べる．

2. 関連研究

制約階層の解消に関する初期の研究は，DeltaBlue 制約解消系[6)] に代表されるグラフ理論的なアプローチによって局所伝播制約を扱うものが多かった．ただし，この種の手法は制約が連立している場合に不向きであり，適用可能な問題が制限される．

制約階層における制約の連立を適切に処理するために，HiRise[7),8),10),11)]，Cassowary[1),4)]，QOCA[4),14),15)] などの線形制約解消系が開発された．本研究で提案する制約解消法はこれらの研究の流れに沿ったものであり，前節で述べたように HiRise と Cassowary の利点を融合することで制約解消の効率化を図っている．

局所伝播制約や線形制約よりも複雑な制約を扱う手法も研究されている．ユーザインタ

フェース分野では例えば非線形制約からなる制約階層を近似的に処理する Chorus 制約解消系[9] や，幾何制約を動的に線形近似する Hurst らの手法[12] が開発されている．またコンピュータ支援設計の分野では前述のグラフ理論的なアプローチに類似する手法が盛んに研究されている[13]．一方，本研究で提案する制約解消法はこれらとは異なり，あくまでも線形制約の範囲で制約階層を効率的に処理することを狙っている．

3. 制約階層

本節では本研究の基礎となる制約階層とその特殊形である順序制約階層について述べる．

3.1 定　　義

最初に制約階層[3] について述べる．$[i,j]$ を i から j まで $(i \leq j)$ の整数の区間とする．$\boldsymbol{x}$ を実数領域 $\mathbb{R}$ 上の n 個の変数 $(x_1, x_2, \ldots, x_n)$ のベクトルとする．$\boldsymbol{v}$ (必要に応じてプライム記号を付ける) を n 個の実数 $(v_1, v_2, \ldots, v_n)$ のベクトルとする ($\boldsymbol{x}$ への変数割当を表す)．C を全ての制約の集合とする．強さとは 0 以上，h 以下 (h は規定の正の整数) の整数である．直観的には強さ 0 が最も強く，大きい数ほど弱い強さを表す．制約階層 H とは制約の添字付き多重集合 $\{c_{k,i}\}_{k\in[0,h]\wedge i\in[1,m_k]}$ である．直観的には $c_{k,i}$ は強さ k の制約の中の i 番目の制約を表す．H のレベル k (H_k で表す) は添字付き多重集合 $\{c_{k,i}\}_{i\in[1,m_k]}$ である．H_0 内の制約は必須制約と呼ばれ，他の制約は選好制約と呼ばれる．

誤差関数とは型 $C \times \mathbb{R}^n \to \{0\} \cup \mathbb{R}^+$ の関数である．直観的には $\mathrm{error}(c, \boldsymbol{v})$ は $\boldsymbol{x}$ の値 $\boldsymbol{v} \in \mathbb{R}^n$ によって c が満たされる場合に 0 を返し，それ以外の場合は c が満たされる度合いが低いほど大きい正の実数を返す．制約階層 H に対して，型 $\mathbb{R}^n \times \mathbb{R}^n$ の関係である better 比較子がその解を決めるために定義される．直観的には $\mathrm{better}(\boldsymbol{v}, \boldsymbol{v}')$ は $\boldsymbol{v}$ が H 内の選好制約を $\boldsymbol{v}'$ よりも良く満たすかどうかを判定する．

H の解は全ての必須制約を満たす変数割当の中でより良いものが存在しない変数割当である．形式的には制約階層 H の解集合 $S(H)$ は以下の通り定義される．

$$S(H) = \{\boldsymbol{v} \in S_0(H) \mid \neg\exists \boldsymbol{v}' \in S_0(H), \mathrm{better}(\boldsymbol{v}', \boldsymbol{v})\}$$

ただし $S_0(H)$ は以下の通り定められる．

$$S_0(H) = \{\boldsymbol{v} \in \mathbb{R}^n \mid \forall c_{0,i} \in H, \mathrm{error}(c_{0,i}, \boldsymbol{v}) = 0\}$$

比較子によって異なる種類の解が得られる．重要な比較子に locally-better があり，以下の通り定義される．

$$\begin{aligned}\text{locally-better}(\boldsymbol{v}, \boldsymbol{v}') \equiv \exists k \in [1,h], \forall k' \in [1, k-1],\\ (\forall c_{k',i} \in H, \mathrm{error}(c_{k',i}, \boldsymbol{v}) = \mathrm{error}(c_{k',i}, \boldsymbol{v}')) \wedge\\ (\forall c_{k,i} \in H, \mathrm{error}(c_{k,i}, \boldsymbol{v}) \leq \mathrm{error}(c_{k,i}, \boldsymbol{v}')) \wedge\\ (\exists c_{k,i} \in H, \mathrm{error}(c_{k,i}, \boldsymbol{v}) < \mathrm{error}(c_{k,i}, \boldsymbol{v}'))\end{aligned}$$

locally-better 比較子には locally-error-better (LEB; locally-metric-better とも呼ばれる) と locally-predicate-better の 2 種類があり，これらは誤差関数で異なる．LEB はユーザインタフェースにおける制約処理に適していることが知られ，実際に Indigo[2]，Cassowary[1),4)]，HiRise[7),8),10),11)] などの既存の制約解消系でも用いられている[⋆1]．

3.2 順序制約階層

一般に locally-better は多数の解を許すが，実際には 1 個または少数の locally-better 解を見つければ十分であることが多い．このため，順序制約階層[16),17)] の概念が有用である．直観的には順序制約階層において制約の優先度は各レベル内でも全順序になる．順序制約階層の解は以下の ordered-better 比較子によって定められる．

$$\begin{aligned}\text{ordered-better}(\boldsymbol{v}, \boldsymbol{v}') \equiv \exists k \in [1,h], \forall k' \in [1, k-1],\\ (\forall c_{k',i} \in H, \mathrm{error}(c_{k',i}, \boldsymbol{v}) = \mathrm{error}(c_{k',i}, \boldsymbol{v}')) \wedge\\ (\exists c_{k,i} \in H, (\forall c_{k,i'} \in H, i' < i \Rightarrow \mathrm{error}(c_{k,i'}, \boldsymbol{v}) = \mathrm{error}(c_{k,i'}, \boldsymbol{v}')) \wedge\\ \mathrm{error}(c_{k,i}, \boldsymbol{v}) < \mathrm{error}(c_{k,i}, \boldsymbol{v}'))\end{aligned}$$

ordered-better の利点として，任意の ordered-better 解が locally-better 解になることが知られている[17]．

定理 1　H を任意の制約階層とし，$S_{\mathrm{OB}}(H)$, $S_{\mathrm{LB}}(H)$ をそれぞれ H の ordered-better 解集合，locally-better 解集合とする．このとき，$S_{\mathrm{OB}}(H) \subseteq S_{\mathrm{LB}}(H)$ が成立する．

しかし，この定理の帰結が $S_{\mathrm{OB}}(H) = \emptyset$ の場合も含むことに注意する必要がある．このため，ordered-better を用いる場合には全ての locally-better 解を失うことがないようにする必要がある．

線形制約からなる制約階層はこの点で良い性質を持つ．線形制約 c は，$\boldsymbol{a} \in \mathbb{R}^n, \bowtie \in \{=, \geq\}$, $d \in \mathbb{R}$ としたとき，$\boldsymbol{a}^T\boldsymbol{x} \bowtie d$ と書ける (これを $c : \boldsymbol{a}^T\boldsymbol{x} \bowtie d$ とも書く)[⋆2]．

⋆1 より正確には Cassowary は weighted-sum-better[3] と呼ばれる LEB よりも高感度な比較子 (解集合が小さくなる) を用いている．

⋆2 本研究では厳密な不等号 ($>$) と等号の否定 ($\neq$) を除外する．これらはシンプレックス法による処理が困難であり，加えて制約階層においては意味論上の問題を生じる[3] ためである．

線形制約 $c : \boldsymbol{a}^T\boldsymbol{x} \bowtie d$ の誤差関数は以下の通り定義される．

$$\mathrm{error}(c, \boldsymbol{v}) = \begin{cases} |\boldsymbol{a}^T\boldsymbol{v} - d| & \bowtie \text{が} = \text{である場合} \\ \max(0, d - \boldsymbol{a}^T\boldsymbol{v}) & \bowtie \text{が} \geq \text{である場合} \end{cases}$$

この誤差関数は計量的誤差関数[3] である．この場合に比較子は ordered-error-better (OEB) になり，LEB の特殊形と見なすことができる．

線形制約からなる制約階層の OEB 解に関して以下の命題が導かれる．

命題 2 H を任意の線形制約からなる制約階層とし，$S_{\mathrm{OEB}}(H)$ を H の OEB 解集合とする．このとき，$S_0(H) \neq \emptyset \Rightarrow S_{\mathrm{OEB}}(H) \neq \emptyset$ が成立する．

4. 提 案 手 法

本節では線形制約からなる制約階層を解く新しい手法を提案する．

4.1 対象とする問題

本研究で提案する制約解消法では線形制約として一般の線形等式 $\boldsymbol{a}^T\boldsymbol{x} = d$，線形不等式 $\boldsymbol{a}^T\boldsymbol{x} \geq d$ に加えて，edit 制約，stay 制約を扱う．edit 制約は特定の変数 x_j に対して与えられるもので，内部的には $x_j = d$ という形式であり，d の値が外部から与えられて繰り返し更新される制約である．これは変数に対する入力を表すためのもので，ユーザインタフェースにおいてはマウスによるオブジェクト移動のような場合に用いられる．一方，stay 制約は特定の変数 x_j に対して与えられるもので，内部的には $x_j = d$ という形式であり，d が x_j の制約解消直前の値に設定される制約である．stay 制約は通常，選好制約として与えられ，対応する変数の値を可能な限り維持するために用いられる．

本手法ではこのような線形制約からなる制約階層を OEB によって順序制約階層として解く．強さ 0 から $h-1$ までの制約はユーザによって外部から与えられる．一方，各変数 x_j に関するデフォルトの制約として，強さ h の x_j に関する stay 制約が常に内部的に生成される．これによって制約の集合が修正されても解が大きく変化することがないようにする．

4.2 問題の内部表現

本手法における個々の制約の内部表現は基本的に Cassowary と同様である．具体的には各制約を以下の通り表現する．

$$\boldsymbol{a}^T\boldsymbol{x} = \begin{cases} d + y_{\mathrm{a}}^+ - y_{\mathrm{a}}^- & \text{必須等式の場合} \\ d + y_{\mathrm{s}} - y_{\mathrm{a}}^- & \text{必須不等式の場合} \\ d + y_{\mathrm{e}}^+ - y_{\mathrm{e}}^- & \text{選好等式の場合} \\ d + y_{\mathrm{s}} - y_{\mathrm{e}}^- & \text{選好不等式の場合} \end{cases}$$

ただし，$y_{\mathrm{a}}^+, y_{\mathrm{a}}^-$ は人工変数[⋆3], $y_{\mathrm{e}}^+, y_{\mathrm{e}}^-$ は誤差変数, y_{s} はスラック変数と呼ばれ，いずれも非負実数値を取る変数である．これらの変数は制限変数と呼ばれ，通常の変数 (x_j) は非制限変数と呼ばれる．本手法の実行によって人工変数は 0 になり (必須制約に矛盾が含まれる場合はこれが達成されずエラーが報告される)，誤差変数は制約階層の意味で可能な限り 0 に近付けられ，スラック変数には適当な非負実数値が与えられる．

本手法では制約階層 H から，階層線形計画問題と呼ぶ以下の最適化問題を内部的に構成する．

$$\text{minimize } \boldsymbol{z} = W\boldsymbol{y}$$
$$\text{subject to } A\boldsymbol{x} + B\boldsymbol{y} = \boldsymbol{d},\ \boldsymbol{y} \geq \boldsymbol{0}$$

$\boldsymbol{x}, \boldsymbol{y}$ はそれぞれ非制限変数，制限変数のベクトルであり，$A\boldsymbol{x} + B\boldsymbol{y} = \boldsymbol{d}$ は H に含まれる制約を任意の順序で縦に並べたものである．

目的関数 $\boldsymbol{z} = W\boldsymbol{y}$ は $\sum_{k\in[0,h]} m_k$ 個の式 $z_i = \boldsymbol{w}_i^T\boldsymbol{y}$ を縦に並べたものである．各 $z_i = \boldsymbol{w}_i^T\boldsymbol{y}$ は，$i = \sum_{k\in[0,k'-1]} m_k + i'$ かつ $1 \leq i' \leq m_{k'}$ であるような制約 $c_{k',i'}$ に対応する．$\boldsymbol{w}_i$ においては $c_{k',i'}$ の人工変数と誤差変数に対応する要素のみを 1 とし，他の要素を 0 とする．目的関数 $\boldsymbol{z}$ の最小化は辞書式順序に基づき，上位の z_i の値が小さくなるように行われる．なお，Cassowary においても目的関数がベクトルとなるが，Cassowary ではレベルごとに人工変数と誤差変数の和を作っているのに対して，本手法では制約ごとに作ることが問題の内部表現における本質的な違いとなっている．

このように構成した階層線形計画問題を解くことで，元の制約階層の解を得ることができる．これは以下の命題により保証される．

命題 3 線形制約からなる制約階層 H の OEB 解集合は，H から構成される階層線形計画問題の解集合と等しい．

4.3 LU 分解形式

本手法は階層線形計画問題を LU 分解に基づくシンプレックス法で解く．通常のシンプ

⋆3 後述する通り人工変数は 0 になるべき変数であり，実装上は必要な場合に限り導入すればよい．

レックス法と同様，変数は基底変数と非基底変数に分けられる．非制限変数，制限変数の基底変数をそれぞれ $\boldsymbol{x}^*$, $\boldsymbol{y}^*$ とし，非制限変数，制限変数の非基底変数をそれぞれ $\boldsymbol{x}^\circ$, $\boldsymbol{y}^\circ$ とする．これらを用いて階層線形計画問題を以下のように書くことができる．

$$\begin{aligned}&\text{minimize } \boldsymbol{z} = W^*\boldsymbol{y}^* + W^\circ\boldsymbol{y}^\circ\\ &\text{subject to } A^*\boldsymbol{x}^* + A^\circ\boldsymbol{x}^\circ + B^*\boldsymbol{y}^* + B^\circ\boldsymbol{y}^\circ = \boldsymbol{d},\ \boldsymbol{y}^* \geq \boldsymbol{0},\ \boldsymbol{y}^\circ \geq \boldsymbol{0}\end{aligned}$$

さらに最初の制約条件を以下のように書き換える．

$$[A^*\ B^*]\begin{bmatrix}\boldsymbol{x}^*\\ \boldsymbol{y}^*\end{bmatrix} = \boldsymbol{d} - [A^\circ\ B^\circ]\begin{bmatrix}\boldsymbol{x}^\circ\\ \boldsymbol{y}^\circ\end{bmatrix}$$

ここで下 3 角行列 L，上 3 角行列 U による $[A^*\ B^*]$ の LU 分解 $[A^*\ B^*] = LU$ を導入し，上式の両辺に左から L^{-1} を掛けることで，本手法における LU 分解形式として以下を得る．

$$U\begin{bmatrix}\boldsymbol{x}^*\\ \boldsymbol{y}^*\end{bmatrix} = \begin{bmatrix}\boldsymbol{d}'_1\\ \boldsymbol{d}'_2\end{bmatrix} + \begin{bmatrix}A'_1 & B'_1\\ A'_2 & B'_2\end{bmatrix}\begin{bmatrix}\boldsymbol{x}^\circ\\ \boldsymbol{y}^\circ\end{bmatrix}$$

ただし，$[(\boldsymbol{d}'_1)^T\ (\boldsymbol{d}'_2)^T]^T = L^{-1}\boldsymbol{d}$, $[(A'_1)^T\ (A'_2)^T]^T = -L^{-1}A^\circ$, $[(B'_1)^T\ (B'_2)^T]^T = -L^{-1}B^\circ$ である．

LU 分解形式は基本的なシンプレックス法で用いられるタブロー形式と密接な関係を持つ．実際，LU 分解形式の両辺に左から U^{-1} を掛けたものと目的関数を用いることで，以下のタブロー形式が得られる．

$$\begin{aligned}\boldsymbol{z} &= W^*\boldsymbol{d}''_2 + W^*A''_2\boldsymbol{x}^\circ + (W^*B''_2 + W^\circ)\boldsymbol{y}^\circ\\ \begin{bmatrix}\boldsymbol{x}^*\\ \boldsymbol{y}^*\end{bmatrix} &= \begin{bmatrix}\boldsymbol{d}''_1\\ \boldsymbol{d}''_2\end{bmatrix} + \begin{bmatrix}A''_1 & B''_1\\ A''_2 & B''_2\end{bmatrix}\begin{bmatrix}\boldsymbol{x}^\circ\\ \boldsymbol{y}^\circ\end{bmatrix}\end{aligned}$$

通常のタブロー形式と同様，暫定解は $\boldsymbol{x}^* = \boldsymbol{d}''_1$, $\boldsymbol{y}^* = \boldsymbol{d}''_2$, $\boldsymbol{x}^\circ = \boldsymbol{0}$, $\boldsymbol{y}^\circ = \boldsymbol{0}$ である．

4.4 ピボット操作

本手法の処理は通常のシンプレックス法と同様，ピボット操作による LU 分解形式の更新によって進められる．ピボット操作にあたり，最初にピボットとなる非基底変数と基底変数の組を決定する必要がある．本手法が通常のシンプレックス法と異なるのは主に，目的関数がベクトルになっていることと，非制限変数が導入されていることの 2 点である．ただし，これらの 2 点は Cassowary でも行われており (前述の通り目的関数の構成方法は異なる)，ピボットの決定方法に関する基本的な考え方は同様である．

ピボットの決定後に LU 分解形式を更新する必要がある．数理計画法の分野でいくつかの更新方法が知られているが，本手法では基本的な Forrest-Tomlin 法[5] を用いている．

4.5 インクリメンタル処理

本手法では以下の 3 種類の操作に応じてインクリメンタルな制約解消を行う．

- 新規の制約を制約階層に追加する．
- 既存の制約を制約階層から削除する．
- 既存の edit 制約を通じて対応する変数の値を更新する．

制約のインクリメンタルな追加処理は比較的容易であり，edit 制約による変数値のインクリメンタルな更新処理は Cassowary と同様に双対シンプレックス法で対応できる．一方，制約の削除に関しては，LU 分解から行を削除する新しい手法を導入することでインクリメンタル処理を実現している．

5. 実　　装

本研究で提案した制約解消法に基づいて制約解消系を現在開発している．実装に用いている言語は Scala であり，Java プログラムからも本制約解消系を使用できる．アプリケーションプログラミングインタフェースの設計は HiRise や Cassowary との互換性に配慮したものである．具体的にはプログラマは変数と制約をオブジェクトとして生成し，制約オブジェクトの追加・削除操作を制約解消系オブジェクトに対して行うことで処理を記述する．

6. おわりに

本稿ではユーザインタフェースを対象とした新しい制約解消法を提案した．本手法は線形制約からなる制約階層を高速に処理できる．本稿ではまた制約解消系としての本手法の実装について述べた．

制約解消系を完成させることが現在の課題である．その後の課題としては，HiRise，Cassowary との比較に基づく評価実験を行うことや，種々のユーザインタフェースアプリケーションを作成して本手法の有効性をより実証的に確認することが挙げられる．

参 考 文 献

1) 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, pp.267–306 (2001).
2) Borning, A., Anderson, R. and Freeman-Benson, B.: Indigo: A Local Propagation Algorithm for Inequality Constraints, *Proc.ACM UIST*, pp.129–136 (1996).
3) Borning, A., Freeman-Benson, B. and Wilson, M.: Constraint Hierarchies, *Lisp*

Symbolic Comput., Vol.5, No.3, pp.223–270 (1992).
4) Borning, A., Marriott, K., Stuckey, P. and Xiao, Y.: Solving Linear Arithmetic Constraints for User Interface Applications, *Proc.ACM UIST*, pp.87–96 (1997).
5) Forrest, J. J. H. and Tomlin, J. A.: Updated Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method, *Math. Prog.*, Vol. 2, pp. 263–278 (1972).
6) Freeman-Benson, B.N., Maloney, J. and Borning, A.: An Incremental Constraint Solver, *Comm.ACM*, Vol.33, No.1, pp.54–63 (1990).
7) 細部博史：GUI を対象とした線形計算による制約階層解消系の高速化，コンピュータソフトウェア， Vol.17, No.2, pp.25–29 (2000).
8) Hosobe, H.: A Scalable Linear Constraint Solver for User Interface Construction, *Proc.CP*, LNCS, Vol.1894, pp.218–232 (2000).
9) Hosobe, H.: A Modular Geometric Constraint Solver for User Interface Applications, *Proc.ACM UIST*, pp.91–100 (2001).
10) 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, pp.13–20 (2002).
11) 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, pp.33–45 (1999).
12) Hurst, N., Marriott, K. and Moulder, P.: Dynamic Approximation of Complex Graphical Constraints by Linear Constraints, *Proc.ACM UIST*, pp.191–200 (2002).
13) Jermann, C., Trombettoni, G., Neveu, B. and Mathis, P.: Decomposition of Geometric Constraint Systems: a Survey, *Intl.J.Comput.Geometry Appl.*, Vol.16, No.5–6, pp.379–414 (2006).
14) Marriott, K. and Chok, S.S.: QOCA: A Constraint Solving Toolkit for Interactive Graphical Applications, *Constraints*, Vol.7, No.3–4, pp.229–254 (2002).
15) Marriott, K., Chok, S. S. and Finlay, A.: A Tableau Based Constraint Solving Toolkit for Interactive Graphical Applications, *Proc.CP*, LNCS, Vol.1520, pp.340–354 (1998).
16) Rudová, H.: Constraints with Variables' Annotations and Constraint Hierarchies, *Proc.SOFSEM*, LNCS, Vol.1521, pp.409–418 (1998).
17) Wolf, A.: Transforming Ordered Constraint Hierarchies into Ordinary Constraint Systems, *Over-Constrained Systems*, LNCS, Vol.1106, pp.171–187 (1996).