The `VerySimple01Problem` Class

Version: 1.01
Date: 04 / 05 / 2012

Antonio Frangioni

Operations Research Group
Dipartimento di Informatica
Università di Pisa

Giovanni Rinaldi

Istituto di Analisi di Sistemi e Informatica
Consiglio Nazionale delle Ricerche

Contents

0.1 The VerySimple01Problem Documentation

0.1.1 Introduction

This file is a short user manual for the distribution of VerySimple01Problem, a small class for solving the simplest problem in 0-1 variables. While this is hardly interesting, the class also "efficiently" enumerates any given subset of all the (2^n) solutions of the problem in objective function value order. This has a few possible applications, and it might be useful as a "blueprint" to someone in need to do the same (enumerate the solutions in objective function value order) for other combinatorial problems with appropriate structure, e.g. some path and matching problems. The ideas behind the approach are fairly extensively described in the interface.

0.1.1.1 Standard Disclaimer

This code is provided "as is", without any explicit or implicit warranty that it will properly behave or it will suit you needs. Although codes reaching the distribution phase have usually been extensively tested, we cannot guarantee that they are absolutely bug-free (who can?). Any use of the codes is at you own risk: in no case we could be considered liable for any damage or loss you would eventually suffer, either directly or indirectly, for having used this code. More details about the non-warranty attached to this code are available in the license description file.

The code also comes with a "good will only" support: feel free to contact us for any comments/critics/bug report/request help you may have, we will be happy to try to answer and help you. But we cannot spend much time solving your problems, just the time to read a couple of e-mails and send you fast suggestions if the problem is easily solvable. Apart from that, we can't offer you any support.

0.1.1.2 License

This code is provided free of charge under the "GNU Lesser General Public License", see the file doc/LGPL.txt.

0.1.1.3 Release

Current version is: 1.01

Current date is: May 04, 2012

0.1.2 Package description

This release comes out with the following files:

- **doc/LGPL.txt**: Description of the LGPL license.
- **doc/refman.pdf**: Pdf version of the manual.
- **doc/html/***: Html version of the manual.
- **VrySmplP/VrySmplP.h**: Contains the declaration of class `VerySimple01Problem`.
- **VrySmplP/VrySmplP.C**: Contains the implementation of the `VerySimple01Problem` class. You should not need to read it, unless you want to use it as a blueprint for implementing solution enumeration techniques for other combinatorial problems.
- **VrySmplP/makefile**: Makefile for the `VrySmplP` class.
- **Main/Main.C**: Contains an example of use of the VerySimple01Problem, which also works as a correctness tester.
- **Main/makefile**: A makefile for Main.C.

0.2 Namespace Index

0.2.1 Namespace List

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

0.3 Class Index

0.3.1 Class List

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

0.4 File Index

0.4.1 File List

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

0.5 Namespace Documentation

0.5.1 VS01P_di_unipi_it Namespace Reference

The namespace VS01_di_unipi_it is defined to hold the VerySimple01Problem class and all the relative stuff.

Classes

- class VerySimple01Problem

  This class solves the Very Simple Problem in 0/1 optimization (VS01P): given n 0/1 variables x[i] and n weights w[i], i = 0 ...

0.5.1.1 Detailed Description

The namespace VS01_di_unipi_it is defined to hold the VerySimple01Problem class and all the relative stuff. It comprises the namespace std.

0.6 Class Documentation

0.6.1 VerySimple01Problem Class Reference

This class solves the Very Simple Problem in 0/1 optimization (VS01P): given n 0/1 variables x[i] and n weights w[i], i = 0 ...

`#include <VrySmplP.h>`

Public Types

Public Types

- typedef double Weight
  Type of the weights w[i].
- typedef unsigned int Index
  Type of the indices "i" in "w[i]", "x[i]".
- typedef double ZeroOne
  Type of the variables x[i]: must be large enough to hold both values 0 and 1 (is there any type which can not?).

Public Member Functions

Constructor

- VerySimple01Problem (Index n)
  Constructor of the class.

Other initializations

- void SetWeights (const Weight *wght)
  Set a new vector of weights, effectively changing all the data of the VS01P.

Solving the problem

- void SolveVS01P (void)
  Solves the problem (not a big deal).

Reading the solution(s)

- Weight GetVal (void)
  Get the objective function value of the next best solution.
- void GetSol (ZeroOne *x)
  Get the next best solution.
- Index NSol (void) const
  Returns the number of solutions enumerated so far.

Reading the data of the problem

- Index **Getn** (void) const
- const Weight * **Getw** (void) const

Destructor

- ~VerySimple01Problem ()
  Destructor of the class.

0.6.1.1 Detailed Description

This class solves the Very Simple Problem in 0/1 optimization (VS01P): given n 0/1 variables x[i] and n weights w[i], i = 0 ...

n - 1, the problem is

$$\max\{\sum_{i=0}^{n-1} w[i]x[i] : x[i] \in \{0,1\} \qquad ,i = 0,...,n-1\}$$

The problem is so exceedingly trivial to solve (scan the variables in any order, assign 1 to the variable if the weight is $>$ 0, assign 0 if the weight is $<$ 0, do whatever if the weight is zero) that one may well argue against the need of a class for doing that. However, this class does more than solving the problem: it can efficiently produce all the solutions of the problem (the 2^n different strings of n bits) ordered by non increasing objective function value.

Of course, enumerating all these solutions can never be done efficiently since they are exponentially many. However, we are able to generate them one by one, and the effort to generate each new solution is very limited: generating the (k+1)-th solution only requires O(n) -- the bare minimum necessary to write it down. Also, only O(k) memory is required to generate k solutions (not counting the memory required to store the solution themselves, though).

This is obtained by iteratively constructing and visiting an enumeration tree, whose structure is somewhat different from those used e.g. in enumeration algorithms. The root of the tree is (one of) the optimal solution(s) of the problem, say x^*. Each node of the enumeration tree (not only the leaf nodes, as usual in enumeration trees) contains a solution of the problem. The sons of the root (imagine them pictured from left to right) are as follows:

- the first node is obtained by flipping variable x[0] (giving it the different value with respect to x^*[0]) and solving the VS01P on all the other variables considering that that variable is fixed, i.e., keeping all the other variables as they are in x^*;
- the second node is obtained by keeping x[0] as in x^*, flipping x[1] and solving the VS01P on all the other variables considering the first two variables as fixed, i.e., keeping all the other variables as they are in x^*;
- ...
- the n-th node is obtained by fixing all the first n - 1 variables as they are in x^* and flipping x[n - 1].

The tree is then constructed by recursively iterating the process on all the sons. In each node, the first k variables are fixed (x[k - 1] is the one that is flipped with respect to the father node). Then, the sons of that node are obtained, for i = k ... n - 1, by keeping x[0] ... x[i - 1] as in the father node, flipping x[i] w.r.t. the father node (and, therefore, w.r.t. x^*) and solving the VS01P on all the other variables considering the first i + 1 variables as fixed, i.e., keeping all the other variables as they are in x^*.

This tree is exponential in size (of course). However, if the variables are properly ordered it is possible to only explicitly construct and visit O(k) nodes of the tree to enumerate the first k solutions (in nonincreasing order of the objective function value) of VS01P. The good order of the variables is that for nondecreasing value of $|$ w[i] $|$; with this order, flipping a variable i produces "less damage" (decreases less the objective function value) than flipping a variable j with j $>$ i.

If the variables are ordered this way, then the solutions of the problem can be listed, in the desired order, as follows. The set of unvisited nodes of the tree from which the exploration must proceed, Q, is initialized with the root (containing the optimal solution x^*). Each time the next best solution is required, the best node (the one with largest objective function value of the associated solution) is extracted from Q and the associated solution is returned. Then, the first son of that node is added to Q. Also, if the node has a father (it is not the root) it surely is the rightmost son of his father inserted in Q as yet; then, its brother next to the right (if any) is also added to Q. No other nodes need to be added to Q, since all other sons, and all other brothers more to the right w.r.t. the immedaite one, surely contain solutions with worse (not better) objective function value w.r.t. the two generated ones.

This exploration strategy ensures that producing the k best solutions to the problem requires generating, and inserting in Q, at most 2k nodes of the enumeration tree. Since each node of can be represented with O(1) memory, the storage required to hold the representation of the fragment of the tree generated so far is O(k). Also, generating the (k + 1)-th solution requires O(lg k) for searching the best node in Q (using e.g. a binary heap), O(1) for generating the corresponding two new nodes and O(n) to write down the solution using the information stored in the current

fragment of the tree (climbing the tree from the node to the root gives the list of all and only variables than need to be flipped w.r.t. x^∗, and those variables are even ordered). Since k is no more than 2^n, the overall complexity of generating a new solution is O(n).

0.6.1.2 Member Typedef Documentation

0.6.1.2.1 typedef double Weight

Type of the weights w[i].

By changing this definition and recompiling the code works with whatever base type is chosen. It may have been set as a template, but it seemed overkill.

0.6.1.2.2 typedef unsigned int Index

Type of the indices "i" in "w[i]", "x[i]".

By changing this definition and recompiling the code works with whatever base type is chosen. It may have been set as a template, but it seemed overkill.

0.6.1.2.3 typedef double ZeroOne

Type of the variables x[i]: must be large enough to hold both values 0 and 1 (is there any type which can not?).

By changing this definition and recompiling the code works with whatever base type is chosen. It may have been set as a template, but it seemed overkill.

0.6.1.3 Constructor & Destructor Documentation

0.6.1.3.1 VerySimple01Problem (Index *n*)

Constructor of the class.

The parameter "n" is the number of variables in the VS01P.

0.6.1.3.2 ~VerySimple01Problem ( )

Destructor of the class.

0.6.1.4 Member Function Documentation

0.6.1.4.1 void SetWeights (const Weight ∗ *wght*)

Set a new vector of weights, effectively changing all the data of the VS01P.

The new weights are expected to be found in the first n positions of the vector wght. The class has the right to retain a pointer to this vector and keep using it until SetWeights() is called again, so the caller must not change or delete the vector until it is in use by the class. Calling again SetWeights() destroys any existing information about the previous set of k-optimal solutions, restarting the generating process anew.

The class does not change the vector wght (the pointer is read-only).

This method has to be called at least once if any method in the following sections is to be called.

0.6.1.4.2 void SolveVS01P (void)

Solves the problem (not a big deal).

This also re-initialize the process for generating the solutions, so the next solution obtained with GetSol() [see below] after a call to SolveVS01P() will always be (one of) the optimal one(s).

This method has to be called at least once every time the weights change [see SetWeights() above] is any solution for the corresponding VS01P is desired.

0.6.1.4.3 Weight GetVal (void)

Get the objective function value of the next best solution.

This method can be called only after SolveVS01P() [see above]. At the first call it returns the optimal objective function value of VS01P. At the subsequent calls it starts returning the objective function values of all the solutions to VS01P, in nonincreasing order. If more than one solution have the same value of the objective function, that value will be returned as many times as there are solutions.

Note

This method must not be called more than 2^n times, as there are "only" that many different solutions.

0.6.1.4.4 void GetSol (ZeroOne * *x*)

Get the next best solution.

This method and be called only after GetVal() [see above]. It returns the solution having the value of the objective function returned by the latest call to GetVal(). The solution is written in the first n positions of the vector x.

Note that the "pointer in the list of the solutions" is only "moved" by calls to GetVal(). That is, two subsequent calls to GetSol() with no calls to GetVal() in between will return the same solution. Analogously, calling k times GetVal() with no calls to GetSol() in between amounts to discarding k - 1 solutions, since only the solution corresponding to the last return value of GetVal() can then be retrieved.

0.6.1.4.5 VerySimple01Problem::Index NSol (void) const [inline]

Returns the number of solutions enumerated so far.

0.7 File Documentation

0.7.1 VrySmplP.h File Reference

Solves the Very Simple Problem in {0, 1} optimization, that is, the one without any constraint.

```
#include <exception>
```

Classes

- class VerySimple01Problem

  This class solves the Very Simple Problem in 0/1 optimization (VS01P): given n 0/1 variables x[i] and n weights w[i], i = 0 ...

Namespaces

- namespace VS01P_di_unipi_it

  The namespace VS01_di_unipi_it is defined to hold the VerySimple01Problem class and all the relative stuff.

0.7.1.1 Detailed Description

Solves the Very Simple Problem in {0, 1} optimization, that is, the one without any constraint. However, also implements an efficient procedure for listing all the solutions of the problem in nonincreasing (for a maximization problem) order of the objective function value.

Version

1.01

Date

14 - 12 - 2003

Author

Antonio Frangioni
Operations Research Group
Dipartimento di Informatica
Universita' di Pisa
Giovanni Rinaldi
Istituto di Analisi di Sistemi e Informatica
Consiglio Nazionale delle Ricerche