

# Contents

## I: Introduction

## II: Notation and Conventions

## III: Getting Started

## IV: Examples: using the student bundle package and interfacing with Matlab(R)

## V: Modifying bundle

## VI: A Note on the make command

## I: Introduction

This guide has information for two different user groups: students who want to use the software in a project and administrators who want to modify or upgrade the software. Both students and administrators should read this introduction and section II: Notation and Conventions since these sections outline what the software package does and how this document is formatted. Beyond these two sections, students need only read the sections III: Getting Started and IV: Examples: using the student bundle package and interfacing with Matlab(R). III: Getting Started explains how to install and run the software package. IV: Examples: using the student bundle package and interfacing with Matlab(R) is the most important section for the student, since it walks the student through the features of the software and instructive Matlab (R) examples. This section includes numerous examples that can be pasted right into the terminal and Matlab (R). The remaining two sections are for administrators and curious students. These final sections require an intermediate understanding of how the Linux operating system works and how a program is constructed in C or C++. V: Modifying bundle describes how to begin modifying the **bundle** and where to find additional information as well as listing the most important changes made while upgrading **bundle**. This information should aid future maintenance. Finally, VI: A Note on the make command explains what the **make** command does and some features in the supplied makefile.

The student bundle package contains a bundle optimization program (the program **bundle**) and a conversion program for converting set problems to various formats (the program **convscp**). The bundle part of the package is a modification of the original sparse bundle package by Reine Säljö, described in his Master's thesis<sup>i</sup>. The **convscp** part of the package was written by Douglas Potter in conjunction with updating the SCP project for the course TMA521/MMA510 Optimization, project course, at Chalmers, University of Technology and University of Gothenburg.

The **bundle** program loads a SCP (set covering problem) from a file which has, what will be referred to as, the standard format. Then **bundle** constructs the Lagrangian dual of the SCP and uses a bundle dual ascent algorithm to solve the Lagrangian dual problem to near optimality (at least for many problems). Many test set covering problems are in, what is referred to as, the rail format, and thus **bundle** cannot solve them until they have been converted to standard format by **convscp**. The program **bundle**, in most cases, outputs a near optimal dual vector (in file: 'uVector'), a near optimal relaxed-primal vector (in file: 'xVector') and a dual objective value (in file: 'objValue'). However the near optimal relaxed-primal vector usually does not constitute a solution to the SCP (the primal problem) since the solution vector is often fractional. However, 'uVector', 'xVector' and 'objValue' can be used to construct a good feasible solution via many heuristics. Such heuristics can be implemented in various computational environments such as MATLAB. One inconvenience is that MATLAB takes several minutes to load even small SCP's. Fortunately, **convscp** can convert a rail or standard formatted SCP to a MATLAB(R) .mat file. The .mat file is a compressed sparse representation of the SCP (plus a file header) which MATLAB can load in seconds.

## II: Notation and Conventions

“Set covering problem” is abbreviated as SCP. Single quotes (i.e. 'single quotes') denote a variable, a file or a directory. Double quotes (i.e. “Double quotes”) are reserved for quoting and citing published works. Boldface usually denotes a command or a program along with options that can be executed verbatim from the shell or within Matlab(R). Boldface can also denote an entry or command in a script or makefile. On a general level, **bundle** refers to the “Object Bundle Package - an OOP version of a Bundle-type algorithm for NonDifferentiable Convex Constrained Optimization” authored by Antonio Frangioni<sup>iii</sup>. However, specifically within the context of the student bundle package contained in 'student sparse.zip', **bundle** refers to the “Object Bundle Package” configured for maximizing the Lagrangian dual of a SCP with a sparse cover matrix.

## III: Getting Started

Although the executables, **bundle** and **convscp**, are not included, the **make** command can be used to create executables.

In order to build **bundle** and **convscp** from the terminal follow the subsequent instructions:

- 1.)unzip or unarchive sparse directory from 'student sparse.zip'
- 2.)in the terminal navigate to the sparse directory
- 3.)confirm the existence of the directories:  
'obj' and 'scr'  
and the files:  
'Documentation for the SCP project.pdf', 'Makefile', 'Parameters', 'academic license for Bundle.txt',  
'Manual for Bundle.txt','bundle', 'example1.m', 'example2.m', 'example3.m', 'openscpslow.m', 'scp41.txt',  
'scp61.txt' and 'rail507'
- 4.)build convscp and bundle by running:  
**make**
- 5.)set the path to the CPLEX license file for bundle by running:  
**export ILOG\_LICENSE\_FILE=/chalmers/sw/sup/cplex-10.1/ilm/access.ilm**
- 6.)set the path to MATLAB by running:  
**export MATLAB=/chalmers/sw/sup/matlab-2006b**
- 7.)set the library path to MATLAB for **convscp** by running:  
**export LD\_LIBRARY\_PATH=\$MATLAB/bin/sol2:\$MATLAB/sys/os/sol2:  
\$MATLAB/bin/glnx86:\$MATLAB/sys/os/glnx86**  
(This should be input as one line.)

Both **bundle** and **convscp** are now ready to be run by typing:.

**./bundle** filename

**./convscp** options

Note: executing **bundle** and **convscp** in a new terminal requires only that steps 5-7 be repeated, since the steps preceding steps 5-7 created **bundle** and **convscp**. The syntax of bundle is always **./bundle** followed by the file name of a scp problem in standard format. The syntax of convscp is more complicated. Executing **./convscp --help** displays its documentation.

## IV: Examples: using the student bundle package and interfacing with Matlab(R)

This section walks the reader through three examples. The purpose of these examples is not to provide the best way of doing things. All the running times and results of the MATLAB examples can be greatly improved. In fact, the examples originate from simplifications of strategies that lost the SCP competition in 2007 (a part of the Optimization Project Course at Chalmers University of Technology). The intention is to show how to use **convscp** and **bundle** and to provide a simple framework for connecting **bundle** to MATLAB—in short, to get pesky computer problems out of the way and let the user focus on mathematics and optimization. The first example walks through running **bundle** on a standard formatted SCP and a rail formatted SCP. The second example shows how to interface MATLAB and bundle. The third example demonstrates a few useful MATLAB commands for working with sparse matrices and some basic ideas concerning SCPs. The descriptions of examples two and three is limited since the MATLAB code is well-commented.

### Example 1: executing bundle from the terminal or shell

The first part of example 1 shows how to use **bundle** on a small SCP with 200 rows and 1000 columns. This SCP is in file 'scp41.txt' provide in 'student sparse.zip' and is in the standard format. This SCP and other test SCP's are available from OR-Library maintained by J E Beasley<sup>iii</sup>. Here is how you run bundle on this SCP:

1. In the terminal navigate the directory where "student sparse.zip" was unarchived
2. Set the environment variables by executing:  
**export ILOG\_LICENSE\_FILE=/chalmers/sw/sup/cplex-10.1/ilm/access.ilm**
3. Run bundle on the scp41.txt by executing:  
**./bundle scp41.txt**

As long as the 'parameters' file has not be modified, **bundle** should output the iteration log followed by:

```
-----  
-- Solution: --
```

```
-----  
f = 429  
u-vector written to file: uVector  
x-vector written to file: xVector  
objective value written to file: objValue  
-----
```

If you did not receive this output, check to make sure you are running the BASH shell and try decompressing the "student sparse.zip" again. You can enter the BASH shell from another shell by executing **bash**. Now the files 'uVector', 'xVector' and 'objValue' can be opened in a standard editor and examined. In order to understand iteration log, examine the source code (a good place to start is 'Oracle.C' found in the 'src' subdirectory) and the documentation of ILOG CPLEX 10<sup>iv</sup>.

Note: 'xVector' contains fractional elements since it is the solution to a relaxation of the SCP.

Now we turn to a larger SCP, 'rail507', which originates from a real world competition involving the Italian railways<sup>v</sup>. The program **bundle** cannot process 'rail507' since 'rail507' is in the rail format. We convert the 'rail507' to the standard format using **convscp**:

1. in the terminal, navigate the directory where "student sparse.zip" was unarchived
2. set the environment variables by executing the following two commands:  
**export MATLAB=/chalmers/sw/sup/matlab-2006b**  
**export**  
**LD\_LIBRARY\_PATH=\$MATLAB/bin/sol2:\$MATLAB/sys/os/sol2:\$MATLAB/bin/glnx86:\$MATLAB/sys/os/glnx86** (note: lines two and three should be entered as one command without a line break)
3. convert 'rail507' by executing:  
**./convscp -ir rail507 -os rail507.scp**
4. to understand what the command-line arguments did, execute:  
**./convscp -help**

The file 'rail507.scp' created by the command **./convscp -ir rail507 -os rail507.scp** contains the same SCP found in 'rail507' in the standard format. If you did not obtain this file, see the trouble shooting advice given for the 'scp41.txt' in the first part of this example. Now **bundle** can process the SCP. Before executing the following commands to do this, note that this will take a couple of minutes with the default configuration.

1. in the terminal navigate the directory where "student sparse.zip" was unarchived
2. set the environment variables by executing:  
**export ILOG\_LICENSE\_FILE=/chalmers/sw/sup/cplex-10.1/ilm/access.ilm**
3. run bundle on the scp41.txt by executing:  
**./bundle rail507.scp**

The files 'uVector', 'xVector' and 'objValue' contain the solution vectors found by **bundle**. If data from another SCP was in these files, it has been overwritten. While this situation is bothersome, it can be resolved with a BASH script or even from Matlab(R). The second example provides Matlab(R) code that takes care of this file bookkeeping. Scrolling back in the terminal, you can see the progress of **bundle** ascending the dual problem. The primal problem, the SCP, is a minimization problem so accordingly the dual problem is a maximization problem. Even though **bundle** usually has several ascent vectors at its disposal, every iteration does not result in an increase of the dual object value. This is because the local information available is not sufficient for determining how far to move in a given ascent direction. For more information see the chapter on "Unconstrained Optimization" in [An Introduction to Continuous Optimization](#)<sup>vi</sup>.

#### Example 2: using **bundle** in Matlab(R)

This example is a guide through some of the useful features of **convscp** and Matlab(R). After completing example 1, we know how to obtain the solution vectors of the Lagrangian dual problem. Efficiently using these vectors to generate a good primal solution is our goal. The first step of this is loading the scp problem in Matlab(R) and second step is loading the vectors from **bundle**. The Matlab(R) code found in the function 'openscpslow' can open SCP's. In the Matlab(R) environment, navigate to the directory where 'student sparse.zip' was unarchived. Then to load the cost vector and the cover matrix of the SCP in 'scp41.txt', run

1.) **[cover,cost] = openscpslow('scp41.txt');**

Although this function, **openscpslow**, only takes a few seconds to load 'scp41.txt' or 'scp41.txt', the loading time becomes unacceptable for even a medium sized SCP found in 'rail507.scp' (created example 1). This inconvenience is circumvented by converting the SCP to a Matlab(R)'s .mat file format using **convscp**. From the terminal, **convscp** convert 'rail507.scp' to .mat formatted file by running:

1.) in the terminal, navigate the directory where "student sparse.zip" was unarchived

2.) set the environment variables by executing the following two commands:

**export MATLAB=/chalmers/sw/sup/matlab-2006b**

**export**

**LD\_LIBRARY\_PATH=\$MATLAB/bin/sol2:\$MATLAB/sys/os/sol2:\$MATLAB/bin/glnx86:\$MATLAB/sys/os/glnx86** (note: lines two and three should be entered as one command without a line break)

3.)run:

**./convscp -is rail507.scp -om rail507.mat**

Alternatively, if example 1 was not completed, the conversion can be performed on the original file 'rail507' by replacing step 3 above with:

3.)run:

**./convscp -ir rail507 -om rail507.mat**

Now in Matlab(R), the SCP in 'rail507' can be loaded by issuing this command:

1.) **load rail507.mat**

Issuing the above command, loads the cost vector into a variable called 'cost' and load the cover matrix into a variable called 'cover'.

If you want to accomplish the conversion from within Matlab(R), we can set the environment variables and run **convscp** with following commands:

1.) **setenv('ILOG\_LICENSE\_FILE', '/chalmers/sw/sup/cplex-10.1/ilm/access.ilm');**

2.) **setenv('MATLAB','/chalmers/sw/sup/matlab-2006b');**

3.) **setenv('LD\_LIBRARY\_PATH','\$MATLAB/bin/sol2:\$MATLAB/sys/os/sol2:\$MATLAB/bin/glnx86:\$MATLAB/sys/os/glnx86');**

(This must be entered as one command on one line.)

4.) **!./convscp -ir rail507 -om rail507.mat**

Note: the first command is not really necessary, but it does make it possible to run **bundle** from within Matlab(R) by running: **!./bundle**

The second part of this example is loading the output files('uVector', 'xVector' and 'objValue') from **bundle** in Matlab(R). The Matlab(R) file 'example1.m' contains commented code that runs **bundle** on a SCP, loads the output from bundle, loads the SCP and finds upper and lower bounds for the optimal solution to the SCP. Consequently, it is highly recommend that you examine 'example1.m'. Moreover, 'example1.m' can automatically save the the output from **bundle** to uniquely named files and reload it for later usage. The prerequisite for running 'example1.m' is that the SCP is in both in the standard format ('scp41.txt') and in .mat Matlab(R) format ('scp41.mat'). The first part of this example covers converting SCP files to the .mat format. Here is how 'example1.m' is run on 'scp41.txt' from Matlab(R):

1.) **[upper, lower] = example1('scp41firstattempt','scp41.txt','scp41.mat',1);**

This should produce output similar to:

```
time(sec) running bundle: 0.06
time(sec) loading data: 0.01
time(sec) running the heuristic: 0.02
upper bound: 2884    lower bound: 428.9999
```

The inputs to 'example1.m' are respectively: a file handle for saving intermediary data ('scp41firstattempt'), a file name of SCP in standard format ('scp41.txt'), a file name of the SCP in .mat format ('scp41.mat') and a flag specifying if bundle should be run (1 = run bundle, 0 = reuse existing bundle output files saved using the file handle argument). The outputs from 'example1.m'--the upper and lower bounds--are saved in the variables 'upper' and 'lower'. The output files from **bundle** are saved to files using the file handle as a naming convention. If an error was received, the most common cause is that '**example1.m**' cannot find either the standard formatted file ('scp41.txt') or the .mat formatted file ('scp41.mat'). If the standard formatted file cannot be found, Matlab(R) is probably pointing the wrong the directory. If Matlab(R) does not find the .mat formatted file, either Matlab(R) is set to the wrong directory or the .mat file has not been created. The existence of the .mat file can be resolved by issuing the following command from the terminal in the directory where scp41.txt resides:

1.) **./convscp -is scp41.txt -om scp41.mat**

If we were to modified the part of 'example1.m' that computes the upper bound, we would want to run this updated code on the output from **bundle** on all the SCP's we had previously tested. However, since **bundle** can take hours to run, we would like to avoid producing the output from **bundle** again. Advantageously, 'example1.m' saves its intermediary output using the file handle. Thus, we can run 'example1.m' again reusing previously generated **bundle** output by running:

1.) **[upper, lower] = example1('scp41firstattempt','scp41.txt','scp41.mat',0);**

This should generate output similar to:

```
cpu time(sec) loading data: 0.01
cpu time(sec) running the heuristic: 0.03
upper bound: 2884    lower bound: 428.9999
```

The utility of this feature becomes clear when run on 'rail507' (note: the time spent running **bundle**):

```
1.) [upper, lower] = example1('rail507','rail507.scp','rail507.mat',1);
```

This should yield output similar to:

```
time(sec) running bundle: 1.17  
time(sec) loading data: 0.22  
time(sec) running the heuristic: 156.53  
upper bound: 122399 lower bound: 156.7175
```

### Example 3: tackling the competition in Matlab(R)

Examples 1 and 2 should have brought you up to speed on using **bundle**, **convscp** and loading and managing the output from **bundle** in Matlab(R). The upper bound on the objective value of the SCP was produced using 'example1.m' by simply adding columns to the solution (by setting the corresponding primal variables to one) until feasibility was obtained (a feasible solution is an upper bound) and the dual solution was used as the lower bound. However, these bounds were awful. The files 'example2.m' and 'example3.m' utilize a few simple techniques for improving the upper bound. 'example2.m' introduces the idea of using the number of rows a column covers combined with the dual solution in deciding which columns should be chosen. Note: keeping an already chosen column from being chosen again, is achieved by setting its cost to the maximum. While this works, chosen columns could instead be removed from the cover matrix, reducing the size of the problem. Also if 'example1.m' has already been run on some SCP's, 'example2.m' can be run with the 0 option to save time. For 'scp41.txt' and 'rail507', 'example2.m' produces:

run:

```
[upper, lower] = example2('scp41firstattempt','scp41.txt','scp41.mat',0);
```

results:

```
cpu time(sec) loading data: 0.02  
cpu time(sec) running the heuristic: 0.01  
upper bound: 438 lower bound: 428.9999
```

run:

```
[upper, lower] = example2('rail507','rail507.scp','rail507.mat',0);
```

results:

```
cpu time(sec) loading data: 0.21  
cpu time(sec) running the heuristic: 92.01  
upper bound: 70817 lower bound: 156.7175
```

The bounds and the running time for the heuristic have improved significantly for both problems, but for 'rail507' we are still a long way from an acceptable result. One limitation of 'example2.m' is that how much columns overlap is not considered. Thus, if two 'low cost' (according to the logic of 'example2.m') columns that cover essentially the same rows, they will probably both be added to the solution. Instead, optimal only one of them should be added along with some columns that cover the difference between the two similar columns. This issue is only partially resolved in 'example3.m' by periodically recalculating the costs according to how many rows are still uncovered and setting the cost of the columns that no longer cover any uncovered rows to the maximum cost. It is time consuming to update the costs so a parameter "freq" can be set to tune how often this occurs (sometimes, we need a decent solution before a deadline). Here are some results for different cost update frequencies for 'scp41.txt' and 'rail507':

'scp41.txt' with freq=10 (modify 'example3.m') and run:

**[upper, lower] = example3('scp41firstattempt','scp41.txt','scp41.mat',0);**

results:

cpu time(sec) loading data: 0.01

cpu time(sec) running the heuristic: 0.01

upper bound: 463 lower bound: 428.9999

'scp41.txt' with freq=5 (modify 'example3.m') and run:

**[upper, lower] = example3('scp41firstattempt','scp41.txt','scp41.mat',0);**

results:

cpu time(sec) loading data: 0.01

cpu time(sec) running the heuristic: 0.01

upper bound: 438 lower bound: 428.9999

'scp41.txt' with freq=1 (modify 'example3.m') and run:

**[upper, lower] = example3('scp41firstattempt','scp41.txt','scp41.mat',0);**

results:

cpu time(sec) loading data: 0.01

cpu time(sec) running the heuristic: 0.1

upper bound: 431 lower bound: 428.9999

'rail507' with freq=100 (modify 'example3.m') and run:

**[upper, lower] = example3('rail507','rail507.scf','rail507.mat',0);**

results:

cpu time(sec) loading data: 0.25

cpu time(sec) running the heuristic: 1.34

upper bound: 666 lower bound: 156.7175

'rail507' with freq=10 (modify 'example3.m') and run:

**[upper, lower] = example3('rail507','rail507.scf','rail507.mat',0);**

results:

cpu time(sec) loading data: 0.24

cpu time(sec) running the heuristic: 2.72

upper bound: 376 lower bound: 156.7175

'rail507' with freq=1 (modify 'example3.m') and run:

**[upper, lower] = example3('rail507','rail507.scf','rail507.mat',0);**

results:

cpu time(sec) loading data: 0.22

cpu time(sec) running the heuristic: 16.33

upper bound: 227 lower bound: 156.7175

The results indicate the upper bound, and thus the solution, can be significantly improved at the cost at the expense of longer running times. Interestingly the method in 'example3.m' does not outperform 'example2.m' for 'scp41.txt' until freq is set to 1. This indicates, as noted above, that this technique, only partially resolves the problem of column overlap or entanglement. Note: the dual vector has not been used and we may have produced an over over (a feasible solution which remains feasible even if one or more columns are removed).



## V: Modifying bundle

If you intend on modifying the **bundle** C/C++ code or the Parameters file, it is highly recommended that you read Säljö's Master's thesis vii and 'Manual for Bundle.txt' by Antonio Frangioni **Error! Bookmark not defined.** found in 'student sparse.zip'. The original sparse bundle package was configured to run on Solaris with ILOG CPLEX(R) 9 in the TCSH shell and consequently it will not build on the current Chalmers systems as of 2007. The current version is designed to work with Linux with ILOG CPLEX(R) 10 in BASH (Bourne Again SHell). (Most students at Chalmers have BASH as their default shell when they open a terminal on a Linux computer.) Furthermore several small changes have been made to **bundle** including the precision of the output vectors, since with thousands of vectors being added and compared, the precision default is not always adequate. In order to rebuild **bundle** on another system, it will probably be necessary to recompile all the source files. This means recompiling the files included in 'student sparse.zip' as well as recompiling several non-distributable files that are not included in the student distribution 'student sparse.zip'. These non-distributable files are the files that correspond to the rows that have been commented out in the bundle section of the 'Makefile' included in 'student sparse.zip'. Once these files have been obtained and added to the source code directory 'src' and the corresponding lines uncommented, **make realclean** can be run to remove all the obsolete objective files and then **make** can be run to build new object files and the executables (see the **VI: A Note on the make command** ). After this is done the rows corresponding to the non-distributable files can once again be commented and then non-distributable files should be removed from the 'src' directory before distributing the package.

The operation of **bundle** and **convscp** can be modified by changing the source files provide in the 'scr' directory. It should be noted that not all the source files are included due to distribution restrictions. This does not pose a problem since the necessary object files are included.

To conclude this section, the minimum modifications required for the original bundle package under the current computer environment (as of February 2008) are listed. Specifically, the modifications are listed for **bundle** as configured in 'spar\_bundle.tar'. These modifications, with slight adjustments, will work on any of the old **bundle** examples and configurations. These changes only suffice to get **bundle** working and do not add all features provided in 'student sparse.zip'. When editing the 'Makefile' take special care not to delete or add extra tabs at the beginning of any line

1.) Make sure your shell is BASH. If not, run in the terminal:

**bash**

2.) uncompress 'spar\_bundle.tar'

3.) uncompress 'hidden.tar' to the same directory where 'spar\_bundle.tar' was unarchived

(It is important the files from 'spar\_bundle.tar' and 'hidden.tar' are in the same directory and not just that directories 'bundle' and 'hidden' are the same directory. If the latter is the case, merge these directories.)

4.) open the 'Makefile' in a text editor and make the following 11 changes:

1. change: **setenv ILOG\_LICENSE\_FILE /usr/site/pkg/cplex-8.1/ilm/access.ilm**  
to: **export ILOG\_LICENSE\_FILE=/chalmers/sw/sup/cplex-10.1/ilm/access.ilm**
2. change: **SYSTEM = ultrasparc\_5\_5.0**  
to: **SYSTEM = x86\_rhel4.0\_3.4**
3. under: **SYSTEM = x86\_rhel4.0\_3.4**  
add: **SHELL = /bin/bash**

4. change: **LIBFORMAT = static\_pic\_mt**  
to: **LIBFORMAT = static\_pic**
5. change: **CPLEXDIR = /usr/site/pkg/cplex-8.1/cplex81**  
to: **CPLEXDIR = /chalmers/sw/sup/cplex-10.1/cplex101**
6. change: **CONCERTDIR = /usr/site/pkg/cplex-8.1/concert13**  
to: **CONCERTDIR = /chalmers/sw/sup/cplex-10.1/concert23**
7. change: **CCC = CC**  
to: **CCC = g++**
8. change: **CCOPT = -O -xtarget=ultra -xarch=v8plus -DNDEBUG -pto -KPIC-DIL\_STD**  
to: **CCOPT = -O1 -DNDEBUG -fPIC -DIL\_STD -Wno-deprecated**
9. change: **CCLNFLAGS = -L\$(CPLEXLIBDIR) -lilocplex -lcplex -L\$(CONCERTLIBDIR) \ -lconcert -mt -lm -lsocket -lnsl**  
to: **CCLNFLAGS = -L\$(CPLEXLIBDIR) -lilocplex -lcplex \ -L\$(CONCERTLIBDIR) -lconcert -m32 -lm -lpthread**
10. change: **rm -f \*~ Main.o Solver.o Oracle.o bundle**  
to: **rm -f \*~ \*.o bundle**
11. under: **Oracle2.o: Oracle2.C**  
**\$(CCC) -c \$(CCFLAGS) Oracle2.C -o Oracle2.o**  
add: **Bundle.o: Bundle.C**  
**\$(CCC) -c \$(CCFLAGS) Bundle.C -o Bundle.o**  
**Quad.o: BMinQuad.C**  
**\$(CCC) -c \$(CCFLAGS) BMinQuad.C -o BminQuad.o**  
**CMinQuad.o: CminQuad.C**  
**\$(CCC) -c \$(CCFLAGS) CMinQuad.C -o CMinQuad.o**  
**QPBund.o: QPBund.C**  
**\$(CCC) -c \$(CCFLAGS) QPBund.C -o QPBund.o**  
(It is crucial on every other line begins with one tab and no a series of spaces)

5.) Open 'OPTtypes.h' in a text editor and make the following one change:

1. change: **///**#include <values.h>****
- to: ****#include <values.h>****

6.) navigate in the terminal to directory where 'spar\_bundle.tar' was unarchived

7.) to build **bundle** run:  
**make**

8.) now **bundle** can be run by typing **./bundle** followed by a SCP file. For example if **scp41.txt** was in the local directory, we could run:  
**./bundle scp41.txt**

## VI: A Note on the make command

The **make** command follows the instructions of the 'Makefile' in the current directory, automating the compiling and linking of the executables. Thus in practice, a 'Makefile' is a specialized script for controlling and automating compiling and linking. The syntax is quite arcane and dates from the 1980s, but **make** is available on most systems and does the job. For example, five space and tab have different meanings, so it is necessary to be quite careful when editing a makefile. The GNU Project's website provides a good reference for **make**. One example of what **make** does can be seen by running **make** after changing one of the source files and observing that make automatically detects this and re-builds the affected targets (usually executables and object files). The 'Makefile' in the sparse directory provides additional functionality which is illustrated by the following examples:

**make** - builds all the targets, in this case both **bundle** and **convscp**

**make clean** - remove the executables and the object files that the student version created

**make bundle** - only build **bundle**

**make convscp** - only build **convscp**

**make realclean** - remove the executables and all the object files

note: this last example should NOT be run unless all source files are present in the 'scr' directory

- 
- i      Reine Säljö  
         Implementing of a bundle algorithm for convex optimization (2004)  
         Chalmers University of Technology, Göteborg, Sweden  
         <http://www.math.chalmers.se/Math/Grundutb/CTH/tma521/0607/Thesis.pdf>
  
  - ii     Antonio Frangioni  
         <http://www.di.unipi.it/~frangio>  
         <http://sorsa.unica.it/it/software.php>
  
  - iii    OR-Library  
         J E Beasley  
         <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html>
  
  - iv     ILOG CPLEX 10.0 User's Manual  
         <http://www.lix.polytechnique.fr/~liberti/teaching/xct/cplex/usrcplex.pdf>
  
  - v      Alberto Caprara, Matteo Fischetti, Paolo Toth  
         A Heuristic Method for the Set Covering Problem (1995)  
         <http://citeseer.ist.psu.edu/cache/papers/cs/3135/http:zSzzSzpromet4.deis.unibo.it/zSz~albertozSzscp.pdf/caprara95heuristic.pdf>
  
  - vi     Niclas Andréasson, Anton Evgrafov and Michael Patriksson (2005)  
         An Introduction to Continuous Optimization  
         Studentlitteratur, Lund, Sweden
  
  - vii    GNU 'make'  
         <http://www.gnu.org/software/make/manual/make.html>