対話型インタフェースのための幾何制約解消の枠組み

細部博史†

ユーザインタフェース（UI）分野において，制約は古くから重要な役割を演じてきた．UI 分野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群の幾何的な配置であり，これによって UI の構築が容易化される．しかしながら，現在，制約の技術を導入したシステムやアプリケーションは少ないといわざるをえない．主要な障害の 1 つとして，連立された非線形な幾何制約を解くことができる，信頼性と利便性の十分に高い制約解消系がほとんど存在しない点があげられる．このような問題に対応するために，本論文では，グラフ配置など，表現力の高い幾何制約を扱う新しい制約解消の枠組みを提案する．本枠組みは，制約解消法とその実装方式の 2 つの部分からなる．制約解消法は，数値的な最適化手法と遺伝的アルゴリズムを組み合わせたものである．一方，実装方式は，オブジェクト指向プログラミングにより，制約解消系においてモジュール機構を実現する点を特徴としており，これによって新たな種類の制約を導入したり，数値最適化手法を交換したりすることが可能である．本枠組みはすでに Chorus 制約解消系として実装されており，本論文ではその性能に関する実験結果も与える．

A Geometric Constraint Solving Framework for Interactive Interfaces

Hiroshi Hosobe†

Constraints have been playing an important role in the user interface field since its infancy. The primary usage of constraints in this field is to obtain geometric layouts of graphical objects, which facilitates the construction of user interfaces. However, most researchers and developers do not incorporate constraint technology into their systems and applications. A major obstacle is that there are few sufficiently robust and usable constraint solvers capable of solving simultaneous nonlinear geometric constraints. To tackle this problem, this paper proposes a novel constraint solving framework, which handles expressive geometric constraints, e.g., for graph layout. It consists of the methods of constraint satisfaction and its implementation. The constraint satisfaction method is the combination of a numerical optimization technique with a genetic algorithm. The characteristic of the implementation method is to realize the module mechanism of resulting constraint solvers by object-oriented programming, which allows users to introduce new kinds of constraints and also to replace its numerical technique with another. This framework has been implemented as the Chorus constraint solver. This paper also provides the results of the experiments on its performance.

1. はじめに

ユーザインタフェース（UI）分野において，オブジェクト間の関係を**制約**として宣言的に記述する手法は，古くから重要な役割を演じてきた[30]．制約は，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．UI 分野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群の幾何的な配置である．オブジェクト間の幾何的関係が制約としていったん宣言されると，その後は，**制約解消系**と呼ばれるシステムソフトウェアによって，その関係が自動的に管理されることになり，必要に応じてオブジェクトの位置や向きなどが適切に求められ設定される．このような機構は，オブジェクト配置の計算が，単純なループや再帰などでプログラムするのが困難な場合には特に効果的である．

制約を利用した UI アプリケーションの構築を容易にするため，過去に，様々な制約解消法が提案され，制約解消系が開発されている．特に，変数の入出力関係に沿って計算が進められるデータフロー制約を対象として，制約解消系の研究がさかんに行われた[6),28),32)]．このようなデータフロー制約解消系には，様々な種類の制約を効率的に扱えるという利点がある．しかし，

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

その一方で，制約が連立されている状況や，不等式制約を扱うことが難しいという問題があるため，この数年間は，線形（1次）の等式および不等式制約を数値的に処理する解消系が提案されている[3),14),23)]．

これらの制約解消系の多くは，**制約階層**[2)]と呼ばれる，制約に優先度を与える枠組みに基づいている[3),6),14),23),28),32)]．制約階層では，制約の優先度は有限個の**強さ**で表され，制約階層の解は，できるだけ強い制約が満たされるように決定される．強さを用いることで，可能な場合にのみ充足されるデフォルトの制約を表現でき，プログラマはオブジェクトの配置方法を容易に制御できるようになる．

UIのための制約解消系に関する研究は多数行われてきたものの，最近の線形制約解消系で十分な域に達したとはいえない．たとえば，それ以前のデータフロー制約解消系で扱うことができた，$x \times y = z$などの**非線形制約**は，最近の線形解消系の範囲外である．また，線形解消系は，平行，垂直，距離の一定などの幾何制約を扱うこともできない．さらには，一般の**グラフ配置**を扱うこともきわめて困難である．

実際に，様々な状況で，非線形幾何制約をUIで利用する可能性が考えられる．最も基本的なアプリケーションは，描画エディタである[11),17),25)]．たとえば，その応用である教育用電子白板システムでは，三角定規とコンパスで描けるような図を扱えることが理想的であるが，その実現には非線形なEuclid幾何制約が必要である．また，情報検索や，データベース，プログラミングなどを対象とした情報視覚化システムにおいて，一般のグラフ配置を用いて情報の構造を表現したい場合がある[18),20),24)]．あるいは，仮想現実感やビデオゲームなどを目的とした3次元空間の構築でも，物体の動的な性質を表現するために，非線形幾何制約が有効である[8)]．さらに今後，コンピュータの高速化やグラフィックス機能の高度化にともない，このような処理への需要が増していくことも十分に予期される．

このような要求に応えるために，本論文では，UIを対象とした非線形幾何制約解消のための枠組みを提案する．本枠組みは高い表現力を達成しており，Euclid幾何制約だけでなく，グラフ配置制約にも対応している．また，制約階層など，従来の制約解消系の研究成果を取り入れており，UI開発に適している．

本枠組みは，制約解消法とその実装方式の2つの部分からなる．本枠組みにおける制約解消法は，数値的な非線形の**最適化**を基礎としている．このような手法を採用する場合の問題点として，最適化計算が局所解に陥りやすいことがあげられる．本枠組みではこの問題に対処するために，**遺伝的アルゴリズム**による大域的探索を統合している．

本枠組みにおける実装方式は，オブジェクト指向プログラミングにより，制約解消系において**モジュール機構**を実現する点を特徴としている．制約の解釈，あるいは意味づけは，拡張可能な評価モジュールによって行われる．この方式を用いることで，数式として表現される算術制約を新たに定義したり，グラフ配置などのより高度なアルゴリズムを実装して制約解消系から利用したりすることが可能である．これにより，新しい視覚化手法の実験のためのテストベッドとして，本枠組みを利用することなども可能である．また，実際に制約解消を行う数値最適化アルゴリズムも，最適化モジュールとして与えられており，交換可能となっている．

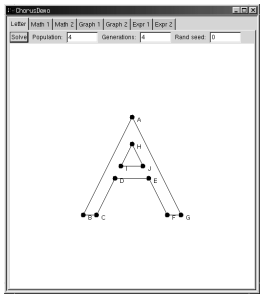
本枠組みはすでに，UI構築のための制約解消系ライブラリChorus☆として，C++とJavaの2種類の言語で実装されている．Chorusによるプログラミングでは，変数と制約をオブジェクトとして生成し，制約オブジェクトを制約解消系オブジェクトに対して追加・削除することで処理を記述する．実際にChorusを用いたソフトウェアとして，これまでに，**図1**に示すようなサンプルアプリケーションや，**図2**に示すような情報視覚化システム[15)]が開発されている．

本論文は，以下の構成からなる．まず次章で，UIを対象とした制約解消に関する従来研究について紹介する．次に，本論文で提案する枠組みにおける制約解消法と実装方式について述べる．その後，この枠組みを実装したChorus制約解消系について概説し，その性能に関する実験結果を与える．そして，提案した枠組みの得失に関する議論の後，結論と今後の課題を述べる．

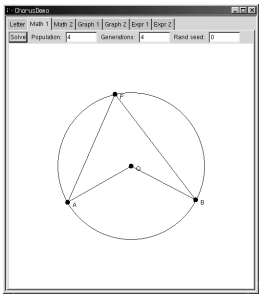
2. 関連研究

UI分野において，これまでに様々な制約解消系が提案されている．その中でも特に，変数の入出力関係に沿って計算を進めるデータフロー制約解消系の研究はさかんに行われた[6),28),32)]．データフロー方式は，制約解消を1種のモジュール機構によって実現する枠組みと見なすことができる．その理由は，データフロー制約の計算方法を，通常のプログラミング言語における関数や手続きとして柔軟に定義できるためである．しかし，データフロー方式では，制約が連立されてい

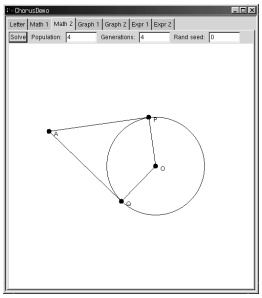
☆ Chorusは，“Constraint hierarchy optimization and resolution system”の略である．



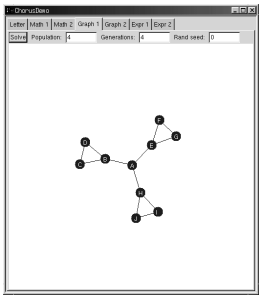
(a)



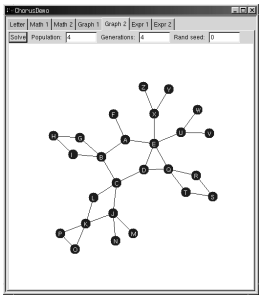
(b)



(c)



(d)



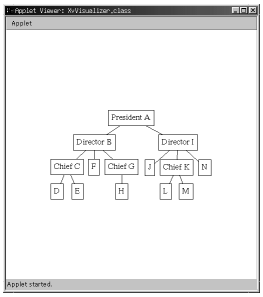
(e)

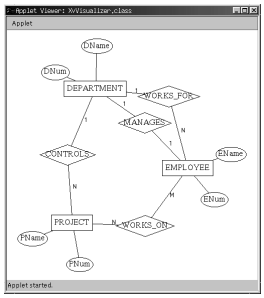
図 1 Chorus 制約解消系のアプリケーション：(a) A 字形の幾何配置；(b) 中心角と円周角の関係を示す対話型アプリケーション（点 A，B，P は円周上に制約されている）；(c) 円の接線に関する数学的アプリケーション；(d) 単純な対称グラフの配置（ノードのドラッグにより，配置の回転を含む移動が可能）；(e) より複雑な非対称グラフの配置

Fig. 1 Applications of the Chorus constraint solver: (a) a geometric layout of the figure of A; (b) an interactive application to teach the relationship between central and inscribed angles (points A, B, and P are constrained to be on the circle); (c) another mathematical application to tangential lines to a circle; (d) a simple symmetric graph layout (which the user can rotate and translate by dragging its node); (e) a more complex, asymmetric graph layout.

る状況や，不等式制約を扱うことが難しいという問題がある．

データフロー方式以外にも，モジュール機構を実現する制約解消手法が提案されている．DeltaStar[5] は，制約階層を制約の優先度に応じたレベル群に分割し，





(a) (b)

図 2 Chorus 制約解消系によって実現された情報視覚化：(a) 組織図；(b) 実体関連図

Fig. 2 Information visualization realized with the Chorus constraint solver: (a) an organization chart; (b) an entity-relationship diagram.

優先度の高いレベルから順に flat solver と呼ばれるモジュールを適用することで，制約階層全体を解消する．*DETAIL*[16] と UltraViolet[1] では，制約階層をグラフ的な連結関係に基づいて分割し，それぞれの成分に含まれる制約の種類に応じた subsolver というモジュールを適用することで制約解消を行う．これらの手法では，制約解消系全体の能力が，実際に可能な個別のモジュールの能力に強く依存しており，新しいモジュールの実現は容易でない．

本論文の枠組みと同様に，数値最適化を利用した制約解消系が，この数年間に相次いで提案されている[3),14),23)]．その中でも，QOCA[23] は，制約解消アルゴリズムのモジュール化を実現することで，制約階層の解の複数の基準に対応している．ただし，これらの制約解消系はすべて，線形等式・不等式制約に特化されたものである．

非線形制約を扱うために，Juno[25] では Newton 法を採用している．また，TRIP[20] や文献 12) のシステムでは，スプリングモデル[19] によるグラフ配置を求めるために，Newton 法を使用している．しかし，遺伝的アルゴリズムなどの確率的手法を用いることもなく，これらの数値的手法だけで大域的な最適解を見つけることは困難である．

文献 10) では，巨大なグラフの大域的に最適な配置を求めるために，それを粗く近似したグラフの局所最適な配置を利用する手法を提案している．ただし，この手法は，本研究で扱っている Euclid 幾何制約によるオブジェクト配置を対象として含むものではない．

Bramble[8] と GLIDE[27] システムは，仮想的な力学シミュレーションを実行することで幾何制約を解消する．これらのシステムでは，ユーザが対話的な操作によって制約解消を修正できるため，不適当な局所解

の訂正が可能である．このような手法は，対話型アプリケーションでは有望であるものの，ユーザの修正操作に頼らない，より純粋な情報視覚化には適していない．

過去にも，幾何制約の解消に遺伝的アルゴリズムを用いる試みがいくつか行われている[7),22),24)]．しかし，著者の知る限りにおいて，Chorus のように，オブジェクトのドラッグなどをサポートする実際的な制約解消系の構築にまで至った例はない．

制約プログラミングの分野で非線形な最適化を実現したシステムとして，Newton[31)] がある．このシステムは，区間演算を用いた Newton 法を基礎とし，大域的な最適化も可能である．ただし，区間演算に基づく最適化手法を実際に UI へ応用した例は見受けられず，その UI に対する有効性は不明である．

3. 制約解消法

本章では，提案する枠組みにおける制約解消法について述べる．

3.1 定 式 化

本節では，本枠組みで扱う問題の定式化として，制約と制約系の定義を与える．

3.1.1 制 約

まず，変数を表現するために，n 個の変数からなる変数ベクトル $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ を導入する．次に，変数の値を表すために，n 個の実数からなる変数値ベクトル $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$ を導入する（v_i は x_i の値を意味する）．

制約の表現のために，誤差関数 $e(\boldsymbol{x})$ を用いる．これは，変数値ベクトル $\boldsymbol{v}$ に対する制約の誤差 $e(\boldsymbol{v})$ を，非負実数値として求める関数である．その結果が 0 であることは，制約が正確に満たされていることを示す．また，誤差関数の勾配 $\nabla e(\boldsymbol{x}) = (\partial e/\partial x_1, \partial e/\partial x_2, \ldots, \partial e/\partial x_n)$ が既知であると仮定する．ただし，$\nabla e(\boldsymbol{x})$ を実際に利用するかどうかは，後に述べる数値最適化手法次第である．

実際の算術等式制約に対する誤差関数の定義では，右辺と左辺の差の 2 乗を用いている．たとえば，2 点 (x_i, x_j)，$(x_{i'}, x_{j'})$ の間の距離を d とする Euclid 幾何制約の誤差関数は，$e(\boldsymbol{x}) = (\sqrt{(x_i - x_{i'})^2 + (x_j - x_{j'})^2} - d)^2$ のように定義される．

3.1.2 制 約 系

本枠組みが扱う制約系は，制約階層[2)] と同様に，制約の強さに対応した有限個のレベルからなる．その解は，強い制約をできるだけ満たし，より弱い矛盾する制約を緩和することで決められる．これは以下のように定義される．$e_{i,j}(\boldsymbol{x})$ を，レベル i $(0 \le i \le l)$ の j 番目 $(1 \le j \le m_i)$ の制約の誤差関数とする．このとき，解 $\boldsymbol{v}$ は，次の最適化問題で決定される．

$$\begin{array}{ll} \underset{\boldsymbol{v}}{\text{minimize}} & E(\boldsymbol{v}) \\ \text{subject to} & e_{0,j}(\boldsymbol{v}) = 0 \quad (1 \le j \le m_0). \end{array} \tag{1}$$

ただし，$E(\boldsymbol{x})$ は次のように定められる目的関数である．

$$E(\boldsymbol{x}) = \sum_{i=1}^{l} \sum_{j=1}^{m_i} w_i e_{i,j}(\boldsymbol{x}).$$

ここで，w_i は，強さ i に対応する重みであり，$w_1 \gg w_2 \gg \cdots \gg w_l$ である．この定式化において，レベル 0 の制約は，**必須**制約（硬い制約）であり，必ず満たされることが保証され（不可能な場合は解なしになる），それ以外の制約は，**選好**制約（軟らかい制約）であり，より強い制約に矛盾する場合には緩和される．直観的に，選好制約は重い（強い）ほど，より良く満たされる．

この定式化は，制約階層を近似するものである．誤差関数の定義に 2 乗を用いる場合，この定式化により，least-squares-better[2),3)] という基準で解かれる制約階層の近似解を得ることができる．最大の違いは，この定式化では，本来は無視されるべき，強い制約に矛盾する弱い制約を多少考慮してしまう点である．しかし，各重み w_i が w_{i+1} よりも十分に大きければ，良い近似解が得られる．

以下では，実際の強さとして，強い順に，`required`（必須），`very strong`，`strong`，`medium`，`weak`，`very weak` の 6 つがあると仮定する．ただし，`very strong` と `very weak` は制約解消系が内部的に使用する強さであり，その外部からは直接指定できないものとする．

3.2 アルゴリズム

以下では，前項で定式化された制約系を解消するアルゴリズムを与える．

3.2.1 必須制約の処理

本アルゴリズムでは，最初に，必須の線形等式制約の処理を行う．これは，線形等式制約の系を解いて一部の変数を消去することで，最適化問題 (1) を，次のような新しい目的関数 $\mathcal{E}(\boldsymbol{x}')$ からなる問題に変換する処理である．

$$\begin{array}{ll} \underset{\boldsymbol{v}'}{\text{minimize}} & \mathcal{E}(\boldsymbol{v}') \\ \text{subject to} & \epsilon_{0,j_k}(\boldsymbol{v}') = 0 \quad (1 \le k \le m_0'). \end{array} \tag{2}$$

ただし，m_0' は，線形等式以外の必須制約の個数であ

り，$\epsilon_{0,j_k}(\boldsymbol{x}')$ は，この変換によって書き換えられた，線形等式でない必須制約の誤差関数である．

3.2.2 局所探索

本アルゴリズムでは，最適化問題 (2) を解くために，数値的な非線形最適化を用いる．ただし，通常の数値最適化手法では，大域的な最適解を見つけることが困難であるため，ここでは，局所解の探索を目的とする．

具体的な数値最適化手法の例としては，準 Newton 法[26]（可変計量法ともいう）があげられる．これは，超 1 次収束する高速な反復法として知られており，過去の履歴を利用することで無駄な探索を避けるため，通常，素朴な Newton 法よりも高速である．この手法は，現在の Chorus 制約解消系で，数値最適化手法の 1 つとして実際に提供されている．

ここで用いる数値最適化手法では，前項で述べた必須制約処理で除去されなかった，線形等式以外の必須制約も扱う．しかし，準 Newton 法などの手法では，これを直接的に実現することができないため，このような場合には，これらの必須制約を，特別な強さ `very strong` の選好制約として処理する．

目的関数の計算における，選好制約の重み w_i は，ここで用いる最適化手法の精度に応じて決定する必要がある．準 Newton 法で倍精度浮動小数点数を用いる場合，目的関数の精度は，10 進数でたかだか 9 桁程度であるため，実際の重みとしては，強さ `very strong`, `strong`, `medium`, `weak`, `very weak` に対して，それぞれ 32^4, 32^3, 32^2, 32^1, 1 を割り当てている．このような重みを用いる場合，たとえば，`strong` の制約 $x = 0$ と `medium` の制約 $x = 100$ からなる制約系に対して，解 $x = 3.0303\cdots(= 100/33)$ が得られることになる．したがって，制約階層の近似として十分であるとはいえないものの，制約の強さの違いは明白となる．

3.2.3 大域探索

本アルゴリズムでは，数値的最適化手法が局所解に陥りやすいという欠点を補うために，遺伝的アルゴリズム[13),21)] による大域的な探索を導入する．一般に，遺伝的アルゴリズムとは，複数の解候補の集団から次の世代の解候補の集団を生成する処理を繰り返す，確率的な探索手法である．

図 3 は，ここで用いる遺伝的アルゴリズムを擬似プログラムとして記述したものである．このアルゴリズムでは，最初に，n 個の解候補の集団をランダムに生成し，それらに対して，準 Newton 法などの数値最適化手法を適用する．その後，評価，選択，交叉，突然変異と呼ばれる遺伝操作と，数値最適化によって，

```
genetic_algorithm() {
  n 個の解候補からなる初期集団を生成する;
  各解候補に対して数値最適化を適用する;
  各解候補を評価する;
  while (評価結果が収束していない) {
    重複を許して 2n 個の解候補を選択する;
    選択された解候補 2 つごとに
        新しい解候補を 1 つ交叉により生成する;
    確率的に新しい解候補に突然変異を加える;
    新しい解候補に数値最適化を適用する;
    新しい解候補を評価する; }
  return 最適な解; }
```

図 3 遺伝的アルゴリズムの擬似プログラム
Fig. 3 The pseudo program of the genetic algorithm.

新しい n 個の解候補の集団を生成するという処理を，解候補の集団が十分に収束するまで繰り返す．

具体的な遺伝操作は，以下のように行っている．解候補 $\boldsymbol{v}$ の評価には，$\mathcal{E}(\boldsymbol{v})$ を用いる．新しい解候補のための親の選択には，トーナメント選択[21)] により，任意の 2 つの解候補を選んで，より良い解候補を親の 1 つとする（重複的な選択を許す）．2 つの親の解候補 $\boldsymbol{v}_1$，$\boldsymbol{v}_2$ から子の解候補を 1 つ生成する交叉では，線形交叉[13)] により，$(\boldsymbol{v}_1 + \boldsymbol{v}_2)/2$，$(-\boldsymbol{v}_1 + 3\boldsymbol{v}_2)/2$，$(3\boldsymbol{v}_1 - \boldsymbol{v}_2)/2$ のいずれかを求めて，子の解候補とする．子の解候補を変更する突然変異では，確率的に解候補のベクトルの 0.5 個の要素に対して，乱数を代入する．

以上の遺伝操作は，大域探索における解候補の多様性を維持するために，あえて収束が遅くなるように設計したものである．このため，実際の制約解消系では，遺伝的アルゴリズムを打ち切る最大の世代数を，集団の大きさ n とともに，外部から明示的に指定できるようにしている．

3.2.4 制約系の修正

UI を対象とする場合，制約解消系は制約系の修正をサポートする必要がある．これは，幾何的構造の変更のために，新規の制約が追加されたり，既存の制約が削除されたりするためである．また，対話型アプリケーションやアニメーションの実現のために，解消系は，変数値を反復的に更新する edit 制約を提供する必要がある．

制約系の修正において，制約解消系はユーザに対して予測可能な振舞いをする必要がある[25)]．すなわち，新しい制約系を解く際に，新しい解を以前の解にできるだけ近づける必要がある．そうしなければ，得られる図形が大きく変化して，ユーザを驚かせたり混乱させたりしてしまうからである．

多くの場合，新しい大域最適解は，以前の解の近く

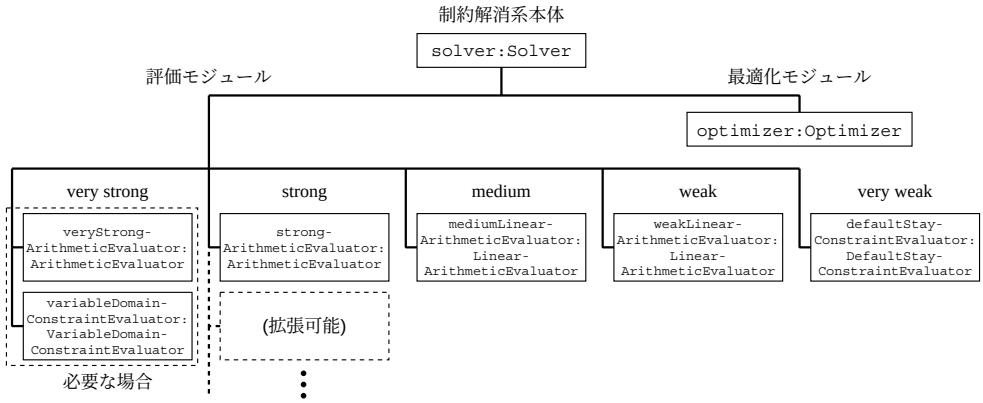


図 4 制約解消系のオブジェクト構造（`object:Class` は `object` が `Class` のインスタンスであることを意味する）
Fig. 4 The object structure of a constraint solver (`object:Class` means that `object` is an instance of `Class`).

に存在する．このため，制約系が修正された際，本アルゴリズムでは，通常，以前の解から出発して，数値最適化で新しい局所最適解を探索する．

しかし，新しい大域最適解が近くにない可能性もあり，この手法だけでは不十分である．このような場合には，すべての変数について，その値を固定する `very weak` の stay 制約を内部的に生成したうえで，遺伝的アルゴリズムによる大域探索を実行する．さらに，遺伝的アルゴリズムの終了後に，`very weak` の stay 制約を外して，数値最適化のみを適用することで，最終的により正確な解を求める．

4. 実装方式

本章では，前章の制約解消法を制約解消系ソフトウェアとして有効に実装する方式として，オブジェクト指向プログラミングにより，モジュール機構を実現する手法を提案する．

4.1 モジュール機構

前章の制約解消法では，個々の制約が誤差関数 $e(\boldsymbol{x})$ として与えられているために，通常の算術制約については，$e(\boldsymbol{x})$ とその勾配 $\nabla e(\boldsymbol{x})$ を計算するプログラムを記述することで表現可能である．そこで，制約の種類に応じて，これらの関数を計算するプログラムを，評価モジュールとして分離し，制約の具体的な意味づけを与えるようにする．

さらにこの考えを発展させて，非算術的な（あるいは擬似的な）制約についても，評価モジュールによる意味づけを許すようにする．すなわち，ある種類の制約の 1 つ 1 つに対しては誤差関数を定義することが難しくても，その種類の制約全体に対して，誤差関数に相当するものを評価モジュールで計算することを可能にする．後に示すように，この方法によって，一般のグラフ配置なども扱うことができるようになる．

また，数値最適化による局所探索のプログラムも，交換可能な最適化モジュールとして部品化する．制約解消アルゴリズムの他の部分である，線形計算による必須制約の処理や，遺伝的アルゴリズムによる大域探索は，数値最適化のプログラムとは独立に実装できるからである．

これらのモジュール化により，制約解消系のオブジェクト構造は**図 4** のようになる．制約解消系の本体であるオブジェクト `solver` は，複数個の評価モジュール群と，1 個の最適化モジュール `optimizer` を保持する形となる．

制約解消系の処理の大まかな流れは以下のようになる．`solver` は制約を受け取ると，その強さと種類に応じて，適切な評価モジュールに引き渡す．たとえば，強さ `medium` の線形制約は，`mediumLinearArithmetic-Evaluator` へ引き渡す．ただし，`required`（必須）の線形等式制約に限っては評価モジュールに送らず，`solver` が線形計算で処理する．一方，実際の制約解消の際に，`solver` は必須線形等式制約の処理と大域探索を行い，その中で，`optimizer` に対して局所探索を依頼する．`optimizer` は，評価モジュールを呼んで目的関数を繰り返し計算しながら，数値最適化を実行する．

4.2 制約と評価モジュール

制約は，**図 5** (a) に示すように，クラス `Constraint` を継承して定義される．一方，評価モジュールは，図 5 (b) のように，クラス `Evaluator` を継承する．

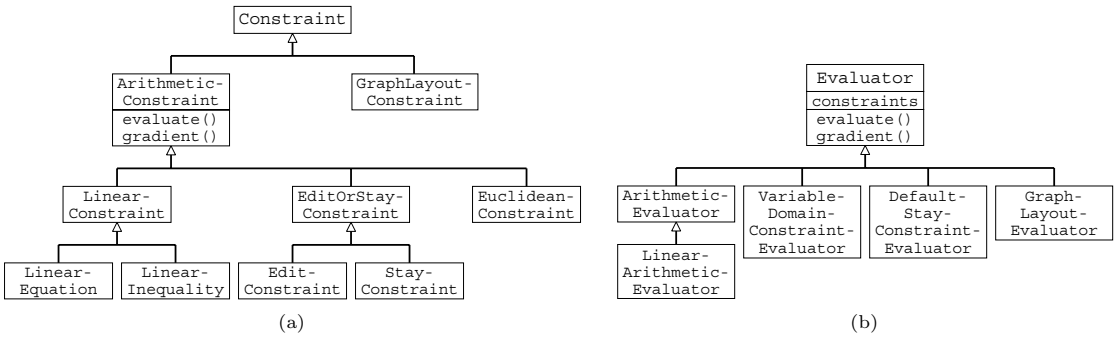


図 5 (a) 制約と (b) 評価モジュールのクラス階層
Fig. 5 The class hierarchies of (a) constraints and (b) evaluation modules.

各評価モジュールは，自身が担当する制約のリスト constraints を保持し，担当するすべての制約の誤差関数とその勾配を統合するメソッド evaluate() と gradient() を定義する．

4.2.1 算術制約

制約には，大きく分けて，クラス ArithmeticConstraint に属する算術制約と，それ以外の制約の 2 種類がある．算術制約は，1 つ 1 つについて，誤差関数 $e(\boldsymbol{x})$ とその勾配 $\nabla e(\boldsymbol{x})$ を定義でき，ArithmeticConstraint は，これらをメソッド evaluate() と gradient() として実現する．

算術制約の処理は，クラス ArithmeticEvaluator に属する評価モジュールが行う．このクラスの evaluate() と gradient() は，それぞれ担当する各制約の evaluate() と gradient() を呼び出し，その総和をとることで計算される．たとえば，図 4 においては，評価モジュール strongArithmeticEvaluator が，ArithmeticEvaluator のインスタンスであり，強さ strong の算術制約を処理するものである．

なお，現在の Chorus 制約解消系では，レベル medium と weak は線形制約に制限されている☆．このため，ArithmeticEvaluator のサブクラスとして，線形制約のみに制限された LinearArithmeticEvaluator を導入し，これらのレベルに使用している．

4.2.2 非算術制約

非算術制約とは，1 つずつ別々に，誤差関数 $e(\boldsymbol{x})$ を定義することが難しいものをいう．このため，そのクラスは，メソッド evaluate() と gradient() を持たず，その制約特有の情報を何らかの形で保持する．非算術制約に対応する評価モジュールは，このような情報から，evaluate() と gradient() を独自に計算する．

現在の Chorus 制約解消系では，非算術制約として GraphLayoutConstraint を提供し，その評価モジュールとして GraphLayoutEvaluator を用意している．これらは，スプリングモデル[19] を用いることで，図 1 (d) と図 1 (e) に示したようなグラフ配置を実現するものである．GraphLayoutConstraint のインスタンスは，グラフのエッジを表しており，プログラマーはエッジを制約として記述できる．

スプリングモデルでは，グラフ内の任意の 2 つのノード（間接的に接続されている場合も含む）どうしをバネで接続した力学系を想定し，バネのエネルギーの総和を最小とする解を，求めるべきグラフ配置とする．GraphLayoutEvaluator は，このようなエネルギー全体の関数を evaluate() として与えることで，グラフ配置を実現している．

4.2.3 暗黙的な制約

外部から明示的に与えられる制約以外に，制約解消系が内部的に生成する暗黙的な制約がある．このような制約には，各変数の可能な値の範囲を記述する，very strong の変数領域制約と，3.2.4 項で述べた，very weak の stay 制約の 2 種類がある．これらの暗黙の制約は，実際に制約をオブジェクトとして生成するのではなく，専用の評価モジュールのクラス VariableDomainConstraintEvaluator と DefaultStayConstraintEvaluator を用いて実現している．ただし，局所探索のための数値最適化手法が必須線形不等式制約を処理できる場合には，変数領域制約は必須制約として処理される．

4.3 最適化モジュール

最適化モジュールは，3.2.2 項で述べた，局所探索のための数値最適化手法を実現するものである．これは，クラス Optimizer を継承して定義され，メソッド optimize() を実装する．

☆ この制限は，目的関数の数値的な性質を良くすることを狙ったものである．

現在の Chorus 制約解消系では，以下の 2 つの最適化モジュールを提供している．最適化モジュール `QuasiNewtonOptimizer` は，Broyden-Fletcher-Goldfarb-Sahnno 公式による準 Newton 法[26] を用いた数値最適化を行う．すでに述べたように，準 Newton 法自体は目的関数の最適化のみを行うもので，必須制約を扱うことができない．このため，制約解消系本体 `solver` によって処理されない線形等式以外の必須制約は，`very strong` の選好制約として近似的に扱われる．

最適化モジュール `Donlp2Optimizer` は，非線形計画法のためのライブラリ DONLP2[29] を利用する．DONLP2 は，非線形な必須制約のもとで目的関数を最適化できる．このため，このモジュールでは，線形等式以外の必須制約を DONLP2 に引き渡すことで，そのまま必須制約として処理することが可能である．

5. Chorus 制約解消系

3 章の制約解消法と 4 章の実装方式に基づき，Chorus 制約解消系を開発した．Chorus がデフォルトでサポートしている制約の種類は，線形等式，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）である．さらに幾何制約を拡張することが可能であり，現時点では，平行，垂直，距離の一定などの Euclid 幾何制約と，スプリングモデル[19] に基づくグラフ配置制約を提供している．

Chorus は，制約系の指定に，制約の追加・削除を基本操作とする差分的な編集モデルを採用している．すなわち，グラフィカルオブジェクトへの操作に際して，必要な新規の制約を制約系へ追加したり，既存の制約を削除したりすることで制約系を更新し，対応する解を計算できるようにしている．また，オブジェクトの移動操作を簡潔に記述する，edit 制約による変数の値の反復的な更新も可能である．

このような処理を記述するアプリケーションプログラミングインタフェースは，最近の線形制約解消系である Cassowary[3] との互換性を配慮して設計されたものである．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，制約オブジェクトを制約解消系オブジェクトに対して追加・削除することで処理を記述する．

Chorus には，C++版と Java 版の 2 種類がある．そのプログラムの大きさは，C++版で約 6,000 行である．また，図 1 と図 2 に示したように，すでに複数のアプリケーションやシステムを Chorus を用いて作成している．図 1 に示したアプリケーションは C++ で作成したもので，GTK+（Gimp Toolkit）を用いており，UNIX と Microsoft Windows のいずれの上でもコンパイルと実行が可能である．また，同様のアプリケーションの Java 版も作成している．一方，図 2 に示したシステムは，XML 文書として記述されたデータを図形的に視覚化するもので，Java によって実装されている．

6. 実　　験

本論文で提案した枠組みの評価のために，C++版の Chorus 制約解消系を用いて実験を行った．プログラムのコンパイルは，GNU C EGCS-2.91.66 で最適化オプション–O3 を付けて行い，実行は，Linux 2.2.14 で動作する Intel Pentium III 800 MHz を搭載したパーソナルコンピュータ上で行った．

最初の実験として，図 1 (a) と図 1 (e) に示されるアプリケーションを用いて，遺伝的アルゴリズムによる大域探索を行わず，ランダムな初期配置から出発して，`QuasiNewtonOptimizer` または `Donlp2Optimizer` を用いた数値最適化による局所探索を 1 回だけ実行する場合の時間を測定した．実際に，最初の解を求めるために，遺伝的アルゴリズムによる大域探索をする場合には，この実験で得られる結果の数十倍の時間がかかると考えてよい．一方，ノードのドラッグなどにともなう局所探索では，すでに分かっている従来の解の近くに新しい解が存在するため，通常，この結果よりも短い時間で計算が終了する．

図 1 (a) の幾何配置では，点の座標を表す 20 個の変数と，12 個の `required` の線形等式制約，7 個の `required`（`QuasiNewtonOptimizer` の場合は `very strong`）の線形不等式制約，5 個の平行に関する `strong` の制約，5 個の距離に関する `strong` の制約が使われている．この配置について 10 回局所探索を実行して得られた平均の時間は，`QuasiNewtonOptimizer` で 16 ミリ秒，`Donlp2Optimizer` で 47 ミリ秒だった．一方，図 1 (e) のグラフ配置は，ノードの座標を表す 52 個の変数と，エッジを表す 31 個の制約からなる．これを 10 回計算して得られた平均の実行時間は，`QuasiNewtonOptimizer` で 117 ミリ秒，`Donlp2Optimizer` で 130 ミリ秒だった．

第 2 の実験として，**図 6** に示されるアプリケーションを用いて，遺伝的アルゴリズムによる大域探索の効果を調べた．このアプリケーションは，長方形が連なった梯子型のグラフ配置を求めるものである．この配置の計算では，梯子型の途中でねじれることによる局所解が生じやすいため，大域探索が必要となる．この実験における局所探索には，`QuasiNewtonOptimizer` を

用いた．

実験では，12 個の長方形が連なる配置を対象とし，集団の大きさを 10，最大の世代数を 10 とした．このような大域探索を 10 回試行した場合の目的関数の値の変化を**図 7** に示す．1 つのグラフは 1 回の試行に対応しており，横軸は世代を示し，縦軸は目的関数の値を表す．ここで，各世代には，各個体に対応する 10 個の点が打たれている．ただし，異なる個体で目的関数の値が等しくなっている状態を判別できるように，各点に対して意図的に左右のずれを与えている．

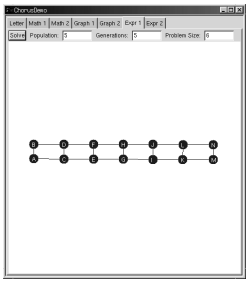


図 6 遺伝的アルゴリズムに関する実験のためのアプリケーション
Fig. 6 The application for the experiment on the genetic algorithm.

この 10 回の試行ではいずれも，グラフ配置にねじれのない，大域的な最適解が得られた．また，1 回の試行の平均の実行時間は，10.5 秒だった．大域探索の進行状況としては，グラフから分かるように，最初から最適解が得られることはほとんどなかったが，世代の進行につれて，解候補の集団が徐々に最適解へと収束していった．時折，世代が進んだにもかかわらず，最適解の個数が減少したり，評価値の低い解候補が発生したりすることがあったが，これは遺伝的アルゴリズムにおける選択や突然変異などの影響によるものである．

最後の実験として，遺伝的アルゴリズムを用いた大域探索との比較のために，第 2 の実験と同様の条件で，

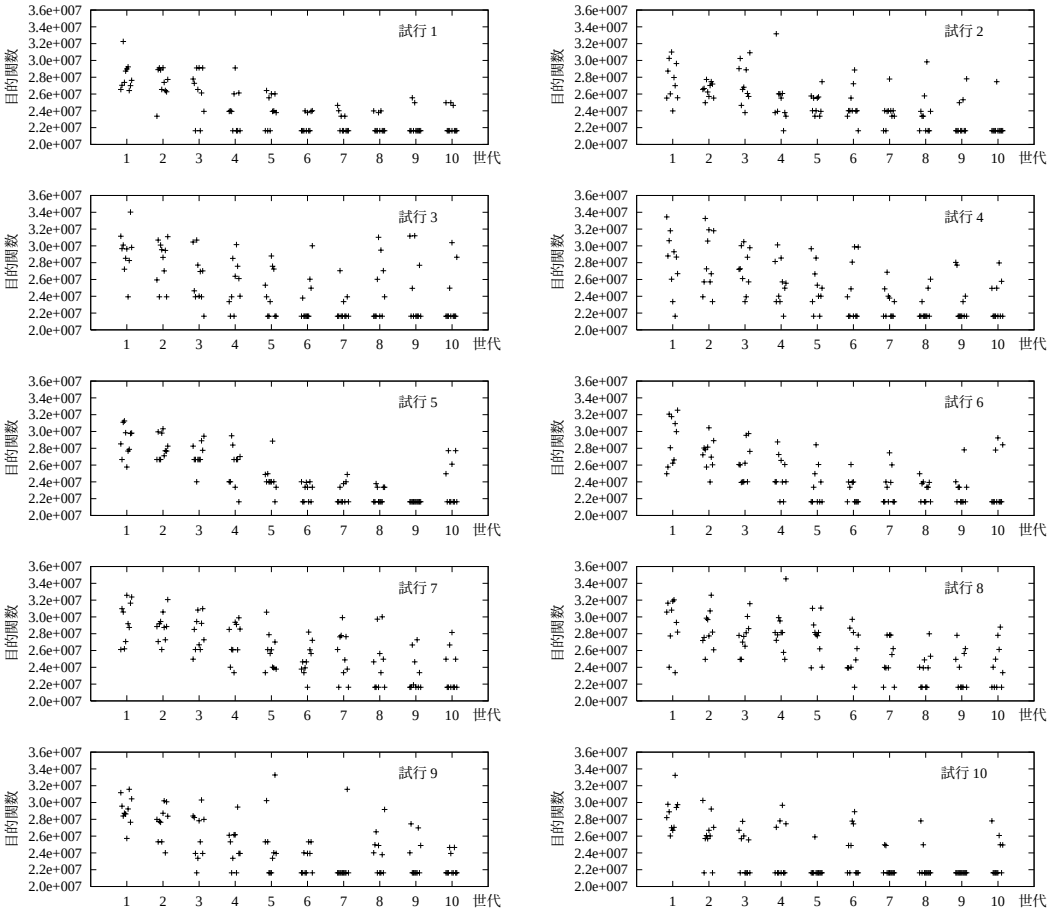


図 7 遺伝的アルゴリズムに関する実験結果
Fig. 7 The experimental results of the genetic algorithm.

焼鈍し法のみを 1 回実行する場合の時間と解の質を調べた．焼鈍し法は大域探索のための確率的アルゴリズムの 1 つで，ここでは既存のパッケージ[4] を用いた．これを 10 回実行して得られた平均の時間は，748 ミリ秒だった．この 10 回のうちで，大域的な最適解を実際に求めることができたのは 1 回だけであり，グラフ配置には平均で 2.1 個のねじれが生じた．ただし，他の焼鈍し法の実装を用いることで，より良い結果が得られる可能性は残されている☆．

7. 議　　論

本枠組みにおける制約解消法は，基本的に，制約階層の近似解を求めるものであるため，その精度が問題になる状況がありうる．具体的には，充足不能な弱い制約が多数存在する場合に，本来は満たすべき強い制約を十分に充足できなくなる可能性がある．同様の問題は，QOCA[23] のような線形の制約階層解消系でも存在しており，その解決のために，Cassowary[3] では「記号的」な重みを導入し，HiRise[14] では階層独立性解析という手法を採用している．しかし，これらの手法は，本枠組みにおける非線形な制約解消に直接応用することができない．

本枠組みにおける制約解消法は，解の大域探索のために遺伝的アルゴリズムを採用しているという点で特徴的である．その最大の利点は，局所数値最適化アルゴリズムを大域探索処理とは独立に選択できる点である．このことにより，本枠組みにおける実装方式では，様々な数値最適化アルゴリズムを最適化モジュールとして実現しており，最適化モジュールを交換することが可能となっている．一方，遺伝的アルゴリズム以外の大域探索手法を用いた場合に，このような機能を容易に実現できるとは限らない．

現在の実装方式では，1 つの制約解消系に対して 1 つの最適化モジュールが割り当てられることを仮定している．しかし，このことは必須ではなく，状況に応じて複数の最適化モジュールを使い分けたり，遺伝的アルゴリズムにおいて個体ごとに異なる最適化モジュールを適用したりするなど，複数個の最適化モジュールを利用する拡張なども考えられる．

8. おわりに

本論文では，UI を対象とした幾何制約解消の枠組みを提案した．本枠組みにおける制約解消法は，数値最適化による局所探索と，遺伝的アルゴリズムによる大域探索を組み合わせることで，Euclid 幾何制約やグラフ配置制約などの高い表現力を達成した．また，モジュール機構を実現する実装方式により，新しい制約の追加や，数値最適化手法の交換を容易にした．

今後の研究課題としては，以下のものがあげられる．まず，現在の制約解消法では，制約階層を十分に正確に解くことができないため，その近似精度の改善を試みる．また，応用面では，より高度なグラフ配置手法のための評価モジュールの実現や，3 次元アプリケーションへの対応などについて計画している．

☆ すでに別の実装として文献 9) を試みたが，この場合は解の局所的な精度に問題を生じた．

参 考 文 献

1) Borning, A. and Freeman-Benson, B.: Ultra-Violet: A Constraint Satisfaction Algorithm for Interactive Graphics, *Constraints J.*, Vol.3, No.1, pp.9–32 (1998).
2) Borning, A., Freeman-Benson, B. and Wilson, M.: Constraint Hierarchies, *Lisp and Symbolic Comput.*, Vol.5, No.3, pp.223–270 (1992).
3) Borning, A., Marriott, K., Stuckey, P. and Xiao, Y.: Solving Linear Arithmetic Constraints for User Interface Applications, *Proc. ACM UIST*, pp.87–96 (1997).
4) Carter, E.: Simulated Annealing Package. http://www.taygeta.com/annealing/.
5) Freeman-Benson, B., Wilson, M. and Borning, A.: DeltaStar: A General Algorithm for Incremental Satisfaction of Constraint Hierarchies, *Proc. IEEE IPCCC*, pp.561–568 (1992).
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) Frick, A., Keskin, C. and Vogelmann, V.: Integration of Declarative Approaches, *Graph Drawing–GD'96*, LNCS, Vol.1190, pp.184–192, Springer (1997).
8) Gleicher, M.: A Graphical Toolkit Based on Differential Constraints, *Proc. ACM UIST*, pp.109–120 (1993).
9) Goffe, B.: SIMANN, Netlib. http://www.netlib.org/opt/simann.f.
10) Harel, D. and Koren, Y.: A Fast Multi-Scale Method for Drawing Large Graphs, *Graph Drawing–GD2000*, LNCS, Vol.1984, pp.183–196, Springer (2001).
11) 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），レクチャーノート/ソフトウェア学，Vol.16, pp.41–49, 近代科学社 (1996).
12) He, W. and Marriott, K.: Constrained

Graph Layout, *Graph Drawing–GD'96*, LNCS, Vol.1190, pp.217–232, Springer (1997).
13) Herrera, F., Lozano, M. and Verdegay, J.L.: Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis, *Artif. Intell. Rev.*, Vol.12, No.4, pp.265–319 (1998).
14) Hosobe, H.: A Scalable Linear Constraint Solver for User Interface Construction, *Principles and Practice of Constraint Programming–CP2000*, LNCS, Vol.1894, pp.218–232, Springer (2000).
15) 細部博史，本位田真一：XML を対象とした制約の利用による情報視覚化方式，インタラクション 2001 論文集，pp.83–90, 情報処理学会 (2001).
16) Hosobe, H., Miyashita, K., Takahashi, S., Matsuoka, S. and Yonezawa, A.: Locally Simultaneous Constraint Satisfaction, *Principles and Practice of Constraint Programming–PPCP'94*, LNCS, Vol.874, pp.51–62, Springer (1994).
17) Igarashi, T., Matsuoka, S., Kawachiya, S. and Tanaka, H.: Interactive Beautification: A Technique for Rapid Geometric Design, *Proc. ACM UIST*, pp.105–114 (1997).
18) 丁 錫泰，田中二郎：Rainbow：ビジュアルシステム生成系におけるレイアウト制約の実現，情報処理学会論文誌，Vol.41, No.5, pp.1246–1256 (2000).
19) Kamada, T. and Kawai, S.: An Algorithm for Drawing General Undirected Graphs, *Inf. Process. Lett.*, Vol.31, No.1, pp.7–15 (1989).
20) Kamada, T. and Kawai, S.: A General Framework for Visualizing Abstract Objects and Relations, *ACM Trans. Gr.*, Vol.10, No.1, pp.1–39 (1991).
21) 北野宏明（編）：遺伝的アルゴリズム，産業図書 (1993).
22) Kosak, C., Marks, J. and Shieber, S.: Automating the Layout of Network Diagrams with Specified Visual Organization, *IEEE Trans. Syst. Man Cybern.*, Vol.24, No.3, pp.440–454 (1994).
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–CP98*, LNCS, Vol.1520, pp.340–354, Springer (1998).
24) Masui, T.: Graphic Object Layout with Interactive Genetic Algorithms, *Proc. IEEE VL*, pp.74–80 (1992).
25) Nelson, G.: Juno: A Constraint-based Graphics System, *Proc. ACM SIGGRAPH*, pp.235–243 (1985).
26) Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T.: *NUMERICAL RECIPES in C* [日本語版], 技術評論社 (1993).
27) Ryall, K., Marks, J. and Shieber, S.: An Interactive Constraint-Based System for Drawing Graphs, *Proc. ACM UIST*, pp.97–104 (1997).
28) Sannella, M.: SkyBlue: A Multi-Way Local Propagation Constraint Solver for User Interface Construction, *Proc. ACM UIST*, pp.137–146 (1994).
29) Spellucci, P.: DONLP2, Netlib. http://www.netlib.org/opt/donlp2/.
30) Sutherland, I.E.: Sketchpad: A Man-Machine Graphical Communication System, *Proc. AFIPS Spring Joint Conf.*, pp.329–346 (1963).
31) Van Hentenryck, P., Michel, L. and Benhamou, F.: Newton: Constraint Programming over Nonlinear Constraints, *Sci. Comput. Prog.*, Vol.30, No.1–2, pp.83–118 (1998).
32) Vander Zanden, B.: An Incremental Algorithm for Satisfying Hierarchies of Multi-Way Dataflow Constraints, *ACM Trans. Prog. Lang. Syst.*, Vol.18, No.1, pp.30–72 (1996).

(平成 12 年 10 月 31 日受付)
(平成 13 年 2 月 1 日採録)

細部 博史（正会員）

1969 年生．1993 年東京大学理学部情報科学科卒業．1995 年同大学大学院理学系研究科情報科学専攻修士課程修了．1998 年同専攻博士課程修了．博士（理学）．日本学術振興会特別研究員-PD，文部省学術情報センター助手を経て，2000 年より国立情報学研究所助手．制約プログラミング，ユーザインタフェース，対話型グラフィックス等に興味を持つ．日本ソフトウェア科学会，ACM 各会員．