Technical Report
Concorde solver installation and use

Túlio Neme de Azevedo - tulioneme10@gmail.com

Marco Antonio Moreira de Carvalho - marco.opt@gmail.com

Departamento de Computação

Universidade Federal de Ouro Preto, MG, Brazil

12 de abril de 2018

Introduction

Concorde[1] is a solver for the classic Traveling Salesman Problem (TSP) and some related optimization problems. Developed in C language, this solver has reported the best benchmark results so far, solving all instances available at *TSPLIB*[2].

[1] `http://www.math.uwaterloo.ca/tsp/concorde`

[2] `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`

This technical report presents how to install and use *Concorde* in a Linux-like environment, also using the linear programming solvers QSopt and CPLEX. All steps below were tested on computers running Ubuntu Linux and Mac OS Sierra.

Installation and compilation

Before getting started, check if your machine has QSopt or CPLEX installed.

Obs: also check your computer architecture, it may prevent you from installing correctly the solvers!

1. Qsopt: 64 bits[3] - 32 bits[4]
2. CPLEX[5] (you willl need to register at the IBM website and then choose the proper version, depending on the architecture of the computer)

[3] `http://www.math.uwaterloo.ca/~bico/qsopt/beta/index.html`

[4] `http://www.math.uwaterloo.ca/~bico/qsopt/downloads/downloads.htm`

[5] `http://www-01.ibm.com/support/docview.wss?uid=swg24036489`

The *Concorde* installation may require different steps, depending on the option of linear programming solver. Therefore, the next two sections describe each different option, one for QSopt and another for CPLEX.

QSopt

After downloading QSopt, create a symbolic link to qsopt.a file. In the same folder where QSopt files are:

```
$ sudo ln -s qsopt.a libqsopt.a .
```

Download *Concorde* source code, available at the download section of the official site[6], (last version available in December 19, 2003).

The file is compressed. To decompress it, on the terminal:

```
$ gunzip co031219.tgz
$ tar xvf co031219.tar
```

Enter the "concorde" folder (just created), and then configure:

```
$ cd concorde
$ ./configure --with-qsopt=DIR
```

(where DIR is the folder containing files "qsopt.h" and "qsopt.a")

Run the *Concorde* makefile:

```
$ make
```

Create a symbolic link to *Concorde* library. From the *Concorde* installation folder:

```
$ sudo ln -s concorde.a libconcorde.a .
```

Now the installation is ready. However, we found that some bugs need to be fixed in order to solve TSP instances, like the ones available at the TSPLIB. For example, the code contains some identifiers named "new" and "class", which are reserved names in C++.

It suffices to edit the file "concorde.h" (in the installation folder) and exclude (or modify) such identifiers when used as names for parameters. Note that excluding these names is correct, because the C++ header files do not need to identify parameters using names.

Obs: Be careful no to "find & replace" all occurrences, as it might get you in trouble.

In order to compile a code using *Concorde*, you need to adjust some parameters. We suggest a makefile that will help. In the following, DIRCONCORDE is the path to the *Concorde* installation folder and DIRQSOPT is the folder where the QSopt files are.

```
CXX= g++

CONCORDE=DIRCONCORDE

QSOPT=DIRQSOPT

CPPFLAGS= -w -m64

LIBFLAGS= -I$(QSOPT) -I$(CONCORDE) -L$(CONCORDE) -L$(QSOPT) -
    ↪ lconcorde -lqsopt -lm -lpthread -ldl

all:
        $(CXX) main.cpp -o exec $(LIBFLAGS) $(CPPFLAGS)

        @echo ‘‘-- DONE --’’
run:
```

[6] http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm

```
./bin/main
```

Run the makefile using the command *make* in the same folder where the makefile is and your binary will be generated.

Alright, ready.

SHOW TIME

CPLEX

After downloading CPLEX, download the *Concorde* source code, available at the download section of the official site[7], (last version available in December 19, 2003).

[7] http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm

The file is compressed. To decompress it, on the terminal:

```
$ gunzip co031219.tgz
$ tar xvf co031219.tar
```

In the *Concorde* folder, edit the file "Makefile.in". Add -lpthread to the end of the line defining "LIBFLAGS".

From the same folder, enter the folder "TSP" and again edit the file "Makefile.in" in the same manner as the previous one.

Using the terminal, create symbolic links to the CPLEX header files. From the *Concorde* installation folder:

```
$ sudo ln -s /opt/ibm/ILOG/CPLEX\_Studio1263/cplex/include/
    ↪ ilcplex/*.h .
$ sudo ln -s /opt/ibm/ILOG/CPLEX\_Studio1263/cplex/lib/x86
    ↪ -64\_linux/static\_pic/libcplex.a .
```

In the "LP" folder, edit the "lpcplex8.c" file, including the line #define CPX_PARAM_FASTMIP 1017. This constant was used in previous versions of CPLEX, however, it was removed from earlier versions.

From the *Concorde* installation folder, configure the use of CPLEX. DIR is the path to the *Concorde* installation folder, that now also contains the links to CPLEX headers:

```
$ ./configure --prefix=DIR --with-cplex=DIR
```

Run the makefile:

```
$ make
```

Create a symbolic link to *Concorde* library. From the *Concorde* installation folder:

```
$ sudo ln -s concorde.a libconcorde.a .
```

Obs: in one of our tests, this last step failed. In this case, a possible workaround it to remove the created "libconcorde.a" file, copy the "concorde.a" file and rename it to "libconcorde.a".

Now the installation is ready. However, we found that some bugs need to be fixed in order to solve TSP instances, like the ones available at the TSPLIB. For example, the code contains some identifiers named "new" and "class", which are reserved names in C++.

It suffices to edit the file "concorde.h" (in the installation folder) and exclude (or modify) such identifiers when used as names for parameters. Note that excluding these names is correct, because in C++ the header files do not need to identify parameters using names.

Obs: Be careful no to "find & replace" all ocurrences, as it might get you in trouble.

In order to compile a code using *Concorde*, you need to adjust some parameters. We suggest a makefile that will help. In the following, DIRCONCORDE is the path to the *Concorde* installation folder.

```
CXX= g++

CONCORDE=DIRCONCORDE

CPPFLAGS= -w -m64

LIBFLAGS= -I$(CONCORDE) -L$(CONCORDE) -lconcorde -lcplex -lm -
    ↪ lpthread -ldl

all:
        $(CXX) main.cpp -o exec $(LIBFLAGS) $(CPPFLAGS)

        @echo ''-- DONE --''

run:
        ./bin/main
```

Run the makefile using the command *make* in the same directory where the makefile is and your binary will be generated.

Alright, ready.

Utilization

For this section we will consider the C++ language and STL containers.

The idea is to solve TSP-like problems, thus, you need to model your problem as a graph with n nodes. To achieve this, an $n \times n$ distance matrix must be provided, indicating the distance between each pair of nodes on the graph. If there is no edge between two nodes, you can assume distance ∞. Additionally, a vector of size n is used by Concorde to return the optimal route.

We describe a function with two parameters: the distance matrix and the solution vector. This function calls a Concorde procedure that transforms the distance matrix into the structure used by the solver, named *CCutil_graph2dat_matrix* and then solves the problem using the function *CCtsp_solve_dat*. Some additional parameters must be provided for these functions, described below. The identifiers used are the same of the official documentation[8] [9].

[8] http://www.math.uwaterloo.ca/tsp/concorde/DOC/util.html#CCutil_graph2dat_matrix

[9] http://www.math.uwaterloo.ca/tsp/concorde/DOC/tsp.html#CCtsp_solve_dat

Function *CCutil_graph2dat_matrix*:

int ncont number of nodes (n);

int ecount number of edges;

*int *elist* array of edges (pairs of nodes);

*int *elen* array of edge lengths;

int defaultelen parametro *default* para o peso das arestas;

*CCdatagroup *dat* Concorde specific data structure containing the problem;

Function *CCtsp_solve_dat*:

int ncont number of nodes (n);

*CCdatagroup *dat* Concorde specific data structure containing the problem (filled by *CCutil_graph2dat_matrix*;

*int *in_tour* provides an initial solution (can be NULL);

*int *out_tour* receives the sequence of nodes on the solution (can be NULL);

*double *in_val* defines an upper bound to the solution value (can be NULL);

*double *optval* solution value;

*int *success* receives value 1 if the function ran successfully. Otherwise, receives value 0;

*int *foundtour* receives value 1 if a solution was found (if *success* = 0, then the solution might not be optimal;

*char *name* name of files created (if NULL, "noname" is used, however, in multithread environments this can cause problems);

*double *timebound* running time limit;

*int *hit_timebound* receives value 1 if the time limit was reached (can be NULL);

int silent if different of zero, supresses part of the output;

*CCrandstate *rstate* object of the random numbers generator;

Obs: To call these functions, the code must include the library "concorde.h".

Next, we present a minimal working example:

```
extern "C" {    #include <concorde.h> }

//function that calls Concorde solver functions
void solving_tsp_concorde(int[][] distance, int[] tour){

    //creating a sequential tour
    for(int i = 0; i < tour->size(); i++){
        tour->at(i) = i;
    }
    if(dist->size() > 4 ){//TSP for more than 4 cities
        int rval = 0;
        int seed = rand();
        double szeit, optval, *in_val, *timebound;
        int ncount, success, foundtour, hit_timebound = 0;
        int *in_tour = (int *) NULL;
        int *out_tour = (int *) NULL;
        CCrandstate rstate;
        char *name = (char *) NULL;
        static int silent = 1;
        CCutil_sprand(seed, &rstate);
        in_val = (double *) NULL;
        timebound = (double *) NULL;
        ncount = dist->size();
        int ecount = (ncount * (ncount - 1)) / 2;
        int *elist = new int[ecount * 2];
        int *elen = new int[ecount];
        int edge = 0;
        int edgeWeight = 0;
        for (int i = 0; i < ncount; i++) {
            for (int j = i + 1; j < ncount; j++) {
                if (i != j) {
                    elist[edge] = i;
                    elist[edge + 1] = j;
                    elen[edgeWeight]    = dist->at(i)[j];
                    edgeWeight++;
                    edge = edge + 2;
                }
            }
        }
        out_tour = CC_SAFE_MALLOC (ncount, int);
        name = CCtsp_problabel("␣");

        CCdatagroup dat;
                CCutil_init_datagroup (&dat);
                rval = CCutil_graph2dat_matrix (ncount, ecount,
                    ↪ elist, elen, 1, &dat);

                rval = CCtsp_solve_dat (ncount, &dat, in_tour,
                    ↪ out_tour, NULL, &optval, &success, &
                    ↪ foundtour, name, timebound, &hit_timebound,
                    ↪ silent, &rstate);

        for (int i = 0; i < ncount; i++) {
            tour->at(i) = out_tour[i];
        }
        szeit = CCutil_zeit();
        CC_IFFREE (elist, int);
        CC_IFFREE (elen, int);
        CC_IFFREE (out_tour, int);
        CC_IFFREE (probname, char);
    }

}

int main(){

    //distance matrix and vector initialization
    solving_tsp_concorde(distance,tour);

    //prints the solution
    for(int i = 0; i < tour->size(); i++){
        cout<<tour->at(i)<<"␣";
    }

    //build here the solution to the original problem

    return 0;
}
```

Model your TSP-like problem according to this guide steps and solve it exactly.

References

Some references were consulted to create this guide:

http://rodrigobrito.net/2016/08/14/instalando-concorde-tsp-solver/

http://www.leandro-coelho.com/installing-concorde-tsp-with-cplex-linux/#more-146

http://www.math.uwaterloo.ca/tsp/concorde