MSArbor class

Version: 1.13
Date: August 8, 2009

Antonio Frangioni

Operations Research Group
Dipartimento di Informatica
Università di Pisa

Contents

0.1 Namespace Index 1
0.2 Class Index 1
0.3 File Index 1
0.4 Namespace Documentation 2
0.5 Class Documentation 2
0.6 File Documentation 5

0.1 Namespace Index

0.1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

MSA_di_unipi_it (The namespace MSA_di_unipi_it is defined to hold the MSArbor class) . . . 2

0.2 Class Index

0.2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

MSArbor (Given a (complete) directed graph G = (N , A), with n = |N| nodes and m = |A| (= n (n - 1)) arcs, and arc costs C[i , j], the Minimal Spanning Arborescence problem (MSA) with root r requires to find a directed spanning tree (arborescence) of G rooted in r with minimal total cost, where the cost is the sum of the costs of the arcs belonging to the tree) 2

0.3 File Index

0.3.1 File List

Here is a list of all documented files with brief descriptions:

MSArbor.h (Solves the Minimal Spanning Arborescence problem, with a C++ implementation of the ARBOR algorithm [M.Fischetti and P.Toth, ORSA J.on Computing 5(4), 1993] for MSA on complete graphs) 5

0.4 Namespace Documentation

0.4.1 MSA_di_unipi_it Namespace Reference

The namespace MSA_di_unipi_it is defined to hold the MSArbor class.

Classes

- class MSArbor

  Given a (complete) directed graph G = (N , A), with n = |N| nodes and m = |A| (= n (n - 1)) arcs, and arc costs C[i , j], the Minimal Spanning Arborescence problem (MSA) with root r requires to find a directed spanning tree (arborescence) of G rooted in r with minimal total cost, where the cost is the sum of the costs of the arcs belonging to the tree.

0.4.1.1 Detailed Description

The namespace MSA_di_unipi_it is defined to hold the MSArbor class.

0.5 Class Documentation

0.5.1 MSArbor Class Reference

Given a (complete) directed graph G = (N , A), with n = |N| nodes and m = |A| (= n (n - 1)) arcs, and arc costs C[i , j], the Minimal Spanning Arborescence problem (MSA) with root r requires to find a directed spanning tree (arborescence) of G rooted in r with minimal total cost, where the cost is the sum of the costs of the arcs belonging to the tree.

```
#include <MSArbor.h>
```

Public Types

- typedef unsigned short Index

  arc or node index (>= 0)

- typedef Index * Index_Set

  set (array) of indices

- typedef const Index cIndex

  a read-only Index

- typedef cIndex * cIndex_Set

  read-only array

- typedef short CNumber

  type of arc costs: since the cost matrix is re-used to store node names, it not should be (too) "smaller" than Index

- typedef CNumber * CRow

vector of costs

- typedef const CNumber cCNumber

  a read-only cost

- typedef cCNumber * cCRow

  read-only cost array

- typedef int FONumber

  type of objective function values; should be able to hold something like (max arc cost) times (max number of nodes)

Public Member Functions

- MSArbor (Index nds)

  Constructor of the class: takes as parameter the number of nodes in the graph G.

- FONumber Solve (cCRow C, CRow RC=0)

  This method solves the Minimum Spanning Arborescence problem on the complete graph described in the vector C[], which is a vector of arc costs, ordered as follows:.

- cIndex_Set ReadPred (void) const

  Returns a read-only vector describing the primal optimal solution of the problem, i.e., the "predecessor" function of the optimal MSA.

- cCRow MSArbor::GetU (void) const

  These three methods describe the dual optimal solution of the problem (a good knowledge of the algorithm is required for understending their meaning).

- Index GetN (void) const

  Returns the size of the problem (number of nodes).

Static Public Attributes

- static cIndex InINF = 65535

  the largest Index

- static cCNumber C_INF = 32767

  the largest arc cost is C_INF - 2, as C_INF - 1 means "no arc is here" and C_INF is reserved

0.5.1.1 Detailed Description

Given a (complete) directed graph G = (N , A), with n = |N| nodes and m = |A| (= n (n - 1)) arcs, and arc costs C[i , j], the Minimal Spanning Arborescence problem (MSA) with root r requires to find a directed spanning tree (arborescence) of G rooted in r with minimal total cost, where the cost is the sum of the costs of the arcs belonging to the tree.

This class solves MSA for r = n - 1 (the node with largest index) with a C++ implementation of the ARBOR algorithm [M. Fischetti and P.Toth, ORSA J. on Computing 5(4), 1993] for complete graphs.

Nodes and arc indices are defined to be of the public type MSArbor::Index, and arc costs are defined to be of the public type MSArbor::CNumber. These types, with the accompanying constants InINF and C_INF, are defined in the public types part of the class. By changing these definitions one can a) solve compilation problems due to different type or constant names in differend compilers, and b) use "small" data types, just large enough to fit the numbers in his/her instances, thereby (possibly) reducing the memory footprint of the object and increasing its efficiency.

0.5.1.2 Constructor & Destructor Documentation

MSArbor (Index *nds*)

Constructor of the class: takes as parameter the number of nodes in the graph G.

The actual graph is passed in Solve() [see below], and different instances can be solved by calling Solve() multiple times on the same MSArbor object, as long as al the instances have the same number of nodes.

0.5.1.3 Member Function Documentation

FONumber Solve (cCRow *C*, CRow *RC* = `0`)

This method solves the Minimum Spanning Arborescence problem on the complete graph described in the vector C[], which is a vector of arc costs, ordered as follows:.

- a sequence of n - 1 Backward Stars: BS[0], BS[1], .., BS[n - 2];
- arcs in each BS[i] are ordered in increasing index of tail node;

Since nodes are numbered from 0 to n - 1, and the root is node n - 1, C[i + n ∗ j] is the cost of (i , j). BS[root] does not exist.

Although the algorithm is developed with complete graphs in mind, it is possible to try solve non-complete instances by giving not-existent arcs the "plus infinity" cost C_INF - 1. If the costs of all other nodes are suitably smaller than that, then the optimal solution will avoid these arcs. However, if there is ∗no∗ feasible solution ∗not∗ using at least one arc with cost C_INF - 1, then the best possible unfeasible solution will be reported instead with no warning, so it is the user's responsibility to check feasibility if that is in doubt. Also, note that arcs should ∗never∗ be given cost C_INF, since that value is reserved for special use by the code.

If RC != 0 (= NULL), after that the Minimum Spanning Arborescence has been found its optimal arc reduced costs are written in RC, in the same format as the arc cost vector C.

Warning:

> the current implementation of the reduced cost computation is a "naive" one, which has an O(n^3) worst case complexity; a smarter O(n^2) implementation is possible, but it is not implemented as yet. Note that the MSA computation has an O(n^2) cost.

MSArbor::cIndex_Set ReadPred (void) const `[inline]`

Returns a read-only vector describing the primal optimal solution of the problem, i.e., the "predecessor" function of the optimal MSA.

For each node i = 0, ..., n - 2, j = ReadPred()[i] is the predecessor of i in the spanning tree, and therefore arc (i, j) belongs to the optimal solution. The root has no predecessor, hence the entry n - 1 of the returned vector should not be checked.

cCRow MSArbor::GetU (void) const `[inline]`

These three methods describe the dual optimal solution of the problem (a good knowledge of the algorithm is required for understending their meaning).

The dual optimal solution is described in terms of at most 2 n - 2 sets of nodes, arranged in a tree structure (called the "auxiliary tree") and with a dual variable attached to each. GetM() and ReadAux() describe the topology of the auxiliary tree: GetM() returns the number of its nodes, while ReadAux()[i] is the predecessor function of the tree. Note that the tree does not include a dummy root node, thus the sons of the dummy root have predecessor InINF. Finally, GetU()[i] is the value of the optimal dual variable associated with the set (node of the auxiliary tree) i.

MSArbor::Index GetN (void) const `[inline]`

Returns the size of the problem (number of nodes).

0.6 File Documentation

0.6.1 MSArbor.h File Reference

Solves the Minimal Spanning Arborescence problem, with a C++ implementation of the ARBOR algorithm [M.Fischetti and P.Toth, ORSA J.on Computing 5(4), 1993] for MSA on complete graphs.

Namespaces

- namespace MSA_di_unipi_it

Classes

- class MSArbor

  Given a (complete) directed graph G = (N , A), with n = |N| nodes and m = |A| (= n (n - 1)) arcs, and arc costs C[i , j], the Minimal Spanning Arborescence problem (MSA) with root r requires to find a directed spanning tree (arborescence) of G rooted in r with minimal total cost, where the cost is the sum of the costs of the arcs belonging to the tree.

0.6.1.1 Detailed Description

Solves the Minimal Spanning Arborescence problem, with a C++ implementation of the ARBOR algorithm [M.Fischetti and P.Toth, ORSA J.on Computing 5(4), 1993] for MSA on complete graphs.

Version:

1.13

Date:

01 - 09 - 2009

Author:

Antonio Frangioni
Operations Research Group
Dipartimento di Informatica
Universita' di Pisa
Daniele Pretolani
Dipartimento di Matematica
Universita' di Camerino
Marisa Traini
Dipartimento di Matematica
Universita' di Camerino