チュートリアル　ユーザーインターフェースにおける制約解消法の研究動向

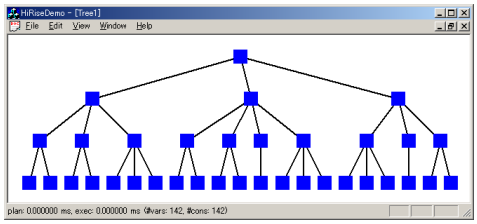
細部 博史

制約を宣言的に記述する手法は，多様な問題解決のための有力な手段であり，ユーザーインターフェース (UI) を含む，様々な分野で広く利用されている．この手法を実現する上で，制約解消法は必要不可欠な基本技術であり，これまで盛んに研究されてきた．本稿では，UI における制約解消法の研究動向について解説する．最初に，制約解消法を制約解消方式という観点から分類する．次に，代表的な制約解消方式である，基本方式，データフロー方式，制約階層を中心に，それぞれの形式的側面とアルゴリズム的側面について概説する．最後に，今後の研究課題について言及する．

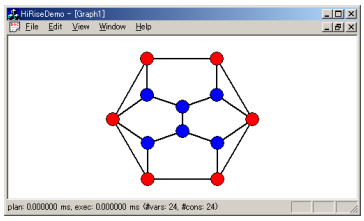
1　はじめに

制約を宣言的に記述する手法は，多様な問題解決のための有力な手段であり，人工知能や，論理型プログラミング，ユーザーインターフェース (UI) など様々な分野で広く利用されている．特に UI 分野において，制約の研究は，1960 年代前半の Sketchpad システム [55] を起源とし，1970 年代後半の ThingLab システム [4] [6] 以降，盛んに行われるようになった．

制約とは，成立すべき関係を宣言的に記述したものであり，数学的には，変数間の関係として表現されることが多い．一旦，宣言された制約は，システムによって自動的に管理されることになり，必要に応じて，制約が与えられた変数には適切な値が計算されて割り当てられる．



(a)



(b)

図 1　制約を用いた (a) 木構造と (b) グラフ構造のレイアウト

UI 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．例えば，図 1は，制約を用いて実現した木構造とグラフ構造のレイアウトである．ここで，木構造のレイアウトは，同じ親ノードを持つ部分木が隣接し，隣り合う葉ノードの間隔が一定になるように，1 次等式制約を用いて宣言的に定義されている．一方，グラフ構造は，内側のノードが，隣接するノードの重心に位置するように定義されている．

グラフィカルレイアウトにおける制約の利点は，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

Constraint Satisfaction in User Interfaces: Introduction and Survey.

Hiroshi Hosobe, 国立情報学研究所, National Institute of Informatics.

コンピュータソフトウェア, Vol.17, No.6(2000), pp.73–85.

[チュートリアル] 1999 年 12 月 30 日受付.

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．これは特に，レイアウトが複雑で，単純なループや再帰では記述しにくい場合に有効である．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．

制約解消は，制約を実現する上で必要不可欠な基本技術である．一般に，制約を用いたシステムの処理能力は，その制約解消の技術に大きく依存する．このため，制約解消は，制約の研究において特に重要視され，盛んに研究されてきた歴史を持つ (制約解消以外の研究課題については 8節で簡単に紹介する)．

本稿では，UI における制約解消法の研究動向について解説する．最初に，制約解消法を概観し，制約解消方式という観点から分類を行う．次に，代表的な制約解消方式である，基本方式，データフロー方式，制約階層を中心に，それぞれの形式的側面とアルゴリズム的側面について概説する．その際，アルゴリズム的側面については，従来研究を網羅的に紹介する．最後に，今後の研究課題について言及する．

2 制約解消法の概観

互いに作用し合う制約の集合を**制約系** (constraint system[†1]) という．ただし，それ以外にも，制約ネットワーク (constraint network) など様々な呼び方があり，特に人工知能分野では，制約充足問題 (constraint satisfaction problem) [60]と呼ばれることが多い．

制約系を解くことを制約解消といい，その方法を制約解消法という．しかし，単に制約解消法というと広い意味を持ち得るため，本稿では，以下の 2 つの言葉で区別することにする．

制約解消方式： 制約系の解を定義する理論的な枠組・体系を意味する．制約解消方式が異なれば，同じ制約系から異なる解が得られることもある．

制約解消アルゴリズム： 特定の制約解消方式に基づき，ある種類の制約からなる系の解を求める計算方法を指す．

†1 constraint system は「制約処理システム」を意味することもある．

異なる制約解消方式が必要となる理由の 1 つは，現実の問題でしばしば生じる，**制約過多** (over-constrained)[†2]の制約系に対処するためである．制約過多の系では，制約系の内部に相矛盾する制約が含まれている．このため，全ての制約の充足を解の条件とする通常の方式では，「解なし」の状態に陥ってしまう．しかし，そのような場合でも，例えば UI でレイアウトのために制約を用いるような状況では，何らかの妥協的な解が要求される．このため，制約系を可能な限り充足する問題緩和の能力を備えた方式が必要となる．

問題緩和を行わない通常の制約解消方式は，一般に，制約の追加に応じて，解集合が部分集合へ縮小するか，または維持されることから，**単調** (monotonic) な方式と呼ばれる．一方，問題緩和による妥協的な解を求める方式は，解集合が不連続に変化し，**非単調** (non-monotonic) な方式と呼ばれる．このような非単調な方式で扱われる制約は，**軟らかい** (soft) 制約といわれる[†3]．これは，問題の緩和を制約の緩和として見なすことができるためである．ただし，非単調方式でも，厳密に充足すべき**硬い** (hard) 制約を同時に提供することは多い．

UI システムで実際に使われている，代表的な制約解消方式としては，以下のものが挙げられる．

基本方式： 単調な方式の一般形であり，例えば，実数領域上の 1 次等式，1 次不等式，2 次等式などの算術制約を扱う．

データフロー方式： 単調方式の特殊形であり，データフロー制約に特化されている．

制約階層： 制約系を複数のレベル群に階層化し，制約の階層的な優先度を実現する非単調な方式である．

以下では，上記の方式を中心に，その形式的側面と，提案されている制約解消アルゴリズムについて紹介する．

†2 制約過多とは逆に，ある個数以上の解を持つ制約系を制約不足 (under-constrained) という．個数の基準には，2 つの場合もあれば，無限個の場合もある．

†3 ここで述べている意味での単調性と，軟らかい制約を同時に実現することは不可能ではないが，あまり現実的でない．ただし，より弱い意味での単調性を維持しながら，軟らかい制約を実現することは可能である [32].

3 基本方式

本稿で，基本方式とは，単調な制約解消方式の一般形を意味する．定式化の上では，変数が取り得る値や，制約の種類，制約系の構造に関して，あらかじめ制限を設定しない(実際の制約解消アルゴリズムがそのような制限を課すことは構わない)．

以下では，制約解消の基本方式について説明した後，その制約解消アルゴリズムについて述べる．

3.1 方　式

基本方式に代表される，単調な制約解消方式では，全ての制約を満たす必要がある．すなわち，変数の値の組合せの中で，全ての制約を充足するものが解となる．これは形式的に以下のように書ける．全ての変数を表す変数ベクトルを $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$，各変数の値を示す変数値ベクトルを $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$ とし(v_i は x_i の値を表す)，制約系を制約の集合 $C = \{c_1, c_2, \ldots, c_m\}$ とする．このとき，C の解集合 $S(C)$ は次のように定義される．

$$S(C) = \{\boldsymbol{v} \mid \forall i\ holds(c_i, \boldsymbol{v})\}$$

ただし，$holds(c_i, \boldsymbol{v})$ は，$\boldsymbol{v}$ のもとで c_i が成立することを意味する†4．

解の定義から容易に導かれるように，基本方式では，2つの制約系 C, C' に対して，次のような単調性が成立する．

$$S(C \cup C') \subseteq S(C)$$

すなわち，制約系に新たな制約を加えた場合，その解集合は不変であるか，部分集合へ縮小されるかのいずれかである．特に，相矛盾する制約が存在する場合には，空の解集合が得られ，制約系は「解なし」になる．

3.2 アルゴリズム

基本方式の制約系は，様々な種類のアルゴリズムで解かれる．例えば算術制約の場合，行列計算や反復法などの数値計算アルゴリズムや，線形計画法等の最適化アルゴリズム，Gröbner 基底や項書換え系によるアルゴリズムなどが利用される．また，扱う領域が有限領域の場合は，バックトラック法などの探索アルゴリズムを用いる制約解消法[60]が適用可能である．

UI を対象とした制約解消アルゴリズムについても様々な例がある．例えば，描画エディター Juno [52]とその後継 Juno-2 [28]では，非線形制約の系を解くために Newton 法を用いている．Oak システム[59]では，定規とコンパスで表現される初等幾何制約を扱い，1次近似を用いて制約解消の高速化を図っている．オブジェクト指向言語のための制約解消系 Equate [67]では，制約の記号的変換によって制約系を解いている．

4 データフロー方式

データフロー方式は単調な方式の特殊形であり，制約系の解は基本方式と同様に定義される．データフロー方式の特徴的な点は，データフロー制約†5を扱う点と，制約系の構造にある．

データフロー方式の最大の利点の1つは，多様な領域を扱えることである．この方式では，各データフロー制約の変数に対して入力または出力の役割を与え，メソッドやフォーミュラなどと呼ばれる関数や手続きを実行することで，入力変数の値に応じた出力変数の値を決定し，制約を充足する．このような関数等は，多くの場合，通常のプログラミング言語で記述され，制約解消系の外部から与えられるため，様々な値の利用が可能である．

以下では，データフロー方式について解説した後，その制約解消アルゴリズムについて述べる．

4.1 方　式

本項では，データフロー方式について，扱う制約の種類ごとに説明する．

4.1.1 単方向制約

データフロー制約の最も単純な形式は，**単方向** (one-way) 制約である．単方向制約は，その名の通り，「一方通行」の制約である．単方向制約では，参照する変数

†4 基本方式では，制約 c を関係として定式化することも多い．すなわち，各変数が取り得る値の領域を D としたとき，c を D^n の部分集合 ($c \subseteq D^n$) とし，$holds(c, \boldsymbol{v})$ の代わりに，$\boldsymbol{v} \in c$ と書く．

†5 制約解消問題に関する文献[27]では，データフロー制約と同様の性質を持つものを，関数 (functional) 制約として分類している．

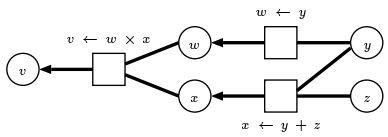


図 2　制約グラフ

のそれぞれに入力または出力の役割があらかじめ定められており，変数の依存関係が一定である．これはちょうど，表計算ソフトウェアでセル間に与えられた数式のようなものであると考えてよい．

以下は，単方向制約からなる，データフロー方式の制約系の例である．

$$v \leftarrow w \times x \quad (1)$$
$$w \leftarrow y \quad (2)$$
$$x \leftarrow y + z \quad (3)$$

各制約において，右辺に現れる変数が入力変数で，左辺に現れる変数が出力変数である．例えば，制約 (1) では，w と x が入力変数で，v が出力変数である．

データフロー制約の系は，制約グラフで図示すると理解しやすい．図 2は，先の制約系を制約グラフで表したものである．この制約グラフは 2 部グラフになっており，円形と正方形の 2 種類のノードがそれぞれ変数と制約を表し，エッジが制約から変数への参照を示し，エッジの矢印が制約の出力変数を指している．

一般に，データフロー制約の系は，各制約を適切な順序で充足することで解かれる．このために，制約系は以下の 2 つの条件を満たす必要がある．

- 衝突がないこと．複数の制約が同じ変数を出力変数として共有していない．
- 循環がないこと．制約の入出力関係に循環的な依存が存在しない．

単方向制約の系において，衝突は，変数の単位で局所的に検出できる．一方，循環は，大域的な解析が必要となるため，通常，制約解消の段階で動的に検出が行われる．

4.1.2　多方向制約

より高度なデータフロー制約に，**多方向** (multi-way) 制約がある．1 つの多方向制約は，複数のメソッドからなり，複数の出力候補変数を持つ．例えば，多方向制約 $x = y + z$ を 3 つのメソッド $x \leftarrow y + z$, $y \leftarrow x - z$, $z \leftarrow x - y$ から構成できる．

多方向制約の系の解消は，各制約に適切なメソッドを割り当てた後，各メソッドを順に実行することで行われる．適切なメソッドを決定するための基準は，衝突と循環を生じないことである．

単調な方式で多方向制約を扱う場合，予測不能性の問題が生じる [61]．これは，制約の与え方が十分でない場合に，各制約に対するメソッドの選び方で妥当なものが複数通りある制約不足の状況になり，可能な解が複数個存在し得るという問題である．この問題を避けるには，制約を必要十分なだけ指定すればよいが，現実には，常にこのように制約を管理することは，プログラマーにとって大きな負担となり得る．この問題の効果的な解決法として，5節で解説する制約階層がある．

4.2　アルゴリズム

本項では，データフロー方式のための制約解消アルゴリズムについて，単方向制約の場合と多方向制約の場合に分けて述べる．

4.2.1　単方向制約

データフロー方式の制約系は，通常，**局所伝播法** (local propagation) によって解消される．局所伝播法では，適切な順序で制約を 1 個ずつ解いていくことで，制約系の解を求める．例えば先の例では，各制約を (2), (3), (1) の順で解くようにすればよい．

実際に制約を解く順序を見つけるには，制約グラフを有向グラフとして見なした上で，トポロジカルソートを適用する．有向グラフのトポロジカルソートとは，各エッジが示すノードの順序関係に矛盾しないように，全ノードを直列に並べる処理である．トポロジカルソートは，深さ優先探索を用いて実現できる [41]．

トポロジカルソートによって，循環が検出されることがある．この場合，単方向制約の解消系では，単純に，循環に含まれる各制約を 1 度ずつ解くことで対症療法的に処置することが多い．

一部の単方向制約の解消系 [35] [51]は，遅延評価を採用している．遅延評価とは，関連する変数の値が実際に必要になるまで制約の計算を遅らせることで，不要な計算を除去する手法である．その有効性については，単方向制約の解消アルゴリズムに関する実験的分析 [63]に

よって支持されている．

なお，単方向制約の処理に関する研究では，通常の制約解消以外の課題を扱ったものが多い．例えば，上記の遅延評価や，ポインター変数の導入[62]，CSCW のサポート[29]などが挙げられる．このような研究が行われる理由は，単方向制約という制限された問題設定が，新しい課題の実現や評価に適しているためであると考えられるが，他の制約解消方式における，これらの課題の有効性も十分に推測され得る．

4.2.2 多方向制約

多方向制約の系の解消は，プランニングと実行の 2 段階からなる．プランニング段階では，制約系の構造に基づいて，各制約に対して適切なメソッドが選択される．一方，実行段階では，プランニング段階におけるメソッド選択の結果に応じて，単方向制約の場合と同様の局所伝播法が行われる．この際，すでにメソッドが選択された多方向制約の系は，単方向制約の系として処理することができる．

基本的なプランニングアルゴリズムとして，**既知状態伝播法** (propagation of known states) と**自由度伝播法** (propagation of degrees of freedom) が知られている[6]．両者は，メソッド選択が一意に決まる状況においては，同等の能力を持つ．一般には，既知状態伝播法は制約過多の状況に向き，自由度伝播法は制約不足の状況に適しているという特徴がある．

既知状態伝播法は，一意に値を決定できる変数を見つけるステップを繰り返すことでプランニングを進める．より具体的には，各ステップにおいて，全ての入力変数の値が既知 (入力変数のない場合を含む) で，出力変数の値が未知であるメソッドを持つ制約を見つけて，そのメソッドを選択し，その後のステップでは，その出力変数の値を既知であるものとして処理を行う．

図 3は，既知状態伝播法の実行例である．図 3(a) は，変数 v, w, x, y, z と，多方向制約 A, B, C, D, E からなる制約系を表し，各制約に添えられた矢印はそのメソッドを示す．この制約系は既知状態伝播法で以下のように解くことができる．最初に，図 3(b) のように，入力変数を持っていない制約 D のメソッドを選択する．これによって，制約 B のメソッドの一方の全入力変数 (y のみ) が既知になったので，図 3(c) のように，そのメソッドを選択する．次に，図 3(d) のように，制約 E のメソッドを選択する．ここで，制約 C のメソッドの 1 つの全入力変数 (y と z) が既知になったので，図 3(e) のように，それを選択する．最後に，図 3(f) のように，制約 A のメソッドの 1 つを選択する．

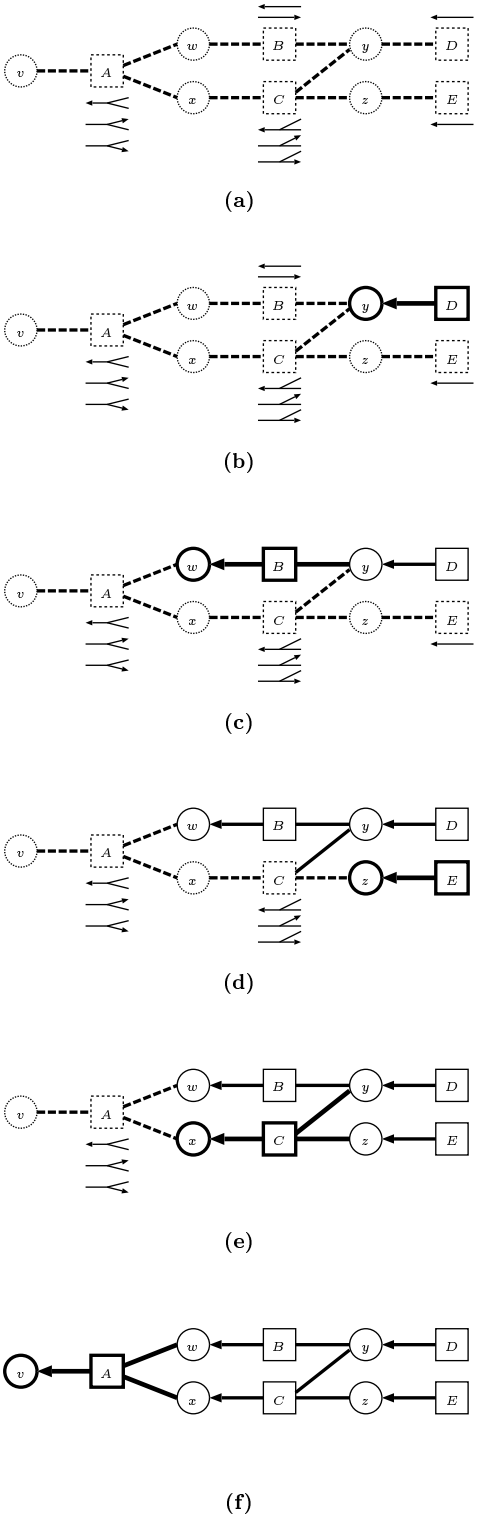
一方，自由度伝播法は，自由に値を決定できる変数を見つけるステップを繰り返すことでプランニングを進める．より具体的には，各ステップにおいて，出力変数の値を自由に (他の制約とは競合せずに) 決定できるメソッドを持つ制約を見つけて，そのメソッドを選択し，その後のステップでは，そのメソッドの入力変数の値の決定に，その制約は関わらないものとして処理を行う．

5 制約階層

制約階層 (constraint hierarchy) [7] [9] [68]は，硬い制約と軟らかい制約という概念を，**強さ** (strength) と呼ばれる，有限段階の優先度へ拡張したものである．いわゆる硬い制約が最も強く，**必須** (required) 制約と呼ばれ，それ以外の軟らかい制約は**選好** (preferential) 制約と呼ばれる．通常，必須制約の強さは required として表され，選好制約の強さは，強いものから順に strong, medium, weak として表される．

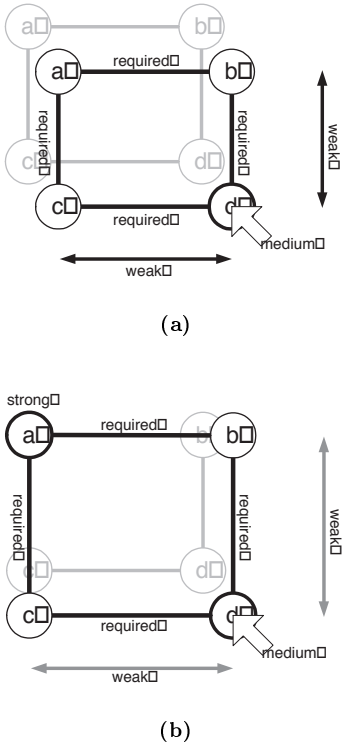
制約階層で制約の強さを適切に利用することで，UI の構築が容易になる．特に，強さは，グラフィカルオブジェクトの振舞いの表現に有効である．典型的には，デフォルトの振舞いを弱い制約で指定しておき，特定の振舞いが必要な状況で，より強い制約を課すという方法が採られる．例えば図 4(a) では，オブジェクト a, b, c, d が，4 つの強さ required の制約で長方形に配置され，その縦・横の間隔が 2 つの weak の制約で指定されている．この場合，weak の制約は，長方形の大きさの保持というデフォルトの振舞いを指定しており，medium の制約によって d がドラッグされると，長方形の大きさを保ったまま，他のオブジェクトも移動する．一方，図 4(b) のように，新たに strong の制約で a の位置を固定すると，weak の制約が無視され，d がドラッグされたときの振舞いは，長方形の大きさの変更になる．

レイアウト以外に UI で制約階層を利用する例として，グラフィカルオブジェクトの属性に制約を与える場合がある．例えば，テキストを表すグラフィカルオブ



(a) (b) (c) (d) (e) (f)

図 3　既知状態伝播法の実行例



(a) (b)

図 4　制約階層の利用例

ジェクトにおいて，そのデフォルトのフォントの大きさを制約で指定しておき，必要に応じて，より強い制約で大きさを変更するといった状況が考えられる．実際にこのように制約階層を用いることで，WWW ページの書式定義のための Cascading Style Sheets (CSS) の拡張に関する研究[2]では，複数の CSS (例えば，ブラウザーのデフォルトのもの，ユーザー個人用のもの，ページの著者によるもの) が与えられた場合に適切に対処できるようにしている．

以下では，制約階層の定式化の概要を説明した後，代表的な 3 種類の制約解消アルゴリズムと，その他のアルゴリズムを紹介する．

5.1　方　　式

ここでは，Borning らが提案した制約階層の定式化の概要を紹介する．これは基本的に[68]に従うが，簡単のためにその一部を修正している．

5.1.1　制約階層とその解

制約階層 H は，レベル 0 からレベル l までの $(l+1)$

個のレベルからなるベクトル

$$H = (H_0, H_1, \ldots, H_l)$$

であり，その各レベル H_k は，強さ k の制約を m_k 個含むベクトル

$$H_k = (c_{k,1}, c_{k,2}, \ldots, c_{k,m_k})$$

である．直観的に，強さは，$0, 1, \ldots, l$ の順で優先度が低くなり，強さ 0 の制約は必須制約で，それ以外は選好制約である．強さが記号的に required, strong, medium, weak と表現される場合，それぞれ強さ 0, 1, 2, 3 を表す．

制約階層の解は，*better* という比較子 (comparator) を用いて定義される．*better* は，制約階層 H に従って 2 つの変数値ベクトル $\boldsymbol{v}$, $\boldsymbol{v}'$ の解としての「良さ」を比べる述語であり，$better(\boldsymbol{v}, \boldsymbol{v}', H)$ が真であることは，H を充足する度合において $\boldsymbol{v}$ が $\boldsymbol{v}'$ よりも良いことを意味する．この比較の際，強い制約をより良く満たす変数値ベクトルを高く評価するように，比較子は定義される．

制約階層 H の解としては，必須制約 H_0 を満たす変数値ベクトルの中から，比較子 *better* の意味で，より良いものが存在しないベクトルが選ばれる．形式的には，H の解集合 $S(H)$ は次のように定められる．

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

ただし，$S_0(H)$ は，全ての必須制約 H_0 を満たす変数値ベクトルの集合である．

$$S_0(H) \equiv \{\boldsymbol{v} \mid \forall i\, holds(c_{0,i}, \boldsymbol{v})\}$$

上の解の定義では，「最も良い」変数値ベクトルではなく，「より良いものが存在しない」変数値ベクトルを選ぶようにしている．このような定義の理由は，比較子の種類によっては，変数値ベクトルを比較できない ($better(\boldsymbol{v}, \boldsymbol{v}', H)$ でも $better(\boldsymbol{v}', \boldsymbol{v}, H)$ でもない) 場合があるために，最も良い変数値ベクトルが存在しないことがあるからである．

5.1.2 比較子

同一の制約階層でも，比較子 *better* の具体的な定義に応じて，その解集合は異なってくる．比較子にはいくつかの具体例が提案されており，ほとんどの比較子は，その定義の方法に応じて，**大域的** (global) 比較子または**局所的** (local) 比較子のいずれかに分類される [9] [68].

大域的比較子の具体例として，least-squares-better (LSB) がある．この比較子は，レベル内の矛盾する制約に対して最小 2 乗法を行う．具体的には，各レベルで制約の誤差の 2 乗の総和を計算し，より強いレベルにおいて総和が小さいほど，変数値ベクトルを良く評価する．形式的には，次のように定義される．

$$\begin{aligned} &least\text{-}squares\text{-}better(\boldsymbol{v}, \boldsymbol{v}', H) \\ &\equiv \exists k \forall k' \,(k' < k \\ &\qquad \Rightarrow \textstyle\sum_i (e(c_{k',i}, \boldsymbol{v}))^2 = \sum_i (e(c_{k',i}, \boldsymbol{v}'))^2) \\ &\qquad \wedge \textstyle\sum_i (e(c_{k,i}, \boldsymbol{v}))^2 < \sum_i (e(c_{k,i}, \boldsymbol{v}'))^2 \end{aligned}$$

ただし，$e(c, \boldsymbol{v})$ は，変数値ベクトル $\boldsymbol{v}$ のもとでの制約 c の誤差を非負実数 (c が充足されるときには 0) として返す，距離的誤差関数 (metric error function) である．これは，通常の算術制約については，等式ならば，右辺と左辺の差の絶対値を取ることで定められる．

大域的比較子には，LSB 以外にも，レベル内の制約の距離的誤差関数の結果の和を用いる weighted-sum-better (WSB) や，レベル内の制約で最大の距離的誤差関数の結果を用いる worst-case-better (WCB) などがある．

局所的比較子は，レベル内において任意の順序で 1 個ずつ制約の誤差を最小化していったときに，解が得られるように定義されている．局所的比較子には，locally-error-better (LEB) と locally-predicate-better (LPB) がある．LEB は，locally-metric-better とも呼ばれ，距離的誤差関数を用いる．一方，LPB は誤差関数として，制約が正しく充足される場合に 0 を，それ以外の場合は 1 を返す†6，述語的誤差関数 (predicate error function) を用いる．実用上，LEB は不等式制約を含む系に，LPB は等式制約のみの系に適しているとされる [5].

大域的比較子と局所的比較子の大きな相異点として，常に変数値ベクトルを比較可能かどうかという点がある．大域的比較子では，各レベルにおける制約の誤差が，常に比較可能な単一の非負実数に帰着されるため，制約階層全体においても変数値ベクトルは常に比較可能

†6 (アルゴリズムにもよるが) 実装上は，計算誤差を考慮して，制約が正しく充足されているかどうか判断する必要がある．

となる．一方，局所的比較子では，レベル内における制約の誤差の最小化の順序の違いによって，変数値ベクトルを比較できない場合が生じることがある．このような性質から，一般に，大域的比較子よりも局所的比較子の方が，解の条件が緩いために解の個数が多くなり，高速な制約解消アルゴリズムを設計しやすくなる傾向がある．

以下に，具体的な比較子を用いて制約階層を解消する例として，次の制約階層 H を LSB で解く場合を説明する．

$$\begin{array}{ll} \mathsf{required} & x_1 = x_2 \\ \mathsf{strong} & x_2 + 1 = x_3 \\ \mathsf{weak} & x_1 = 0 \\ \mathsf{weak} & x_3 = 3 \end{array}$$

最初に，required レベルの制約を満たすよう，$S_0(H)$ が以下のように決定される．

$$S_0(H) = \{(v_1, v_2, v_3) \mid v_1 = v_2\}$$

次に，$S_0(H)$ の中で，より良い変数値ベクトルが存在しないものの集合として，解集合 $S(H)$ が決定される．変数値ベクトルの比較は，LSB 等の大域的比較子では，各レベルの誤差を表す非負実数を，強いレベルから順に比べることで可能である．比較の例を示すため，表 1に，$S_0(H)$ の要素の内，$(0,0,1)$, $(1,1,2)$, $(2,2,3)$, $(0,0,3)$ に対応するレベルの誤差を列挙する (例えば，$(0,0,1)$ に対する weak レベルの誤差は，$(v_1-0)^2 + (v_3-3)^2 = 4$ として計算される)．まず，$(0,0,3)$ など，strong レベルの誤差が 0 でない変数値ベクトルは，他により良いものが存在するため，解になり得ない．また，strong レベルの誤差が 0 になる変数値ベクトルの中でも，$(0,0,1)$ や $(2,2,3)$ などは，$(1,1,2)$ より良くないため，解ではない．一方，$S_0(H)$ のいかなる要素も $(1,1,2)$ より良くはない (実際に，$v_1 = v_2$ かつ $v_2 + 1 = v_3$ という条件のもとで，$v_1^2 + (v_3-3)^2$ を最小化することで確認できる) ことから，最終的に，唯一の解 $(1,1,2)$ からなる解集合 $S(H)$ が求められる．

$$S(H) = \{(1,1,2)\}$$

表 1　変数値ベクトルに対するレベルの誤差

(v_1, v_2, v_3)	レベルの誤差	
	strong	weak
(0, 0, 1)	0	4
(1, 1, 2)	0	2
(2, 2, 3)	0	4
(0, 0, 3)	1	0

5.2　アルゴリズム

以下では，制約階層のための制約解消アルゴリズムを，精製法，局所伝播法，最適化アプローチ，その他の 4 つに分類し，解説する．

5.2.1　精製法

精製法は，各レベルを強い順に充足することで，制約階層を解消する．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 [7]では，各レベルごとに，制約を 1 次等式で近似し，制約系の最小 2 乗解を山登り法で求める緩和法 (relaxation) を適用していた．Orange アルゴリズム [14][17]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 WSB または WCB で充足している．DeltaStar [14][68]は，Orange を一般化したアルゴリズムで，大域的比較子だけでなく局所的比較子にも対応している．

5.2.2　局所伝播法

局所伝播法を採用した制約解消法は，通常，強さを与えられた多方向制約の系を局所的比較子で解く．単調なデータフロー方式の場合と同様，制約解消は，プランニングと実行の 2 段階からなる．プランニング段階では，制約の強さを考慮し，制約階層の解の定義に従うように，充足すべき制約のメソッドを選択する．一方，実行段階では，単方向制約の場合と同様の局所伝播法を実行する．

Blue [45]は，制約階層のための最初の局所伝播法アルゴリズムである．Blue は，既知状態伝播法を実行しながら，各ステップで選択すべきメソッドを検索する際に，制約を強い順に調べるようにしている．この方法で，循環を必要としない場合には，LPB 解を得ることが可能である．

DeltaBlue [17][45]は，プランニングをインクリメン

タルに行うことで，LPB 解を効率的に求めるアルゴリズムである．具体的には，新規の制約が追加された場合，または既存の制約が削除された場合に，現在のメソッド選択を部分的に修正することで，新しいメソッド選択を得るようにしている．その効率化の鍵となっているのは，walkabout strength と呼ばれる手法である．これによって，制約追加の際に，その制約を充足すべきかどうか素早く判断した上で，充足すべき場合には，それまで充足されていた制約の中からメソッド選択を解除すべきものを高速に発見することを可能にしている．

DeltaBlue 以降，様々なインクリメンタルアルゴリズムが提案されている[12][56][57]．SkyBlue [53][54]は，DeltaBlue をより一般化し，制約の循環的関係と，多出力のメソッドを持つ制約に対応したものである．QuickPlan [61]は，自由度伝播法を基礎としたアルゴリズムで，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．*DETAIL* [32][34]は，局所伝播法の枠組を拡張して，LSB などの比較子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．

通常の局所伝播法では等式制約を扱うが，Indigo [5]では，区間演算を利用することで，循環的関係のない，不等式制約を含む制約階層を LEB で解くことができる．

5.2.3 最適化アプローチ

最適化アプローチは，強さを重みに対応させ，制約階層を，線形計画法などの 1 つの最適化問題に変換して解く．例えば，5.1.2節で与えた制約階層は，次のような最適化問題に変換される．

$$\begin{aligned} \text{minimize} \quad & w_{\text{strong}} f(\varepsilon_1) + w_{\text{weak}} f(\varepsilon_2) \\ & + w_{\text{weak}} f(\varepsilon_3) \\ \text{subject to} \quad & x_1 = x_2 \\ & x_2 + 1 = x_3 + \varepsilon_1 \\ & x_1 = 0 + \varepsilon_2 \\ & x_3 = 3 + \varepsilon_3 \end{aligned}$$

ただし，w_{strong}，w_{weak} は，それぞれ強さ strong，weak に対応する重みである．重みは，対応する強さが強いほど，重くなるように決められる (制約階層に従うよう実現されることもあれば，近似的に実現されることもある)．また，関数 f は，比較子に応じて，$f(\varepsilon) = |\varepsilon|$ または $f(\varepsilon) = \varepsilon^2$ のように定められる．

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応用して LEB 解を求める Cassowary [3][11]と，アクティブセット法を用いて近似的な LSB 解を求める QOCA [11][46]がある．

5.2.4 その他の制約解消アルゴリズム

上述の 3 種類に分類されない制約階層の解消アルゴリズムとして，以下のものが挙げられる．UltraViolet [8][10]は，制約階層を部分グラフに分割して，部分グラフごとに，Blue や Indigo 等の特定用途の解消系を呼び出すことで LEB 解を求める「メタ」な制約解消系である．projection を用いるアルゴリズム[24]は，1 次等式と 1 次不等式からなる静的な制約階層を LEB で解き，その計算方法をプログラムコードとして出力する．HiRise [31][33]は，数千個の 1 次等式からなる制約階層の LPB 解を実時間で計算できる．

6 その他の制約解消方式

制約階層が提案される以前のシステムでは，非単調な制約解消方式として最小 2 乗法を使っているものが多い．例えば，Sketchpad [55]と ThingLab [6]では，局所伝播法が適用不能な場合に，緩和法を実行する．また，TRIP システム[43]の制約解消系 COOL では，1 次等式制約を硬い制約と軟らかい制約に分類した上で，一部の変数を硬い制約で消去して得られる 1 次等式系 $A\boldsymbol{x} = \boldsymbol{c}$ の最小 2 乗解 $\boldsymbol{v}$ を，$A^T A\boldsymbol{v} = A^T\boldsymbol{c}$ (A^T は A の転置行列) を解くことで計算する†7．

differential approach [19]は，単純化された物理シミュレーションを行うことで，様々な非線形制約を実現する数学的枠組であり，軟らかい制約を導入するための仕組も提供している．このアプローチでは，通常の直接操作のように絶対座標でオブジェクトの位置を指定することができないため，オブジェクトを目的地に向けて徐々に移動する，differential manipulation [20]と呼ばれる入力方式が採用されている．また，制約解消の際に導関数を扱う必要があるため，これを簡明に実現

†7 COOL は，1 次等式制約に加えて，無向グラフの配置のための制約を扱うことができる．このために，スプリングモデルに基づくグラフ配置を Newton 法で計算するアルゴリズム[42]を統合している．

する手法として snap-together mathematics [21]が提案されている．このアプローチに基づき，描画エディター Briar [22]や，2 次元 /3 次元グラフィックスツールキット Bramble [18]が実装されている．

実行可能制約[25] [26]は，マクロと制約を統合する非単調方式である．個々の実行可能制約は，通常の制約に類似した条件部と，マクロのように値の計算を行う手続き列からなり，半順序の優先度が与えられている．この枠組において，制約系の解は，数値的アルゴリズム等を用いずに，制約の手続き列を実行することで計算され，矛盾を生じる場合には，優先度の高い制約が選ばれるようになっている．実行可能制約は，例示プログラミングとの親和性があり，エンドユーザーが実行可能制約を指定できる描画エディターが実現されている．

7 制約解消に関する今後の研究課題

制約解消法に関する研究課題は，通常，表現力の強化と，効率の向上の 2 点に集約される．本節では，UI への応用という観点から，それぞれの点について今後の研究課題を簡潔に説明する．

表現力の強化という課題は，さらに以下の問題に分けて考えることができる．

- 制約解消方式の表現力の強化．制約階層の拡張[32]など，制約解消の枠組の強化を行う．これは必ずしも数学的に厳密なものであるとは限らず，例えば，遺伝的アルゴリズムを用いた，より曖昧な枠組[47]なども考えられる．
- 制約の表現力の強化．これまで UI で扱われてきた制約の多くは，2 次元図形に関する 1 次または 2 次の制約である．しかし，仮想現実感などの 3 次元インターフェースを視野に入れれば，回転など，より複雑な関係も必要となってくる．また，アプリケーションによっては，複数の制約の選言 (論理和) や，ある条件が成立する場合にのみ有効化すべき制約など，複合的な制約が要求される可能性もある．

効率の向上という課題に対しては，以下のようなアプローチが考えられる．

- 制約解消アルゴリズムの高速化．制約解消に関する多くの従来研究と同様，インクリメンタル性の実現などによって，アルゴリズムの時間計算量を削減する．
- 制約解消に要する記憶領域の削減．これまでの制約解消系の多くは実行時に大量の記憶領域を必要とするため，特に実装技術的観点から記憶領域を削減する必要がある[38] [64] [65].
- 静的な制約解消手法．多くの制約解消アルゴリズムのように制約系を動的に解くのではなく，制約系を静的に解いてコンパイルすることで実行速度を向上する[15] [24].

一般に，表現力と効率はトレードオフの関係にあり，両者を同時に改善することは困難である．これらの適切なバランスを取ることは，新たな制約解消法を模索していく上での重要な基準となる．

8 制約解消以外の研究課題

本稿は制約解消に焦点を合わせたが，それ以外にも，制約に関する様々な研究課題がある．本稿を締め括る前に，その一部を簡単に紹介する．

制約の記述・指定： 制約解消系はライブラリーとして実装されているものが多いが，この場合，制約の記述や指定が煩雑になる．この問題に対処するため，プログラミング言語への制約の統合[16] [30] [51]や，制約のグラフィカルな指定[6] [36] [40]などを実現する．

制約の抽出・推論： プログラマーやユーザーが制約を直接的に指定するのではなく，システムが図形などの入力を分析することで，適切な制約を自動抽出したり推論したりする．これは，描画エディター[1] [13] [23] [39] [44] [58]や，例示プログラミング[48] [49] [50] [66]で要求される．

制約のデバッグ： 制約を用いたプログラムでもバグの発生は避けられない．そこで，システムによるデバッグ支援として，制約系の解析[53]や視覚化[37]等の機能を提供する．

9 おわりに

本稿では，UI における制約解消法の研究動向について解説した．その際，制約解消法を制約解消方式という観点で分類し，その形式的側面とアルゴリズム的側面について概説した．また，制約の研究に関する今後の課題

についても言及した．

以下に，本稿の読者が，制約を利用したUIシステムに関する理解をさらに深められるよう，有用なWWWページを紹介する．

- A. Borning らのページ．
  `http://www.cs.washington.edu/research/constraints/`
  制約階層や制約プログラミング言語などに関する様々な論文や，Cassowary 制約解消系等のソフトウェアを入手できる．
- B. A. Myers のページ．
  `http://www.cs.cmu.edu/~bam/`
  単方向制約を採用したC++向けUIツールキットAmuletと，Common Lisp向けツールキットGarnetが配布されている．
- S. Hudson のページ．
  `http://www.cs.cmu.edu/~hudson/`
  単方向制約を用いたJava向けのツールキットsubArcticを入手できる．

UI分野において，制約に関する研究は長い歴史を持つが，現状では，実用的なシステムで利用されている例はまだ少ないと言わざるを得ない．その一因として，従来の研究が，UIを対象としたものであっても，理論に偏る傾向があったことは否めない．今後は，より実用的な視点から要件を分析し，実証的な研究開発を進めることが必要となってくることだろう．このような視点に立つことができれば，本来は多様な問題解決のための有力手段である制約が，将来の知的インターフェースの実現に向けて，その重要性をさらに拡大していくことができると期待される．

参考文献

[1] 馬場昭宏, 田中二郎: Spatial Parser Generatorを持ったビジュアルシステム, 情処学論, Vol. 39, No. 5 (1998), pp. 1385–1394.

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

[3] 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.

[4] Borning, A.: The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory, *ACM Trans. Prog. Lang. Syst.*, Vol. 3, No. 4 (1981), pp. 353–387.

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

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

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

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

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

[10] Borning, A. and Freeman-Benson, B. N.: The OTI Constraint Solver: A Constraint Library for Constructing Interactive Graphical User Interfaces, *Principles and Practice of Constraint Programming—CP'95* (Montanari, U. and Rossi, F. (eds.)), LNCS, Vol. 976, Springer, 1995, pp. 624–628.

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

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

[13] Chok, S. S. and Marriott, K.: Automatic Construction of Intelligent Diagram Editors, *Proc. ACM UIST*, 1998, pp. 185–194.

[14] Freeman-Benson, B., Wilson, M., and Borning, A.: DeltaStar: A General Algorithm for Incremental Satisfaction of Constraint Hierarchies, *Proc. IEEE IPCCC*, 1992, pp. 561–568.

[15] Freeman-Benson, B. N.: A Module Mechanism for Constraints in Smalltalk, *Proc. ACM OOPSLA*, 1989, pp. 389–396.

[16] Freeman-Benson, B. N. and Borning, A.: Integrating Constraints with an Object-Oriented Language, *Proc. ECOOP*, LNCS, Vol. 615, Springer, 1992, pp. 268–286.

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

[18] Gleicher, M.: A Graphical Toolkit Based on Differential Constraints, *Proc. ACM UIST*, 1993, pp. 109–120.

[19] Gleicher, M.: A Differential Approach to Graphical Interaction, Technical Report CMU-CS-94-217,

School of Computer Science, Carnegie Mellon University, 1994.
[20] Gleicher, M. and Witkin, A.: Differential Manipulation, *Proc. Graphics Interface*, 1991, pp. 61–67.
[21] Gleicher, M. and Witkin, A.: Supporting Numerical Computations in Interactive Contexts, *Proc. Graphics Interface*, 1993, pp. 138–145.
[22] Gleicher, M. and Witkin, A.: Drawing with Constraints, *The Visual Computer*, Vol. 11, No. 1 (1994), pp. 39–51.
[23] Gross, M. D. and Do, E. Y.-L.: Ambiguous Intentions: a Paper-Like Interface for Creative Design, *Proc. ACM UIST*, 1996, pp. 183–192.
[24] Harvey, W., Stuckey, P., and Borning, A.: Compiling Constraint Solving using Projection, *Principles and Practice of Constraint Programming—CP97* (Smolka, G. (ed.)), LNCS, Vol. 1330, Springer, 1997, pp. 491–505.
[25] 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)), レクチャーノート / ソフトウェア学, Vol. 16, 近代科学社, 1996, pp. 41–49.
[26] Hattori, T.: Programming Constraint System by Demonstration, *Proc. IUI*, ACM, 1999, pp. 202.
[27] Hentenryck, P. V., Deville, Y., and Teng, C.-M.: A Generic Arc-Consistency Algorithm and its Specializations, *Artificial Intelligence*, Vol. 57 (1992), pp. 291–321.
[28] Heydon, A. and Nelson, G.: The Juno-2 Constraint-Based Drawing Editor, Research Report 131a, DEC Systems Research Center, 1994.
[29] Hill, R. D.: The Rendezvous Constraint Maintenance System, *Proc. ACM UIST*, 1993, pp. 225–234.
[30] Horn, B.: Constraint Patterns as a Basis for Object Oriented Programming, *Proc. ACM OOPSLA*, 1992, pp. 218–233.
[31] 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 (2000), pp. 25–29.
[32] Hosobe, H., Matsuoka, S., and Yonezawa, A.: Generalized Local Propagation: A Framework for Solving Constraint Hierarchies, *Principles and Practice of Constraint Programming—CP96* (Freuder, E. C. (ed.)), LNCS, Vol. 1118, Springer, 1996, pp. 237–251.
[33] 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.
[34] 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.)), LNCS, Vol. 874, Springer, 1994, pp. 51–62.
[35] Hudson, S. E.: Incremental Attribute Evaluation: A Flexible Algorithm for Lazy Update, *ACM Trans. Prog. Lang. Syst.*, Vol. 13, No. 3 (1991), pp. 315–341.
[36] Hudson, S. E. and P.Mohamed, S.: Interactive Specification of Flexible User Interface Displays, *ACM Trans. Inf. Syst.*, Vol. 8, No. 3 (1990), pp. 269–288.
[37] Hudson, S. E., Rodenstein, R., and Smith, I.: Debugging Lenses: A New Class of Transparent Tools for User Interface Debugging, *Proc. ACM UIST*, 1997, pp. 179–187.
[38] Hudson, S. E. and Smith, I.: Ultra-Lightweight Constraints, *Proc. ACM UIST*, 1996, pp. 147–155.
[39] Igarashi, T., Matsuoka, S., Kawachiya, S., and Tanaka, H.: Interactive Beautification: A Technique for Rapid Geometric Design, *Proc. ACM UIST*, 1997, pp. 105–114.
[40] Ingalls, D., Wallace, S., Chow, Y.-Y., Ludolph, F., and Doyle, K.: Fabrik A Visual Programming Environment, *Proc. ACM OOPSLA*, 1988, pp. 176–190.
[41] 石畑 清: アルゴリズムとデータ構造, 岩波講座ソフトウェア科学, Vol. 3, 岩波書店, 東京, 1989.
[42] Kamada, T. and Kawai, S.: An Algorithm for Drawing General Undirected Graphs, *Information Processing Letters*, Vol. 31, No. 1 (1989), pp. 7–15.
[43] Kamada, T. and Kawai, S.: A General Framework for Visualizing Abstract Objects and Relations, *ACM Trans. Gr.*, Vol. 10, No. 1 (1991), pp. 1–39.
[44] Kurlander, D. and Feiner, S.: Inferring Constraints from Multiple Snapshots, *ACM Trans. Gr.*, Vol. 12, No. 4 (1993), pp. 277–304.
[45] 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.
[46] Marriott, K., Chok, S. S., and Finlay, A.: A Tableau Based Constraint Solving Toolkit for Interactive Graphical Applications, *Principles and Practice of Constraint Programming—CP98* (Maher, M. J. and Puget, J.-F. (eds.)), LNCS, Vol. 1520, Springer, 1998, pp. 340–354.
[47] Masui, T.: Graphic Object Layout with Interactive Genetic Algorithms, *Proc. IEEE VL*, 1992, pp. 74–80.
[48] Miyashita, K., Matsuoka, S., Takahashi, S., and Yonezawa, A.: Interactive Generation of Graphical User Interfaces by Multiple Visual Examples, *Proc. ACM UIST*, 1994, pp. 85–94.
[49] Miyashita, K., Matsuoka, S., Takahashi, S., Yonezawa, A., and Kamada, T.: Declarative Programming of Graphical Interfaces by Visual Examples, *Proc. ACM UIST*, 1992, pp. 107–116.
[50] Myers, B. A.: *Creating User Interfaces by Demonstration*, Academic Press, San Diego, 1988.
[51] 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.
[52] Nelson, G.: Juno, a Constraint-based Graphics System, *Proc. ACM SIGGRAPH*, 1985, pp. 235–243.
[53] 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.
[54] Sannella, M.: SkyBlue: A Multi-Way Local Propagation Constraint Solver for User Interface Construction, *Proc. ACM UIST*, 1994, pp. 137–146.
[55] Sutherland, I. E.: Sketchpad: A Man-Machine Graphical Communication System, *Proc. AFIPS Spring Joint Conf.*, 1963, pp. 329–346.
[56] Suzuki, T., Kakinuma, N., and Tokuda, T.: An Experimental Comparison of Three Modified DeltaBlue Algorithms, *Principles and Practice of Constraint Programming—CP96* (Freuder, E. C. (ed.)), LNCS, Vol. 1118, Springer, 1996, pp. 425–435.
[57] Suzuki, T. and Tokuda, T.: An Incremental and Hierarchical Constraint Solver with the Lazy Planning Phase for User Interface Construction, *Proc. Intl. Conf. Software Engineering and Knowledge Engineering*, Knowledge Systems Institute, 1998, pp. 339–342.
[58] Takahashi, S., Matsuoka, S., Yonezawa, A., and Kamada, T.: A General Framework for Bi-Directional Translation between Abstract and Pictorial Data, *Proc. ACM UIST*, 1991, pp. 165–174.
[59] Tonouchi, T., Nakayama, K., Matsuoka, S., and Kawai, S.: Creating Visual Objects by Direct Manipulation, *Proc. IEEE VL*, 1992, pp. 95–101.
[60] Tsang, E.: *Foundations of Constraint Satisfaction*, Academic Press, London, 1993.
[61] 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.
[62] Vander Zanden, B., Myers, G. A., Giuse, D., and Szekely, P.: The Importance of Pointer Variables in Constraint Models, *Proc. ACM UIST*, 1991, pp. 155–164.
[63] Vander Zanden, B. T. and Venckus, S. A.: An Empirical Study of Constraint Usage in Graphical Applications, *Proc. ACM UIST*, 1996.
[64] Vander Zanden, B. T.: Optimizing Toolkit-Generated Graphical Interfaces, *Proc. ACM UIST*, 1994, pp. 157–166.
[65] Vander Zanden, B. T. and Halterman, R. L.: Reducing the Storage Requirements of Constraint Dataflow Graphs, *Proc. ACM UIST*, 1999.
[66] Vander Zanden, B. T. and Myers, B.: Demonstrational and Constraint-Based Techniques for Pictorially Specifying Application Objects and Behaviors, *ACM Trans. Comput. Human Interaction*, Vol. 2, No. 4 (1995), pp. 308–356.
[67] Wilk, M. R.: Equate: An Object-Oriented Constraint Solver, *Proc. ACM OOPSLA*, 1991, pp. 286–298.
[68] Wilson, M.: Hierarchical Constraint Logic Programming (Ph.D. Dissertation), Technical Report 93-05-01, Dept. of Computer Science and Engineering, University of Washington, 1993.