

The Cassowary Linear Arithmetic Constraint Solving
Algorithm: Interface and Implementation

情報学科

1 G 0 1 p 0 1 8 - 1 宇佐美智史

passion.of.ice@suou.waseda.jp

1 Abstract

1 ページ

線形の等式・不等式制約は普通、あるウインドウが他のウインドウの左にあることを要求することや、窓ガラスがあるウインドウの左側3分の1を占めるように要求することや、もし可能であるなら、あるオブジェクトを三角形の中に入れることを要求するといったような、ユーザーインターフェイスの多くの側面を特定する上で起こってくる。UI アプリケーションに対してデザインされた現在の制約ソルバでは、効果的に連立の等式・不等式を扱うことが出来ない。このことは大きな制限である。そのような制約系を効果的に解くことのできる二次元シンプレックス法をもとにしたインクリメンタルなアルゴリズムである、Cassowary をつくる。

この非公式のテクニカルレポートは Cassowary のアルゴリズムの最新バージョンについて述べている。それは UIST'97 Proceedings で出版された、Alan Borning, Kim Marriott, Peter Stuckey, Yi Xiao 著である、「Solving Linear Arithmetic Constraints for User INterface Applications」に由来している。UIST 紙は非常に近いソルバで線形制約に対して最小2乗法を見つける QOCA の記述も含まれている。・・・

2 Introduction

線形の等式・不等式制約は自然とユーザーインターフェイスの多くの点を特徴付ける際に出てくるものである、特にレイアウトや他の幾何的な関係である。不等式制約は、特に、"inside", "above", "below", "left-of", "right-of", "overlaps" といった関係を表現するのに必要とされる。例えば、ウェブドキュメントをデザインしようとしているとして、figure1 は figure2 の左側にあるという要求を $figure1.rightSide \leq figure2.leftSide$ というように表現できる。

2 ページ

制約系において要求と同様に優位性も表現することができることは重要なことである。一つの用法は、あるイメージの一部が動いている時に、安定性への要求を表現することである。ものは、それらが動くのに何かの理由がない限り今までいた場所に居続けるべきである。二つ目の用法は、潜在的に価値のないユーザ入力を優雅な方法で処理することです。例えば、ユーザがある図形を制限窓の外に動かそうと試みると、エラーと与えるよりも、窓の側面に押し上げ、止める方がよい。三番目の用法は、衝突している要求のバランスをとることです、例えば、グラフをレイアウトするときです。

もし、制約ネットワークが非同期的なら、効果的なテクニックがそのような制約を解くのに利用できる。しかしながら、現実世界の問題に大して制約系を適用しようとすると、制約の集合はしばしば同期的であることに気づいた。このことは、時々、プログラマーが冗長な制約を加える際に起こる、サイクルは注意深い解析によって避けられることができたのだ。しかしながら、このことはプログラマーにより負担になる。さらに、プログラマーに常にサークルと冗長な制約を避けるように要求するというのはすべての企業精神に明らかに反する。結局、制約を与える上での一つのゴールはプログラマーにプログラマーが宣言的なファッション？において保ちたい関係を述べるのを許すことである、これらの関係を施行するために今あるシステムに適応させる。他のアプリケーションにとって様々なゴールやサークルを伴う複雑なレイアウトはさげられない。

2.1 Constraint Hierarchies and Comparators

制約系において要求と同様に優劣を表現できるようになりたいので、衝突している優劣がどのように交換されているかの明細が必要である。制約階層制このことに対する一般的な理論である。制約階層制ではそれぞれの制約は強さをもっている。required という強さは特別であり、この強さの制約は満たされなければならない。ほかの強さはすべて non-required の制約というようになる。与えられた強さの制約は完全に自分より弱い強さを持った制約を支配する。理論では、コンパレータは制約に対して異なる可能な解を比べ、その中から選ぶように使われている。

このフレームワーク内では多くのバリエーションが可能である。一つの決定は制約ごとを基本に解を比較するかどうか (a local comparator)、強さが与えられている制約で満たされていないものの集合測定を取るかどうか (a global comparator)。二つ目の選択は制約が満たされているかどうかというだけを心配する (a predicate comparator)、またそれがどの程度満たされているかどうかを知りたいかどうか (a metric comparator)、()

Indigo ソルバを考えると、不等式制約においては a predicate comparator より、a metric comparator の方を使うことが重要である。線形の等式・不等式を伴った用法でもっともらしいコンパレータは、locally-error-better、weighted-sum-better、least-squares-better である。制約の束が与えられると、Cassowary は locally-error-better な解と weighed-sum-better な解を見つける。(反対に、QOCA は least-squares-better な解を見つける。least-squares-better コンパレータは、同じ強さの制約が交換される時に、強く中心から離れた値にペナルティを与える。それは、本質的に衝突する優劣がある木や、グラフや、窓の集合をレイアウトするような仕事に適している。) locally-error-better コンパレータはより許容的なコンパレータである、制約よりも多くの解を認めるという点で。(実際にどんな least-squares-better やどんな weighed-sum-better な解もまた、locally-error-better な解である。) locally-error-better な解を見つけるアルゴリズムと実践することはより簡単である、特に、連立の等式・不等式に対するサブソルバと数の制約を含んだサブソルバを含んだハイブリッドなアルゴリズムをデザインすることである。

2.2 Adapting the Simplex Algorithm

3 ページ

線形プログラミングは次のような問題を解決することと関係がある。n 個の実数変数の集合を考える、そのそれぞれは 0 以上である。ここに m 個の線形等式、または線形不等式制約が x_i においてある、その形は以下の通り。

これらの制約が与えられて、objective function の値を最小化 (または最大化) するそれぞれの x_i に対する値を見つけない。

この問題はここ 50 年間強く研究されてきた。その問題を解くのに一番一般的に使われたアルゴリズムは 1940 年に Dantzig によって発展させられた、今ではその多くのバリエーションが存在する。不運なことに、存在しているシンプレックス法の実践は UI アプリケーションにとって本当に適切なものではないのである。

主な問題は増大性である。インタラクティブなグラフィカルアプリケーションは似たような問題を繰り返し解く必要がある、ひとつの問題を一回解くというよりもむしろ。インタラクティブなレスポンスタイムを達成するために、とても速いインクリメンタルなアルゴリズムが必要とされる。ここに 2 つの場合がある、最初の

一つは、マウスや他の入力デバイスであるオブジェクトを移動させようとする時、私達は典型的に図形の一部である要求されている x 、 y の組にマウスの位置を関係付けるという相互関係を表現する。この場合は、同じ制約の集合を再度満たさねばならない、単にマウスの位置が異なっただけで、毎時間スクリーンは書き換えられている。2番目には、オブジェクトを編集している時、私達は制約と他の部品を加えたり、取り除いたりしたい、これらの作業を速くしたい、できる限り前の求めた解を再利用することによって。作業要求は2番目よりも1番目の方が非常にきびしい。

他の問題は、適切な目的関数を定義することである。標準的なシンプレックス法のアルゴリズムでの目的関数は線形方程式でなければならないが、locally-error-better、weighted-sum-better、least-squares-better なコンパレータに対する目的関数はすべて非線形である。幸運なことにこれらの場合を扱ったオペレーションリサーチにおいて技術が発達し、ここにおいて適用している。最初の2つのコンパレータは目的関数はほぼ線形であるが、一方、3つ目のコンパレータは2時曲線問題になってしまう。

さいごに3つ目の問題は、正と負、両方の値をとる変数を調整することである。一般的に、UI アプリケーションの場合であるが。(標準的なシンプレックスアルゴリズムでは、すべての変数は非負である。) 制約論理プログラミングを実践する今夏的な技術をここに提案する。

4 ページ

3 Incremental Simplex

お解りのように、線形プログラミングは用語や記号のやぶに囲まれている。これらの厄介な防御は厳格なその分野に没頭している持者によって開拓された。実際に線形プログラミングの基本の考えはとてもシンプルである。

今、手作業でシンプレックス法のインクリメンタルなバージョンについて説明する。説明では、Figure 1 でのダイアグラムで描かれている running example を使う。

5 ページ

3.0.1 内容について

3.0.2 実現上の課題

3.1 問題2の解答