Solving Difference Constraints Incrementally

G. Ramalingam, J. Song, L. Joscovicz and R. E. Miller

presented by

Veena S. Mellarkod

Feb 17, 2006

Structure

The talk is structured as follows:

- introduce difference constraints
- problem description
- a simple algorithm
- an improved algorithm

Introduction

A system of *difference constraints* $< V, C >$

- set V of variables
- set C of inequalities of the form $v - u \leq b$, where $v, u \in V$ and b is a real

A *feasible solution* for a system of difference constraints is an assignment of real values to the variables that satisfies all the constraints

$\{x - y \leq 5\}$ $\;and\;$ $\{x - y \leq 0,\ y - x \leq -1\}$

A system is feasible iff it has a feasible solution

Constraint Graph

The constraint graph of a system of difference constraints $< V, C >$ is a directed, weighted graph $G =< V, E, length >$ where

$$E = \{u \rightarrow v \ \ | \ \ v - u \leq b \in C\}$$

$$length(u \rightarrow v) = b \quad iff \quad v - u \leq b \in C$$

$$Example: \{x - y \leq 3, \quad y - z \leq -1, \quad x - z \leq 4\}$$

For convenience, we will assume that a system of constraints contains at most one inequality per ordered pair of variables.

Augumented Constraint Graph

The augumented constraint graph of a system of difference constraints $< V, C >$ is a directed, weighted graph $G' =< V', E', length' >$ where

- $V' = V \cup \{src\}$ where $src \notin V$
- $E' = E \cup \{src \rightarrow v \ \ | \ \ v \in V\}$
- $length'(u \rightarrow v) = b \quad if\ v - u \leq b \in C$
- $length'(src \rightarrow v) = 0 \quad for\ v \in V$

Theorems

Theorem: A system of difference constraints is consistent if and only if its augumented constraint graph has no negative cycle if and only if its constraint graph has no negative cycles

Theorem: Let G be the augumented constraint graph of a consistent system of constraints $< V, C >$. Then D is a feasible solution for $< V, C >$, where $D(u) = dist_G(src, u)$

$dist_G(u, v)$ is the length of a shortest path from u to v in graph G

Shortest Path Algorithms

We can compute lengths of shortest paths from src to all vertices using Dijkstra's single source shortest path algorithm in $O(m + nlogn)$ time. The algorithm works only for positive edge weights in the graph

If there are negative edges we can use Bellman-Ford algorithm, which has a complexity of $O(mn)$

m is the number of constraints and n is the number of variables

Dijkstra's Algorithm

```
**function** shortestPath ($G$, $w$, $s$)
[$a$]   Initialize-Single-Source($G$,$s$)
[$b$]   $S := \emptyset$
[$c$]   $Q := V[G]$
[$d$]   **while** $Q \neq \emptyset$
[$e$]       $u :=$ Extract-Min(Q)
[$f$]       $S := S \cup \{u\}$
[$g$]       **for each** vertex $v \in Adj[u]$
[$h$]       **do** Relax(u,v,w)
```

```
**function** Relax ($u$,$v$,$w$)
[$a$]   **if** $d[v] > d[u] + w(u, v)$ **then**
[$b$]       $d[v] := d[u] + w(u, v)$
[$c$]       $pred[v] := u$
```

Problem

To maintain a feasible solution to a system of constraints as it undergoes changes

- addition or deletion of a constraint
- modification of a existing constraint
- addition or deletion of a variable

Variable addition or deletion

The addition or deletion of a variable can be handled easily.

- We update the constraint graph
- If adding a new variable: initialize its value to be zero

The system continues to be feasible

Such changes are processed in constant time

Constraint deletion or relaxation

Deletion of a constraint does not introduce infeasibility since the system becomes less constrained

Relaxation of a constraint corresponds to increase in length of the corresponding edge in the graph and it does not affect the solution

- deletion: remove edge from constraint graph
- relaxation: change length of the edge
- the values of variables remain the same

Such changes are processed in constant time

Constraint addition

Addition of a new constraint $v - u \leq b$ corresponds to addition of an edge from $u \rightarrow v$ of length b

This can affect the feasibility of a system

Tightening an existing constraint also may affect the system

We will look at an incremental algorithm for solving the system when a constraint is added

A basic algorithm

```
**basic_solver** $(< V, E, length > \ , \ v - u \leq b \ )$
        $constraint\_graph\ G =< V, E, length >$
        $new\ constraint\ to\ add:\ \ v - u \leq b$
{
            $E' = E \cup \{u \rightarrow v\}$
            $length(u \rightarrow v) = b$
            $G' =< V, E', length >$
            $\mathbf{return}\quad Bellman\_Ford(G', src)$
}
```

The basic algorithm uses Bellman_Ford algorithm and therefore it is $O(mn)$ complexity

Comments

Let G be the old graph and G' be the new graph obtained from G by adding a new edge

We know that G' is not feasible iff there exists negative cycles in G'

We also know that G is feasible and therefore has no negative cycles

Therefore, if G' has negative cycle then it must involve the new edge $u \rightarrow v$

Hence, the problem can be reduced to computing if $dist_G(v, u) + b < 0$

Comments

In general, the graph G will contain edges of negative length

Computing $dist_G(v, u)$ using the standard algorithm can take $O(mn)$

We can do better by using the feasible solution for the original set of constraints

Theorem: Let $G =< V, E, length >$, let f be a real valued function on V, the set of vertices. Define a new graph G_f, the graph scaled by f as follows: $G_f =< V, E, length_f >$, where $length_f(x, y) = f(x) + length(x \to y) - f(y)$. A path P from x to y is a shortest path in G iff it is a shortest path in G_f. Further, $dist_{G_f}(x, y) = f(x) + dist_G(x, y) - f(y)$

A Simple Algorithm

We can scale the original graph G by any feasible solution D.

The new length of an edge $x \rightarrow y$ will be $D(x) + length(x \rightarrow y) - D(y)$, and is non-negative

This implies that we can use Dijkstra's algorithm in O(m+nlogn) time

We can compute $dist(v,u)$ in O(m+nlogn) time and determine if the new system is feasible

$d_{G'}(src, x) =$
$min(d_G(src, x), d_G(src, u) + len(u \rightarrow v) + d_G(v, x))$

An Improved Algorithm

The simple algorithm does not fully utilize the original solution in computing a new one

Let $< V, C >$ denote original system of constraints, and D a feasible solution. Let $C' = C \cup \{v - u \leq c\}$, and

$D'(x) = min(D(x), D(u)+len(u \rightarrow v)+dist_G(v, x))$

Call vertex x is affected if $D'(x) \neq D(x)$

Theorem: $< V, C' >$ is feasible iff u is not affected

Theorem: If $< V, C' >$ is feasible then D' is feasible solution for $< V, C' >$

Computing D'

Theorem: Let x be the parent of y in some shortest path tree for v in the original graph. If x is unaffected, then y is also unaffected

Call an edge $x \rightarrow y$ affected iff vertex x is affected. Let H denote the subgraph of G consisting only of affected edges.
Thus, $H =< V, E_a, length|E_a >$

$$D'(x) = min(D(x), D(u) + len(u \rightarrow v) + dist_H(v, x))$$

Improved Algorithm

```
function AddToFeasible ($G$, $v - u \leq b$, $D$)
[$a$]   Add edge $u \rightarrow v$ to $E$
[$b$]   $len(u \rightarrow v) := b$
[$c$]   $D' := D$
[$d$]   $priorityQ := \emptyset$
[$e$]   insertHeap($priorityQ$,$v$,0)
[$f$]   while $priorityQ \neq \emptyset$ do
[$g$]     ($x$, $d_x$) := find&DeleteMin($priorityQ$)
[$h$]     if $D(u) + len(u \rightarrow v) + d_x < D(x)$ then
[$i$]       if $x = u$ then
[$j$]           remove edge $u \rightarrow v$ from $E$
[$k$]           return false
[$l$]       $D'(x) := D(u) + len(u \rightarrow v) + d_x$
[$m$]       for every vertex $y$ in $Succ(x)$ do
[$n$]           $sl := d_x + (D(x) + len(x \rightarrow y) - D(y))$
[$o$]           if $(sl < keyOf(priorityQ, y))$ then
[$p$]             adjustHeap($priorityQ$, $y$, $sl$)
[$q$]   $D := D'$
[$r$]   return true
```

Handling Infeasible Systems

So far, we modified feasible constraint systems and rejected any constraint that introduced infeasibility

Sometimes, it will be useful to allow addition of constraints that cause the system to become infeasible

They present *AddConstraint* and *DeleteConstraint* algorithms to add/delete constraints to such systems

The complexity of algorithms is $O(m + nlogn)$

Handling Infeasible Systems

The algorithm partitions the set of constraints into two sets: a feasible set and an unprocessed set

The feasible set is represented by a constraint graph and a feasible solution is maintained

The unprocessed consists of the remaining constraints that make the system infeasible

If the whole system is feasible then the unprocessed set is empty

Add Constraint

When a new constraint c is added to a feasible system, the algorithm checks if the system is still feasible

If so, c is added to feasible set and constraint graph extended else c is added to unprocessed set and the system becomes infeasible

If the system is infeasible then adding new constraints does not make it feasible, so they are added directly to set unprocessed

Delete Constraint

When a constraint c is deleted, the algorithm checks if c is in feasible set or unprocessed set

If c is in unprocessed set, it simply removes it from the set, if unprocessed becomes empty then we know the system is feasible

If c is from feasible set then removing it might make an infeasible system to become feasible

So when c is removed from the constraint graph, all the constraints in unprocessed are processed one by one till either all are added to the graph or one of constraints cannot be added

Extensions

The algorithm does not identify the smallest change necessary to the current solution to produce a feasible solution

One reason is that, the algorithm attempts to propagate the effects of the added constraint forward along the edges of the constraint graph.

It will be worth exploring the possibility of propagating the effects of the added constraint along both directions of the edges