Searching for Generalized Coverings with Problem Dependant Optimization

Iliya Bluskov and Aaron Germuth

April 4th, 2016

This program searches for generalized coverings and or packings with a fixed block size. This application is a fork of Numela and Ostergaard's COVER. It is uses Problem Dependant Optimization (PDO) as a search strategy. Below you can find some notes on executing the program (cover.exe).

The current version of the program works best on Windows. In order to use it on UNIX environments, a couple of headers need to be changed in cover.h. The old UNIX headers for random number generation are still there, but commented out, and labelled as such. This program is executed by opening a command prompt in the executable's directory, and running it (by typing ./cover). This will begin the search.

Parameters:

The application can also take parameters at run time, in the same way as the previous fork. Parameters are specified after the name like so:

>> ./cover v=12 m=5 t=4 b=35

In this way as many parameters as you like can be specified. Each parameter begins with it's name (for example, v or m above), then an equals sign, and then its value. Many of the parameters are boolean in nature and therefore can only be set to 0 or 1. Most of the other options are integer valued, with some floating point parameters. The large majority of parameters have exact backwards compatibility with Numela and Ostergaard's COVER, but now work for PDO instead. There are also some added parameters. By default, on finding a solution, two output files are produced (.res and .log). The .res file prints out all the blocks of the found solution (conveniently in the same format which the program make take as input with startFromFile=1). The .log file contains some other information such as what parameters were used. The files are named in the following format "(v,k,m,t,lambda) - b.xxx". Below is a dictionary of most of the parameters currently accepted:

k,t,m,v,l

These variables are the parameters of the design. The default value for l is 1, and if only one of the values m and t are given it is assumed that m==t.

b

This variable is the number of k-sets (blocks) in the design.

TestCount

This variable determines, how many annealing runs for a given b are accomplished. Variable TestCount can be abbreviated TC. Default 1.

EndLimit

Sometimes you may want to stop the annealing even before all sets are covered. You can give an limit for an acceptable solution with this variable. Abbreviation EL.

OntheFly

Set this variable to 1, if you want the program not to allocate all the tables and to compute some of the needed data on-the-fly. Abbreviation OF.

Pack

You can search for packing desings by setting this variable to 1. Abbreviation P.

pdoK

A parameter of the PDO search algorithm. Each value of the jump down function is normally set to pdoK * neighbourhoodLength. pdoK has a default value of 10.

pdoJ

A parameter of the PDO search algorithm. Once at a solution of cost k, we use the jump down function to asses how long we should try to find a solution of matching or better cost. The amount of times we try is pdoJ * jumpDownFunction(k). The default value is 2.

pdoPrint

This can be set to three levels to control how the program prints progress updates. At 0, the program prints nothing while searching. At 1, the program prints updates of the jump down function and count of the bottom 10 costs. At 2, the program behaves similarly to 1, but prints only the 10 lowest costs which have actually been visited. This way, if your program is searching

at costs 50-60, you will see still be seeing updates, rather than just statically looking at costs 1-10. The default value is 3.

pdoPrintFreq

This controls how often to print progress updates. When set to 1, every time the values are updated, they are printed. The only problem with this approach is that massive amounts of output to a command prompt often lead to performance issues. Therefore, the default value is 50.

pdoMaxJDF

This controls how high the jump down function values can be. Currently the values returned are maxed at 5 million. It must be maxed at some value since the data types for integers have limited space available.

startFromFile

This allows you to start looking from a given solution. It reads the solution from a text file called startFile.txt. See the Optimization heading below.

greedyStart

This uses a slightly more greedy approach when first constructed a solution. Normally, it simply grabs b random blocks. In this case it makes sure each block covers at least one uncovered m-set.

MemoryLimit

This allows you to set a manual limit for how much memory the program may use.

Optimization:

It is possible to use this program as an optimizer, or verifier. You may take an existing solution and have the program start from that point with the option (startFromFile=1). Make sure your solution file is called “startFile.txt”, located in the same directory as the binary executable, and has blocks in the format [0,(k-1)]. Each point in a block is seperated by spaces, and each block is on it’s own line. See below for an example:

“startFile.txt” for (5,3,2,2,1) covering in 4 blocks

0 1 2
0 2 3

0 2 4
1 3 4

Memory Allocations:

Cover normally (when set to OnTheFly=0) precomputes the entire table of neighbourhood and covering data, in order to make the search more efficient. This often takes a large amount of memory, and you may receive an error for trying to use too much. The datatypes of many of the parameters can be increased in cover.h at compile time. There are also many run-time options associated with memory (OnTheFly, MemoryLimit). As of now, the program may output:

- "Binomial coefficient overflow"
  - Overflow of binCoefType when computing (v choose k)
- "Internal overflow"
  - Overflow of an internal type.
- "Invalid parameters"
  - If one of the following is true (t > k || t > m || k > v || m > v || b <= 0 || t <= 0)
- "Parameter v too large, not enough space reserved"
  - The maxV parameter must be reduced. (Declared in cover.h)
- "Memory allocation error"
  - Operating System does not have enough memory the amount requested. Remove other programs running, or decrease parameters.
- "Parameter b is larger than maxkSetCount"
  - Too many blocks in the design. Either decrease b, or increase the size of coveredType
- "Space demands exceed the limit given by MemoryLimit option"
  - MemoryLimit is a parameter you can set at runtime, it must be increased
- "RankType is too small to contain the binomial coefficients needed."
  - Overflow of rankType when computing the rank of a block