HiRise: GUI構築のためのインクリメンタルな制約解消系

細部 博史　松岡 聡　米澤 明憲

制約を用いたグラフィカルユーザーインターフェース(GUI)の研究における近年の最重要テーマの1つは，制約の優先度の実現である．特に，強さと呼ばれる優先度を導入した制約階層をGUIで利用できるようにするため，これまで，局所伝播法に基づくものを中心に，様々な制約解消系が研究されてきた．本論文では新たに，1次等式からなる制約階層を解く制約解消系HiRiseを提案する．HiRiseは，数値的アルゴリズムによる信頼性と，局所伝播法のようなインクリメンタルな解消による高速性を両立した，新しい制約解消法に基づくものである．この両立によって，従来の制約解消系では効率的に解けなかった，数千個の制約が連立されているような状況でも，パーソナルコンピューター上で数十〜数百ミリ秒で処理できるようになった．このため，HiRiseを利用することで，複雑で巨大なGUIアプリケーションを構築することが可能になる．

HiRise: An Incremental Constraint Solver for Constructing Graphical User Interfaces.

Hiroshi Hosobe, 文部省学術情報センター研究開発部, Department of Research and Development, National Center for Science Information Systems.

Satoshi Matsuoka, 東京工業大学大学院情報理工学研究科数理・計算科学専攻, Department of Mathematical and Computing Sciences, Graduate School of Information Science and Engineering, Tokyo Institute of Technology.

Akinori Yonezawa, 東京大学大学院理学系研究科情報科学専攻, Department of Information Science, Faculty of Science, University of Tokyo.

コンピュータソフトウェア, Vol.16, No.6(1999), pp.33–45.

[論文] 1999年6月1日受付.

1 はじめに

グラフィカルユーザーインターフェース(GUI)の構築における制約の有用性は，今日，広く認められつつある．事実，制約はすでに多くの研究用GUIシステムで提供され[4][20][21][22][27][28][34]，現在では商用システムへの普及も徐々に進んでいる†1．

GUI分野における制約の最大の用途は，グラフィカルレイアウトである．この用途での利点は，グラフィカルなオブジェクト群の間の幾何的な関係を制約で記述しておくことで，レイアウトの維持が自動化されて容易になることである．また，レイアウト以外にも，内部データに合わせて，グラフィカルオブジェクトの大きさを調節したり，場合によっては，内部データ同士の関係を管理するために，制約が用いられることもある．

制約を用いたGUIの研究における近年の最重要テーマの1つは，制約の**優先度**の実現である[5][13][15][16][17][19][22][30][31][33][35]．直観的には，制約に優先度がある場合，できるだけ優先度の高い制約が満たされる．制約の優先度に関する具体的な理論としては，ThingLab II [22]で導入された**制約階層**[5][7]が特に有名である．制約階層の理論では，制約の優先度を有限個の**強さ**で表す．その強さに基づき，1つの制約階層は，階層的な**レベル**に分割された，制約の集合として表現される．そして，制約階層の解は，できる

†1 例えば，VRML 97 [11]におけるイベントのルーティングや，Java Studio [32]におけるコンポーネント間の接続などは，あらかじめ解く方向の定められた，単方向のデータフロー制約と見なすことができる．

だけ強い制約が満たされるように決定される．

制約階層の主要な特徴の1つは，DeltaBlueアルゴリズム[13]など，**局所伝播法**と呼ばれる効率的な制約解消法が盛んに研究されてきた点である[16][19][22][30][31][33][35]．これらの多くは，インタラクティブなGUIへの応用を目的として設計され，さらにその一部は，制約解消系として実装・配布されており，このことが，GUI分野における制約階層の普及を促進した．

しかし，局所伝播法によるアルゴリズムはデータフロー制約を扱うように設計されているため，1次等式などの代数的制約を扱う際に誤動作の可能性があるなどの問題があった[2]．これに対処するため，我々は[18]において，**階層線形系**とその解消アルゴリズムを提案した．これにより，1次等式からなる制約階層については，極めて簡潔な枠組で，数値的アルゴリズムによる信頼性の高い制約解消を実現できることを示した．

一方，優先度以外の重要な研究テーマとして，制約解消のスケーラビリティが挙げられる．通常，制約解消には，ある程度大きな計算量が必要である．しかし，制約を用いたGUIでは，グラフィカルオブジェクトの追加・削除などに応じて，制約の集合が変更されるので，そのたびに制約の集合全体を最初から解いていては，制約解消系は，制約の増加にほとんど対応することができなくなってしまう．このような制約解消のスケーラビリティの問題を解決する鍵となるのが，**インクリメンタル性**である．インクリメンタルな制約解消では，制約の集合の修正に応じて必要最小限の部分だけを解き直す，差分的な処理が行われる．インクリメンタルな制約解消は，従来の局所伝播法アルゴリズムの多くで，その主題とされていた[13][16][19][22][30][31][33][35]．

しかし，数値的に制約階層を解くアプローチでは，インクリメンタル性はまだ新規な問題である．そこで本研究では，階層線形系の枠組を利用して，数値的アルゴリズムによるインクリメンタルな制約解消を実現した．実際に設計・開発した制約解消系HiRise[†2]は，

- 制約階層の数値的な解消による信頼性と
- 局所伝播法のようなインクリメンタルな解消による高速性

を両立した，新しい制約解消法に基づくものである．この両立によって，従来の制約解消系では効率的に解けなかった，数千個の制約が連立されているような状況でも，標準的なパーソナルコンピューター上で数十〜数百ミリ秒で処理できるようになった．このため，HiRiseを使えば，従来の制約解消系を使った場合にはインタラクティブな操作が困難だった，複雑で巨大なGUIアプリケーションでも構築することが可能になる．

†2 HiRiseとは，HieRarchical linear system engineの略である．

2 関連研究

既存の制約階層のための制約解消アルゴリズムの多くは，以下の3種類の方式に分類できる[16]．

精製法： 強いレベルから順に充足することで，制約階層を解消する，最も単純な方式である．

局所伝播法： 制約階層の構造に基づいて制約をスケジューリングし，一意に充足可能な制約を順に選んで解いてゆくデータフロー方式である．特徴的なのは，この順序が必ずしも強さの順ではなく，弱い制約を強い制約よりも先に解く場合がある点である．

最適化アプローチ： 強さを重みに対応させ，制約階層を，線形計画問題などの1つの最適化問題に変換して解く方式である．

精製法の具体例としては，以下のようなものがある．Borningらの初期のシステム[5]では，各レベルごとに弛緩法を適用していた．Orangeアルゴリズム[13]では，各レベルにシンプレックス法を適用することで，1次等式と1次不等式からなる制約階層を充足している．DeltaStar [36]は，Orangeをより一般化したアルゴリズムで，複数の制約階層の解の決定法に対応している．Indigo [3]は，区間伝播法を利用することで，循環的関係のない，不等式制約を含む制約階層を解くことができる．

局所伝播法に基づく制約解消アルゴリズムには，以下のものが挙げられる．Blue [22]は，データフロー制約からなる制約階層を解く最初のアルゴリズムで，DeltaBlue [13][22]は，Blueをインクリメンタルにしたものである．SkyBlue [30][31]は，DeltaBlueをより一般化し，制約の循環的関係と，多出力の制約に対応したものである．QuickPlan [35]は，データフロー

制約からなる制約階層をインクリメンタルに解くアルゴリズムで，循環的関係を生じずに解く方法がある場合に，それを必ず発見することを保証している†3．*DETAIL* [16][19]は，局所伝播法の枠組を拡張して，最小2乗比較子と呼ばれる解の決定法を可能にしたアルゴリズムで，同時に，制約の循環的関係にも対応している．Houria [9][10]は，*DETAIL* に類似の枠組で，*DETAIL* とは異なる解の決定法の導入を試みたものである．

最適化アプローチを採用した制約解消系には，シンプレックス法を用いて，局所比較子による解を求めるCassowary [1][8]と，アクティブセット法を用いて，最小2乗比較子による解を求めるQOCA [8][23]がある．

上述の3つの方式に分類されないものとしては，制約階層を部分グラフに分割して，部分グラフごとに適切なsubsolverを呼び出すという，「メタ」な制約解消系であるUltraViolet [6]や，プロジェクションによる静的解消法[14]，探索方式のIHCS [26]などが挙げられる．

後に7節で，HiRise制約解消系の得失について，代表的な関連研究との比較をしながら議論する．

3 HiRise制約解消系の概要

本論文で提案するHiRise制約解消系は，我々が[18]で提案した制約解消系と同等の機能を提供しながら，速度を大幅に向上したものである．

HiRiseで扱う制約は，[18]の制約解消系と同様，以下のようなものである．

- 個々の制約には，その優先度として強さが与えられる．強さは有限個の段階からなり，例えば，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`strong`，`medium`，`weak`のような選好制約のレベルが続く†4．
- 制約には1次等式，**stay**，**edit**の3種類がある．stayは変数の値を一定にする制約であり，editは変数の値を繰り返し更新するための制約である．

ただし，[18]の制約解消系と同様，内部的に，強さは全順序の優先度へ変換される．このため，例えば，ある2つの制約が同じ強さを持っていたとしても，内部的にはどちらか一方が常に優先されることになる．また，stay/edit制約も1次等式に変換されて処理が行われる．

HiRiseの制約解消アルゴリズムは，前節で分類した，制約解消の3つの方式には特に当てはまらない．しかし，制約解消をインクリメンタルに行うために，従来の局所伝播法アルゴリズムで利用されていた，walkabout strength [13]の手法を採り入れている．このため，HiRiseは，データフローではなく数値的アルゴリズムを採用しているとはいえ，局所伝播法とは密接な関係がある．

4 アルゴリズム

本節では，HiRise制約解消系のアルゴリズムについて述べる．

4.1 概　要

HiRise制約解消系では，階層線形系のLU分解をその主要データ構造としている．これは，我々が[18]で提案したアルゴリズムと同様，**階層独立**である1次等式制約(より優先度の高い制約に対して係数ベクトルが線形独立である制約)の係数ベクトルを集めて行列を作り，それをLU分解したものである．以下では，このようにLU分解の対象となる制約を**有効**な制約と呼び，階層独立でない，すなわち階層従属であるためにLU分解の対象にならない制約を**無効**な制約と呼ぶ．

インクリメンタル性の実現のために，HiRiseのアルゴリズムは，[18]のものとは大きく異なっている．具体的には，以下のような問題を解決する必要があった．

- 有効な制約の管理．新しい制約を追加する場合，それを有効にすべきかどうか素早く判断し，さらに，それを有効にする場合には，それまで有効だった多数の他の制約の中から，代わりに無効にすべきものを高速に発見する必要がある．
- LU分解の修正．有効な制約の集合を更新するとき，対応するLU分解を効率的に修正する必要があ

†3 循環的関係を生じずに解く方法が常にあるとは限らない．本研究は，むしろ，そのような方法がない場合を重視したものである．

†4 HiRise制約解消系では，強さの段階数はコンパイル時に決めることができる．デフォルトでは16段階である．

る．

以下では，まず先に LU 分解とその修正のための基本的技法について説明し，その後，有効な制約を管理して，インクリメンタルに制約解消を行うアルゴリズムについて述べる．

4.2 LU 分解

HiRise における最も基本的な LU 分解は以下のようなものである．

$$BP_1U_1P_2U_2\cdots P_nU_n = L \qquad (1)$$

ここで，B は有効な制約の係数ベクトル $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, …, $\boldsymbol{b}_n$ からなる $n \times n$ 行列であり，P_i は以下のような $n \times n$ 置換行列であり，U_i は以下のような $n \times n$ 上 3 角 η 行列であり，L は全ての対角要素が 1 である下 3 角行列である†5．

$$B = \begin{pmatrix} \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_n \end{pmatrix}$$

$$P_i = \left(\begin{array}{c|c|c|c|c} E_{i-1} & & & & \\ \hline & 0 & & 1 & \\ \hline & & E_{j-i-1} & & \\ \hline & 1 & & 0 & \\ \hline & & & & E_{n-j} \end{array}\right)$$

$$U_i = \left(\begin{array}{c|c|c} E_{i-1} & & \\ \hline & \frac{1}{b'_{ii}} & -\frac{b'_{i,i+1}}{b'_{ii}} \quad \cdots \quad -\frac{b'_{in}}{b'_{ii}} \\ \hline & & E_{n-i} \end{array}\right)$$

$$L = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \ddots & \ddots & \\ l_{n1} & l_{n2} & & 1 \end{pmatrix}$$

直観的に，(1) は，P_1, U_1, P_2, U_2, …, P_n, U_n が B を下 3 角行列 L に変換すると理解できる．これはちょうど，Gauss の消去法[29]における行と列の関係を逆転したものである．その計算過程で，P_i と U_i は以下のような役割を果たしている．

- P_i は，(i,i) 要素が非零になるように第 i 列と第 j 列を交換する．
- U_i は，(i,i) 要素を 1 にし，全ての $k > i$ について (i,k) 要素を 0 にする．

この LU 分解を利用すれば，$B\boldsymbol{x} = \boldsymbol{c}$ を $\boldsymbol{x}$ について容易に解くことができる．n ベクトル $\boldsymbol{y}$ を導入して，

$$\boldsymbol{x} = P_1U_1P_2U_2\cdots P_nU_n\boldsymbol{y} \qquad (2)$$

とおくと，$L\boldsymbol{y} = \boldsymbol{c}$ が成立する．このため，前進代入[29]で $L\boldsymbol{y} = \boldsymbol{c}$ を $\boldsymbol{y}$ について解いた後，(2) を計算する (P_i と U_i の特殊性のために容易) ことで，$\boldsymbol{x}$ が得られる．

4.3 LU 分解の修正

LU 分解を修正する技法について述べる．これは，線形計画法のために考案された Forrest と Tomlin の方法[12]からヒントを得たものである．

この技法では，B の LU 分解 (1) を利用して，次の新しい LU 分解を求める．

$$B'P_1U_1P_2U_2\cdots P_nU_nQU'_iU'_{i+1}\cdots U'_n = L'$$

ここで，B' は以下のように B から行を削除し，新しい行を下に付加して得られる行列であり，Q は以下のような $n \times n$ 置換行列であり，U'_i は U_i と同様の行列である．

$$B' = \begin{pmatrix} \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_{i-1} \\ \boldsymbol{b}_{i+1} \\ \vdots \\ \boldsymbol{b}_n \\ \boldsymbol{b}'_i \end{pmatrix}$$

$$Q = \left(\begin{array}{c|c|c} E_{i-1} & & \\ \hline & & 1 \\ \hline & E_{n-i} & \end{array}\right)$$

B' に $P_1U_1P_2U_2\cdots P_nU_nQ$ を掛けることで，次の行列が得られる．

$B'P_1U_1P_2U_2\cdots P_nU_nQ$

†5 本論文において，E_k は $k \times k$ 単位行列を表す．また，書き下された行列の空白部分は 0 で埋められているとする．

$$= \begin{pmatrix} 1 & & & & & & \\ \vdots & \ddots & & & & & \\ l_{i-1,1} & & 1 & & & & \\ l_{i+1,1} & \cdots & l_{i+1,i-1} & l_{i+1,i} & 1 & & \\ \vdots & & \vdots & \vdots & & \ddots & \\ l_{n1} & \cdots & l_{n,i-1} & l_{ni} & l_{n,i+1} & & 1 \\ b''_{i1} & \cdots & b''_{i,i-1} & b''_{ii} & b''_{i,i+1} & & b''_{in} \end{pmatrix} Q$$

$$= \begin{pmatrix} 1 & & & & & & \\ \vdots & \ddots & & & & & \\ l_{i-1,1} & & 1 & & & & \\ l_{i+1,1} & \cdots & l_{i+1,i-1} & 1 & & & l_{i+1,i} \\ \vdots & & \vdots & & \ddots & & \vdots \\ l_{n1} & \cdots & l_{n,i-1} & l_{n,i+1} & & 1 & l_{ni} \\ b''_{i1} & \cdots & b''_{i,i-1} & b''_{i,i+1} & & b''_{in} & b''_{ii} \end{pmatrix}$$

さらに，$i \leq k < n$についてU'_kを掛けて(k,n)要素を消去し，U'_nを掛けて(n,n)要素を1にすることで，容易に新しい下3角行列L'を得ることができる．

明らかに，この技法を繰り返して，LU分解を何度も修正することができる†6．

以下では，簡単のため，LU分解される行列を常にBと呼ぶ．

4.4 制約の追加

新しい制約を階層線形系に追加する方法について述べる．階層線形系に制約を追加するとき，次の2つの場合があり得る．

- 新しい制約は階層独立であり，有効にすべきである．この場合，これまで階層独立だった，より弱い制約で階層従属に変化したものを，対応する行をBから削除することで無効にし，新しい制約の係数ベクトルを下に付け加える†7．その後，新しいLU分解をインクリメンタルに求める．
- 新しい制約は階層従属であり，有効にすべきでない．この場合，現在のLU分解を修正する必要はない．

制約追加の際にインクリメンタルに制約解消するアルゴリズムは以下のようになる．

1. 新しい制約を追加したために無効になる制約を予測する．そのような制約がない場合，アルゴリズムはLU分解を修正せずに終了する†8．予測された制約がある場合，対応する行のBにおけるインデックスをkとする．そして，Bから第k行を削除し，4.3節の技法に従って，残りの行について部分的なLU分解をインクリメンタルに求める．
2. 変換行列$P_1U_1P_2U_2\cdots$を新しい制約の係数ベクトルに掛けることで，$(a_1\ a_2\ \cdots\ a_n)$を求める．
   (a) a_nが0でない場合，予測は正しい．そこで，現在のBに対して，新しい制約の係数ベクトルを下に付加して，最終的なLU分解を求める．
   (b) そうでない場合，予測は誤っている．この際は，式(1)により非インクリメンタルにLU分解を求める†9．

このアルゴリズムの鍵は，無効にすべき制約の予測である．このために，DeltaBlue [13]などの局所伝播法アルゴリズムで用いられているwalkabout strengthの手法を採り入れた．HiRiseでは，walkabout strength w_jは変数x_jに割り当てられ，図1のように計算される．直観的には，変数のwalkabout strengthは，その変数の値を計算するために使われる制約の中で最も弱いものの階層線形系におけるインデックスを差す．

無効にすべき制約の予測について説明する．元の階層線形系の第$(i-1)$制約と第i制約の間に，新しい制約が挿入されるとする．係数が0の変数を省略すること

†6 これにより，変換行列$P_1U_1P_2U_2\cdots$の列は段々と伸びていくことになる．このことは速度低下の要因になり得るが，我々の使用の範囲で特に問題になったことはない．また，実際のHiRise制約解消系では，ガーベッジコレクションのように，不要な変換行列を除去する機能も提供している．

†7 この操作のために，Bの行の順序は，元の階層線形系で対応する制約の順序とは異なってくる．

†8 本アルゴリズムにおいて，「無効にすべき制約がない」という予測は常に正しい．一方，「無効にすべき制約がある」という予測は誤っている可能性がある．ただし，後者の予測の正誤については，その後のアルゴリズムの過程で判断可能である．

†9 この場合の計算には時間が掛かるが，このような状況が生じるのは稀である．また，実際のHiRise制約解消系では，完全な非インクリメンタル処理ではなく，必須制約があることを利用して，可能な限り，それまでのLU分解を利用するようにしている．

```
 1  for i ← 1 to n do {
 2   w_i ← (b_i に対応する制約のインデックス);
 3   for j ← 1 to i − 1 do
 4    if l_ij ≠ 0 then w_i ← min(w_i, w_j); }
 5  for k ← (変換行列の個数) to 1 step −1 do {
 6   T_k ← (第k変換行列);
 7   if T_k の形が P である then {
 8    T_k が交換する列のインデックスを j, j′ とする;
 9    swap(w_j, w_j′); }
10   else if T_k の形が Q である then {
11    j ← (T_k が右側に移動する列のインデックス);
12    w ← w_n;
13    for j′ ← n to j + 1 step −1 do
14     w_j′ ← w_j′−1;
15    w_j ← w; }
16   else { // T_k の形が U である
17    i ← (T_k が変更する行のインデックス);
18    for j ← i + 1 to n do
19     if T_k(i, j) ≠ 0 then w_i ← min(w_i, w_j); } }
```

図 1 walkabout strength の計算

で，追加される制約は次のように表現できる．

$$a_{j_1}x_{j_1} + a_{j_2}x_{j_2} + \cdots + a_{j_l}x_{j_l} = c$$

ここで，

$$k = \min(w_{j_1}, w_{j_2}, \ldots, w_{j_l})$$

とすると，以下のようにして予測ができる．

1. k が i より小さい場合，無効にすべき制約はない．
2. そうでない場合，元の階層線形系の第 k 制約が無効にすべき制約の候補となる．

この方法により，追加された制約が参照する変数の値を決めるために使われた制約の中で最も弱いものを見つけることができる．

4.5 制約の削除

階層線形系から制約を削除するアルゴリズムは，以下のようになる．

1. 削除すべき制約が無効である場合，アルゴリズムは LU 分解を修正せずに終了する．そうでない場合，行列 B から対応する行を削除し，残りの行について LU 分解を部分的に求める．
2. 削除された制約より弱い無効な制約の中で最強のものを選ぶ．その係数ベクトルに変換行列 $P_1U_1P_2U_2\cdots$ を適用して得たベクトルの第 n 要素が 0 でない場合，その制約を有効にする．そうでない場合，次に強い無効な制約を選び，この処理を繰り返す．

4.6 時間計算量

ここでは，HiRise の時間計算量を与える．まず，本節で述べたアルゴリズムの構成要素の時間計算量は以下のようになる．ただし，変数の個数を n，制約の個数を m とする．

- 4.2節で述べた，LU 分解を最初から求める方法の計算量は，$O(mn^2)$ である．
- 4.3節で述べた，LU 分解をインクリメンタルに修正する技法の計算量は，$O(n^2)$ である．
- 4.2節の最後に述べた，LU 分解を用いて変数の値を計算する方法の計算量は，$O(n^2)$ である．
- 4.4節で述べた，変数の walkabout strength を計算する手法の計算量は，$O(n^2)$ である．

これらの計算量を総合すれば，HiRise に対する各操作における時間計算量を以下のように求めることができる．

- 制約解消を最初から行う際の計算量は，$O(mn^2)$ である．
- 新しい制約を追加する際の計算量は，walkabout strength による予測が成功した場合には $O(n^2)$ であり，失敗した場合には $O(mn^2)$[†10]である．
- 既存の制約を削除する際の計算量は，$O(m'n^2)$ である．ただし，m' は，削除された制約の代わりに充足されるべき制約の候補として，実際に試されるものの個数である．
- edit 制約による入力に応じて，制約の集合自体は変更せずに，変数の値のみを再計算する際の計算量は，$O(n^2)$ である．

5 実　装

前節のアルゴリズムに基づき，制約解消系を C++ と Java で 2 種類実装した．この際，LU 分解などを表す行列は，実際のアプリケーションにおいて疎になる傾向

†10 4.4節の脚注でも述べたように，必須制約の存在を利用することで，この計算量の m の部分を小さくできる．直観的には，必須制約の割合が多いほど，これは $O(n^2)$ に近付くことになる．

```
HRVar x(1), y, z;   // 変数 (x の初期値は 1)
// x に関する強さ 0(必須) の stay 制約
HRStay stay(0); stay.add(x);
// y に関する強さ 1 の edit 制約
HREdit edit(1); edit.add(y);
// 強さ 2 の 1 次等式制約 x + 2 * y - z = 3
HRLinear linear(2);
linear.add(x, 1).add(y, 2).add(z, -1)
    .setConst(3);
HRSolver solver;    // HiRise 制約解消系
solver.add(stay);   // 制約を追加
solver.add(edit);
solver.add(linear);
solver.plan();      // LU 分解の計算
edit.set(2);        // edit によって y に 2 を代入
solver.execute();   // LU 分解による代入計算
// 変数の値を表示
printf("x = %f, y = %f, z = %f\n", *x, *y, *z);
edit.set(3);        // edit によって y に 3 を代入
solver.execute();   // 再実行
printf("x = %f, y = %f, z = %f\n", *x, *y, *z);
```

図 2　HiRise を用いたプログラムの例

```
x = 1.000000, y = 2.000000, z = 2.000000
x = 1.000000, y = 3.000000, z = 4.000000
```

図 3　図 2 のプログラムの実行結果

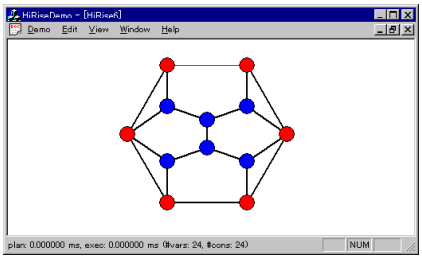
```
HRParser parser;
parser.createVar("x", 1);
parser.createVar("y");
parser.createVar("z");
HRCon* stay = parser.createCon(0, "stay(x)");
HRCon* edit = parser.createCon(1, "edit(y)");
HRCon* linear =
    parser.createCon(2, "x + 2 * y - z = 3");
```

図 4　パーザーの利用例

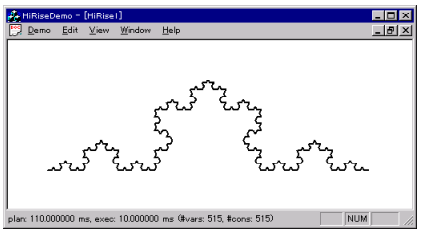
があるので，配列ではなくリストとして実装した．これにより，巨大な制約階層において時間とメモリーを節約できるようになった．

HiRise 制約解消系を利用する際，変数と制約は C++ または Java のオブジェクトとして表現する．例えば，C++ 版を使う場合，図 2のようにプログラムを記述することになる．これを実行すると，図 3のような結果が得られる．また，C++ 版では，制約をより容易に記述するためのパーザーを提供している．これを利用すれば，変数と制約を図 4のように記述できる．

HiRise 制約解消系は，標準的な C++ と Java で実装されているので，多様なプラットフォーム上で利用できる†11．図 5は，Microsoft Windows 95/98/NT 4.0 上で動作する C++ 版のサンプルアプリケーションのスクリーンスナップショットである．図 5(a) のアプリケーションでは，ユーザーがノードの追加や，その位置の固定，エッジの方向の固定などの操作を行い，グラフを編集できる．一方，図 5(b) のアプリケーションは，フラクタル図形の 1 種である Koch 曲線の近似を行い，それをユーザーに操作させるものである．



(a) グラフの編集



(b) Koch 曲線の操作

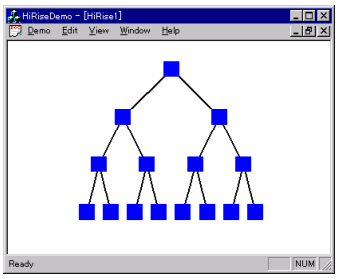
図 5　HiRise のサンプルアプリケーション

なお，HiRise 制約解消系とそのサンプルアプリケーションは，フリーソフトウェアとして配布されている [24] [25].

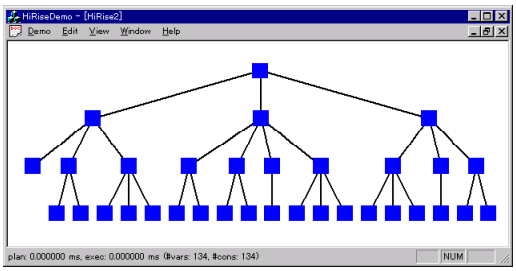
6 実　　験

HiRise 制約解消系に関する性能評価の一環として，巨大で複雑な制約階層を生成するアプリケーションを用いて実験を行った．プログラムのコンパイルには，Microsoft Visual C++ 6.0 を最適化オプションを付けて利用した．実験の環境には，Windows NT 4.0 Workstation で動作する，Intel Pentium II 450

†11　C++ 版は GNU CC v2.6.3 以降と Microsoft Visual C++ 5.0 以降でコンパイルでき，Java 版は JDK 1.1 相当の処理系でコンパイルできる．



(a) 2 分木の編集



(b) 一般の木の編集

図 6　実験に用いたアプリケーション

MHz を搭載した PC/AT 互換機を使用した．

性能比較の対象としては，SkyBlue [30][31]と Cassowary [1][8]の 2 つの制約解消系を採用した．具体的には，SkyBlue については，原論文の著者が開発した C 版の実装に，Crout 法[29]による LU 分解を用いた cycle solver を組み込んだものを使用した．また，Cassowary については，原論文の著者らが配布している C++ 版の実装を利用した．

以下では，実際に GUI アプリケーションを使って行った 2 つの実験において，各制約解消系が処理に要した時間を与える．最初の実験は，図 6(a) の 2 分木の編集のアプリケーションを用いたものである．このアプリケーションでは，葉の間隔が一定になるようにノードを配置しており，正しく解くためには，制約の連立が不可欠である．制約の定義において，各ノードの位置は，垂直方向は階層的に，水平方向は隣り合う部分木のバウンディングボックスが隣接するように決められている．この例では，レイアウトのための制約は，全て必須制約となっており，制約階層のほとんどが必須制約からなっている．

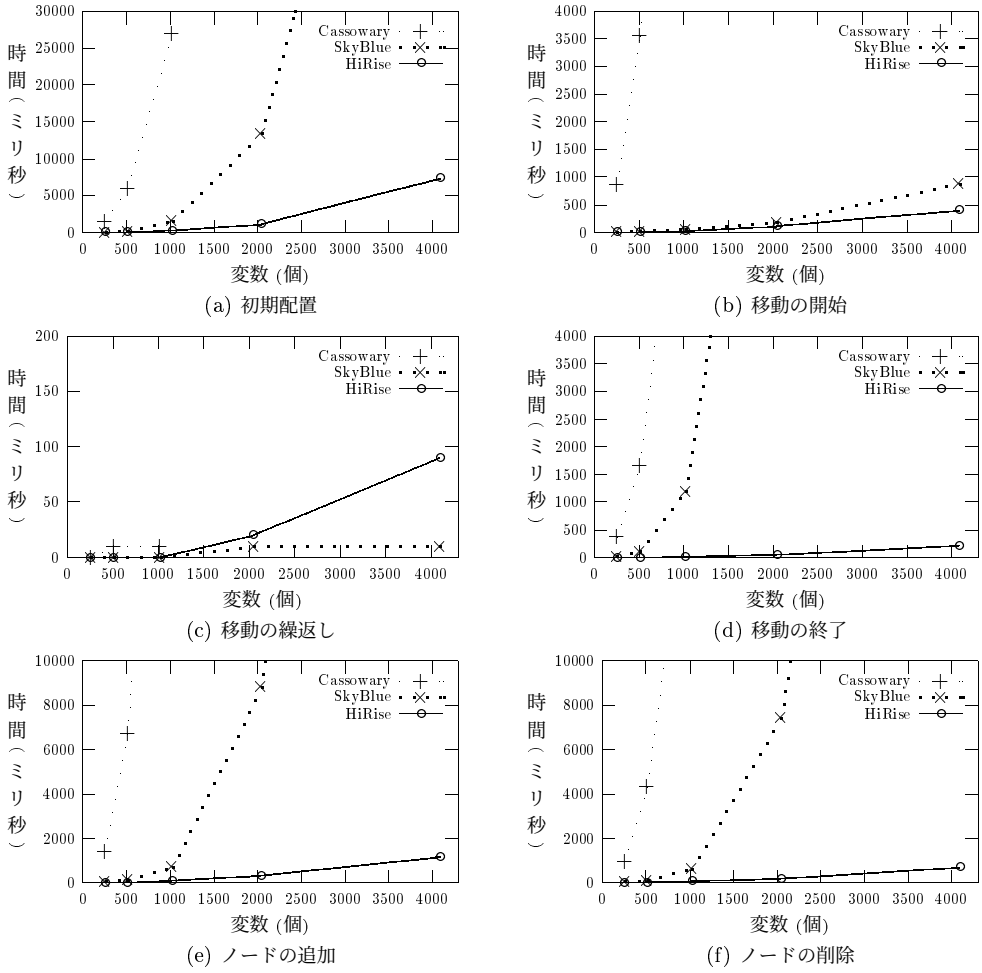
図 7のグラフは，このアプリケーションにおける処理に伴って，各制約解消系が計算に必要とした時間を示している．グラフの横軸は，実験に使用した制約階層における変数の個数を表したもので，通常の状態で，これは制約の個数に等しい．一方，グラフの縦軸は，制約階層を解くのに要した時間をミリ秒で表したものである (ただし，測定の精度は 10 ミリ秒である)．実際に実行時間を測定した処理は，(a) 2 分木の初期配置，(b) ノードのドラッグによる 2 分木の移動の開始，(c) 移動の繰返し，(d) 移動の終了，(e) 新しいノードの追加，(f) 既存のノードの削除の 6 種類である．

実験の結果，HiRise は，SkyBlue や Cassowary に比べて，全体的に高い処理速度を示した．例外は，(b) の移動の繰返しに伴う計算が，SkyBlue よりも遅かったことだが，特に問題になる程の遅さではなかった．一方，SkyBlue は，処理の種類によって，速さにばらつきがあった．また，Cassowary は，変数が千個を超えるあたりから速度が大幅に低下し，インタラクティブな動作に支障を来した．

もう 1 つの実験は，図 6(b) の一般の木の編集のアプリケーションを用いたものである．基本的には，先の 2 分木のアプリケーションを拡張したものだが，木のレイアウトの定義を多少単純化している．また，レイアウトの制約の一部を選好制約とすることで，制約全体の内で必須制約の占める割合を 6 割程度にした．木の初期形状は，高さと，1 個のノードが持ち得る子ノードの最大数を指定し，乱数を用いて生成した．ただし，この実験では，同じ乱数の種を使用することで，各制約解消系に対して，同じ形の木を与えるようにした．

図 8のグラフは，その実験結果を示している．グラフの読み方と実験の内容は，先の 2 分木の場合と同様である．この実験でも，HiRise が全体的に速い傾向があった．一方，SkyBlue は，先の実験ほどではなかったが，ここでも速度のばらつきが認められた．Cassowary は，この実験においても，変数が千個を超えると，ほとんど対応できなくなった．

これらの実験から，HiRise 制約解消系が，数千個の 1 次等式制約が連立されているような制約階層で，高い効率を示すことが確認された．



(a) 初期配置　(b) 移動の開始

(c) 移動の繰返し　(d) 移動の終了

(e) ノードの追加　(f) ノードの削除

図 7　2 分木の編集のアプリケーションにおける制約解消の時間

7 議　論

本節では，HiRise 制約解消系の得失について，代表的な関連研究との比較をしながら議論し，最後に HiRise の適正な用途についてまとめる．議論の際，特に表現能力と処理速度の 2 つの面に焦点を当てる．その理由は，一般に制約解消系の表現能力と処理速度との間には，トレードオフが存在するからである．

なお，以下では，精製法による制約解消については議論の対象としない．この方式は，一般的な数値計算手法を採り入れ易いために表現能力は高いが，実際の制約解消系 [3] [5] [13] [36]の多くは初期のもので，効率面に問題があり，現実的な GUI アプリケーションには向かないためである†12．

7.1 表現能力

前節の実験における SkyBlue [30] [31]のように，従来の局所伝播法による制約階層の解消系でも，cycle solver や subsolver などを導入して，1 次等式などの連立に対応できるものもある．しかし，これらは，誤動作を引き起こす危険性がある [2]など，信頼性の上で問題がある．例えば，前節で紹介した木の編集のアプリケーションでは，あるノードの位置を強い制約で固定し

†12 Indigo [3]は比較的新しいが，これは UltraViolet [6]の subsolver として開発されたもので，適用領域が限られている．また，Indigo は非インクリメンタルなアルゴリズムであり，効率的であるとは言い難い．

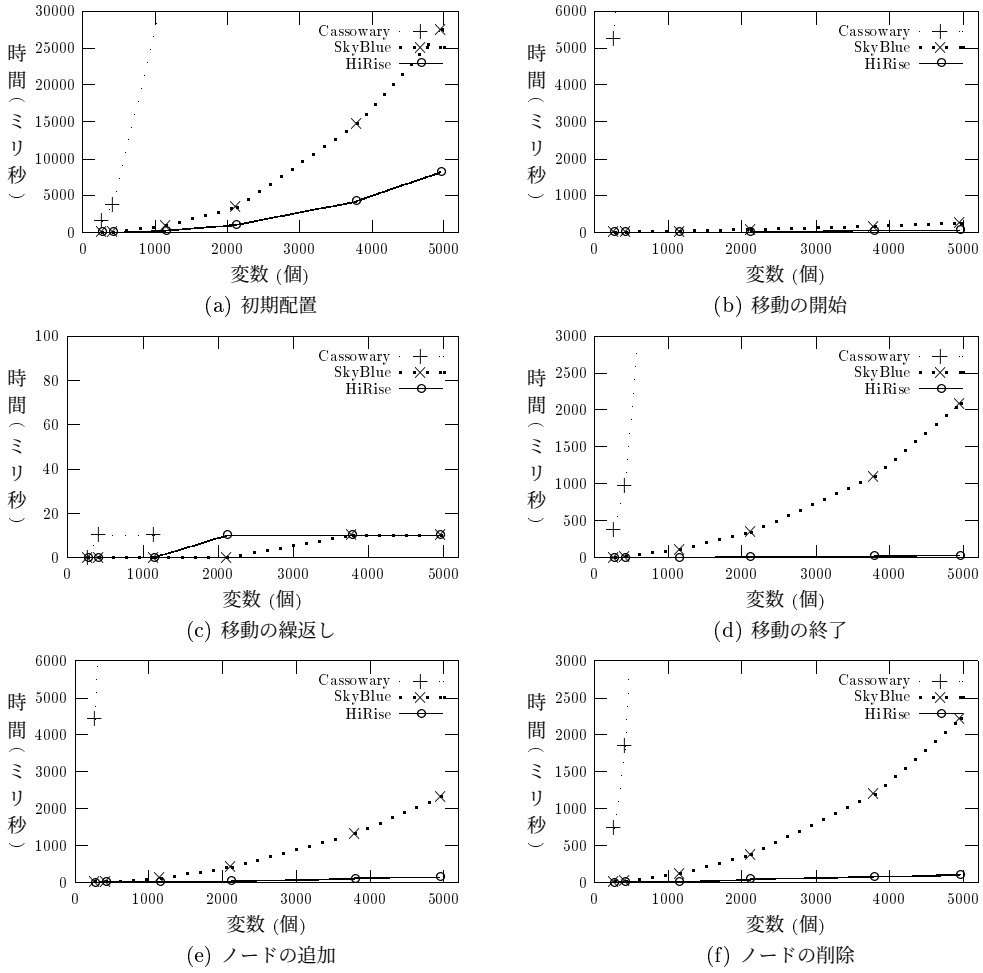


図 8　一般の木の編集のアプリケーションにおける制約解消の時間

ておいた状態で，それと同じレベルにあるノードをドラッグした場合，理論上，それらのノードは水平を保つ必要があるにもかかわらず，SkyBlue では，その条件を満たすことができない†13．一方，HiRise は，誤差の累積などによる例外を除けば，与えられた制約階層を正しく解くことを保証しており，信頼性の面で優位である．

局所伝播法による制約解消系では，内部で制約のデータフローに循環的関係が生じない限り (すなわち，制約が連立されない限り)，1 次等式に限らず，様々な制約†14を扱うことができる [13][16][19][22][30][31][33][35]．一方，HiRise では，制約を 1 次等式，stay，edit のみに制限しており，それら以外の制約を入力することができない．

Cassowary [1][8]などの最適化アプローチによる制約解消系は，1 次等式だけでなく，1 次不等式を扱うこともできる．この方式では，制約の連立も正しく扱うことができ，表現能力の面では，HiRise に比べて優れているといえる†15．

†13 内部的に，エラーが生じていることは検出できる．しかし，そのエラーを回復する手段を提供することが困難である．

†14 多くの場合，制約は，メソッドと呼ばれる手続きの組合せとして表現される．

†15 HiRise は，1 次不等式制約が扱えないので，あるオブジェクトが他のオブジェクトの右にあるといった制約が記述できない (もちろん，10 ドット右にあるというように，等式で書ける制約は記述可能である)．また，

より一般的な観点から考えると，HiRiseでは，制約の優先度を内部的に全順序で管理していることが，問題点として指摘され得る．実際に，このために，HiRiseが，ユーザーにとって好ましくない解き方をする可能性がある．具体的には，例えば，アプリケーションを使っている際に，水平または垂直に近い直線上に制約されたオブジェクトのドラッグが困難になるといった問題を起こすことがある．ただし，この問題は，HiRiseだけでなく，SkyBlueやCassowaryなど，局所的比較子[5][7][13][16][17][36]と呼ばれる解の決定法を用いた制約解消系に共通のものである．この問題を解決する一般的な方法は，*DETAIL* [16][19]やQOCA [8][23]などのように同一レベル内で最小2乗法を適用することである．

しかし，この問題は，通常，それほど意識する必要はなく，それが生じ得る状況を，アプリケーション等の製作段階で認識できることも多い．このため，限定的に，真に同じ強さの制約を指定できるようにすれば，実用上十分に対処可能であるともいえる．本論文で詳しくは述べないが，すでに我々はHiRiseにこの方法を導入して，一定の成果を挙げている[25]．

7.2 処理速度

局所伝播法による制約解消系は，制約間に循環的関係が生じなければ(すなわち，制約が連立されなければ)，極めて高速に制約階層を解けることが多い．しかし，制約間に循環的関係が生じる(すなわち，制約が連立される)場合，以下の2つの理由により，処理速度が大幅に低下することがある．

- 制約解消アルゴリズム自体が，本質的に，循環的関係を効率的に扱えないことがある．例えば，SkyBlueでは，循環的関係がある場合の時間計算量が，制約の個数に対して指数的である[30][†16]．
- 循環的関係のある部分でインクリメンタルな制約解消ができず，cycle solverやsubsolverの内部での計算に時間が掛かることがある．その理由は，巨大な循環的関係が突然発生したり，消滅したりする可能性があり，cycle solverやsubsolverと連続的な連携を保つことが難しいためである．

一方，HiRiseでは，制約が連立されていたとしても，ほとんどの状況において，安定した速さで制約解消を行うことができる．4.4節で述べたように，例外的に，インクリメンタルな制約解消を適用できず，非インクリメンタルな処理を行う場合があるが，その場合でも，最悪で，全ての制約を解き直した場合と同程度の時間で抑えられる[†17]．

最適化アプローチによる制約解消系は，前節の実験結果からわかるように，現状では，表現能力の向上と引き換えに，効率を犠牲にしているといわざるを得ない．

7.3 HiRiseの適正用途

以上の議論から，HiRiseは，以下のような条件を満たすアプリケーションに適しているといえる．

- 1次等式，stay，editからなる制約階層で記述される．
- 制約が連立している．
- 制約の個数が数千のオーダーである．

このようなアプリケーションで，HiRiseは，処理速度と信頼性の両面において，他の制約解消系よりも優れている。

また，それ以外にも，HiRiseのこのような特性を活かして，UltraViolet [6]のような「メタ」な制約解消系のsubsolverとして組み込むという用途も考えられる．この場合，メタな制約解消系が制約階層を自動的に分割し，HiRiseに適した部分を割り当てるという形になる．

8 おわりに

本論文では，1次等式からなる制約階層のための制約解消系HiRiseを提案した．HiRiseは，数値的アルゴ

特定のウィンドウの中にグラフィカルオブジェクト群を閉じ込めるといった制約も記述できないが，このような場合については，制約解消の結果を拡大・縮小・移動したり，ウィンドウにスクロールバーを取り付けたりするなどの方法で，ある程度対処できる．

†16 実際に，本研究の実験で用いた，SkyBlueを組み込んだアプリケーションでも，インタラクティブな操作に対して，初期配置で掛かった以上の時間を費やすような状況を観察した．

†17 局所伝播法の制約解消系にcycle solverやsubsolverを導入したものは，このような状況で遅くなるとは限らないが，その一方で，誤動作する可能性がある．

リズムによる信頼性と，局所伝播法のようなインクリメンタルな解消による高速性を両立した，新しい制約解消法に基づいている．実際に， HiRise を使うことで，数千個の制約が連立されているような状況でも，パーソナルコンピューター上で数十〜数百ミリ秒で処理できることを実験により確認した．

今後は， HiRise 制約解消系のさらなる性能向上のために，階層線形系を表す行列の疎性を積極的に利用することを検討している．特に，疎行列のオーダリング法を前処理として採り入れる手法が有望であると考えている．

参考文献

[1] Badros, G. J. and Borning, A.: The Cassowary Linear Arithmetic Constraint Solving Algorithm: Interface and Implementation, Technical Report 98-06-04, Dept. of Computer Science and Engineering, University of Washington, 1998.

[2] Borning, A.: Problem with SkyBlue and Cycles, 1995. http://www.cs.washington.edu/research/constraints/solvers/skyblue-cycles.html.

[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. and Duisberg, R.: Constraint-Based Tools for Building User Interfaces, *ACM Trans. Gr.*, Vol. 5,No. 4(1986), pp. 345–374.

[5] Borning, A., Duisberg, R., Freeman-Benson, B., Kramer, A., and Woolf, M.: Constraint Hierarchies, *Proc. ACM OOPSLA*, 1987, pp. 48–60.

[6] Borning, A. and Freeman-Benson, B.: UltraViolet: A Constraint Satisfaction Algorithm for Interactive Graphics, *Constraints J.*, Vol.3, No.1 (1998), pp. 9–32.

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

[8] Borning, A., Marriott, K., Stuckey, P., and Xiao, Y.: Solving Linear Arithmetic Constraints for User Interface Applications, *Proc. ACM UIST*, 1997, pp. 87–96.

[9] Bouzoubaa, M., Neveu, B., and Hasle, G.: Houria: A Solver for Equational Constraints in a Hierarchical System, *Proc. of the Workshop on Over-Constrained Systems at CP'95*, 1995.

[10] Bouzoubaa, M., Neveu, B., and Hasle, G.: Houria II: A Solver for Hierarchical Constraint Systems, *Proc. of the Workshop on Constraints for Graphics and Visualization at CP'95*, 1995.

[11] Carey, R., Bell, G., and Marrin, C.: The Virtual Reality Modeling Language, International Standard ISO/IEC 14772-1:1997, The VRML Consortium Incorporated, 1997.

[12] Chvátal, V.: *Linear Programming*, Freeman, New York, 1983; 阪田省二郎, 藤野和建, 田口東訳: 線形計画法, 下巻, 啓学出版, 東京, 1988.

[13] Freeman-Benson, B. N., Maloney, J., and Borning, A.: An Incremental Constraint Solver, *Comm. ACM*, Vol. 33,No. 1(1990), pp. 54–63.

[14] Harvey, W., Stuckey, P., and Borning, A.: Compiling Constraint Solving using Projection, *Principles and Practice of Constraint Programming—CP'97*(Smolka, G.(ed.)), Lecture Notes in Computer Science, Vol. 1330, Springer-Verlag, 1997, pp. 491–505.

[15] 服部隆志: 編集操作におけるマクロと制約の統合, インタラクティブシステムとソフトウェア IV (日本ソフトウェア科学会 WISS'96) (田中二郎 (編)), 近代科学社, 1996 年, pp. 41–49.

[16] Hosobe, H.: *Theoretical Properties and Efficient Satisfaction of Hierarchical Constraint Systems*, PhD Thesis, Dept. of Information Science, University of Tokyo, 1998.

[17] Hosobe, H., Matsuoka, S., and Yonezawa, A.: Generalized Local Propagation: A Framework for Solving Constraint Hierarchies, *Principles and Practice of Constraint Programming—CP'96*(Freuder, E. C.(ed.)), Lecture Notes in Computer Science, Vol. 1118, Springer-Verlag, 1996, pp. 237–251.

[18] 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) (尾内理紀夫 (編)), 近代科学社, 1997 年, pp. 129–134.

[19] Hosobe, H., Miyashita, K., Takahashi, S., Matsuoka, S., and Yonezawa, A.: Locally Simultaneous Constraint Satisfaction, *Principles and Practice of Constraint Programming—PPCP'94*(Borning, A.(ed.)), Lecture Notes in Computer Science, Vol. 874, Springer-Verlag, 1994, pp. 51–62.

[20] Hudson, S. E. and Smith, I.: Supporting Dynamic Downloadable Appearance in an Extensible User Interface Toolkit, *Proc. ACM UIST*, 1997, pp. 159–168.

[21] Igarashi, T., Matsuoka, S., Kawachiya, S., and Tanaka, H.: Interactive Beautification: A Technique for Rapid Geometric Design, *Proc. ACM UIST*, 1997, pp. 105–114.

[22] Maloney, J. H., Borning, A., and Freeman-Benson, B. N.: Constraint Technology for User-Interface Construction in ThingLab II, *Proc. ACM OOPSLA*, 1989, pp. 381–388.

[23] Marriott, K., Chok, S. C., and Finlay, A.: A Tableau Based Constraint Solving Toolkit for Interactive Graphical Applications, *Principles and Practice of Constraint Programming—CP'98*(Maher, M. J. and Puget, J.-F.(eds.)), Lecture Notes in Computer Science, Vol. 1520, Springer-Verlag, 1998, pp. 340–354.

[24] 松岡 聡, 細部博史: 階層連立1次方程式のための効率的解消系の開発, 委託研究, 日本情報処理開発協会先端情報技術研究所, 1998. http://www.icot.or.jp/AITEC/FGCS/funding/97/gaiyou17.html.

[25] 松岡 聡, 細部博史: 階層連立1次方程式のための制約解消系パッケージHiRiseの改良と移植, 委託研究, 日本情報処理開発協会先端情報技術研究所, 1999. http://www.icot.or.jp/AITEC/FGCS/funding/98/gaiyou19.html.

[26] Menezes, F., Barahona, P., and Codognet, P.: An Incremental Hierarchical Constraint Solver, *Proc. PPCP'93*(Saraswat and van Hentenryck(eds.)), MIT Press, 1994.

[27] Myers, B. A., Giuse, D. A., Dannenberg, R. B., Vander Zanden, B., Kosbie, D. S., Pervin, E., Mickish, A., and Marchal, P.: Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces, *IEEE Computer*, Vol. 23,No. 11(1990), pp. 71–85.

[28] Myers, B. A., Miller, R. C., McDaniel, R., and Ferrency, A.: Easily Adding Animations to Interfaces Using Constraints, *Proc. ACM UIST*, 1996, pp. 119–128.

[29] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T.: *NUMERICAL RECIPES in C: The Art of Scientific Computing*, Cambridge University Press, New York, 1988; 丹慶勝市, 奥村晴彦, 佐藤俊郎, 小林誠訳: *NUMERICAL RECIPES in C* [日本語版], 技術評論社, 東京, 1993.

[30] Sannella, M.: Constraint Satisfaction and Debugging for Interactive User Interfaces, Technical Report 94-09-10, Dept. of Computer Science and Engineering, University of Washington, 1994.

[31] Sannella, M.: SkyBlue: A Multi-Way Local Propagation Constraint Solver for User Interface Construction, *Proc. ACM UIST*, 1994, pp. 137–146.

[32] Sun Microsystems, Inc.: *Java Studio 1.0*, 1998.

[33] Suzuki, T., Kakinuma, N., and Tokuda, T.: An Experimental Comparison of Three Modified DeltaBlue Algorithms, *Principles and Practice of Constraint Programming—CP'96*(Freuder, E. C.(ed.)), Lecture Notes in Computer Science, Vol. 1118, Springer-Verlag, 1996, pp. 425–435.

[34] Takahashi, S., Miyashita, K., Matsuoka, S., and Yonezawa, A.: A Framework for Constructing Animations via Declarative Mapping Rules, *Proc. IEEE VL*, 1994, pp. 314–322.

[35] Vander Zanden, B.: An Incremental Algorithm for Satisfying Hierarchies of Multi-Way Dataflow Constraints, *ACM Trans. Prog. Lang. Syst.*, Vol. 18,No. 1(1996), pp. 30–72.

[36] Wilson, M.: Hierarchical Constraint Logic Programming (Ph.D. Dissertation), Technical Report 93-05-01, Dept. of Computer Science and Engineering, University of Washington, 1993.