B Concorde TSP Solver

Concorde TSP Solver [2] is a package for solving the symmetric Traveling Salesman Problem and other network optimization problems. It's written in C and is free for academic research use. It has been used to solve all the 110 instances of the TSPLIB to the optimum.

B.1 Building Concorde

To use the Concorde TSP Solver Callable Library from a C program, the source code must be compiled into a static library file (`.a` extension in Linux). The latest release of Concorde dates back to 2003, when CPLEX version was 8.0. Hence, parts of the code must be adapted to make it compatible with the latest CPLEX releases. Below, we explain how to compile the source on a Linux machine.

After downloading, extracting, and entering the `concorde` directory, the first thing to do is configuring the `Makefile`'s. This is done by running the `configure` script inside the current folder with option `--with-cplex` set to the absolute path of the folder containing the `cplex.h` header file. On our machine it was:

```
$ ./configure --with-cplex=/opt/ibm/ILOG/CPLEX_Studio1210/cplex/include/ilcplex
```

The `Makefile`'s generated by the configuration script incorrectly assume the path to the CPLEX static library. To correct this issue open each `Makefile` and replace line

```
LPSOLVER_LIB = /path/to/cplex/libcplex.a
```

with

```
LPSOLVER_LIB = /path/to/cplex/library/libcplex.a
```

On our machine it was

```
LPSOLVER_LIB = /opt/ibm/ILOG/CPLEX_Studio1210/cplex/lib/x86-64_linux/static_pic/
    libcplex.a
```

[2]http://www.math.uwaterloo.ca/tsp/concorde.html

Now that the `Makefile`'s are correct, we need to edit file `LP/lpcplex8.c`. In fact, it uses the deprecated CPLEX parameter `CPX_PARAM_FASTMIP`, resulting in the error shown below.

```
lpcplex8.c:1921:39: error: 'CPX_PARAM_FASTMIP' undeclared (first use in this function)
  1921 |     rval = CPXsetintparam (cplex_env, CPX_PARAM_FASTMIP, params->fastmip);
```

To fix this last problem, one could either get rid of the faulty line, or re–define the parameter. We chose to redefine the parameter, hence editing `LP/lpcplex8.c` by adding the following lines of code:

```
#ifndef CPX_PARAM_FASTMIP
#define CPX_PARAM_FASTMIP 1017
#endif
```

Finally, run `make` to compile the sources. A static library file named `concorde.a` will be generated in the current directory. Make it executable with

```
$ chmod +x concorde.a
```

and change its name to `libconcorde.a`. At this point, we simply copied `concorde.h` header file to the include path of your project and `libconcorde.a` to the library path of our project, and compiled the sources with `-lconcorde` flag in GCC.

B.2 Useful functions

In our work we used two of the several functions that Concorde makes available. The first one is `CCcut_connect_components`. Its purpose is to detect the connected components of a graph. It takes the following arguments:

`ncount` An integer that specifies the number of nodes in the graph.

`ecount` An integer that specifies the number of edges in the graph.

`elist` A pointer to a vector of length `2 * ecount` that specifies the two nodes pertaining to each edge. In detail, `elist[2*i]` contains the index of one of the nodes of the i–th edge, and `elist[2*i + 1]` contains the other node index.

`x` A pointer to a vector of length `ecount` that specifies the solution to be examined. Each value is the capacity of the corresponding edge.

`ncomp` An integer pointer to receive the number of connected components.

`compscount` An array to receive the number of nodes in each component.

`comps` An array to receive the edges pertaining to each component in the same format of `elist`. This should be used in conjunction with `compscount` to find which component each edge belongs to.

The second function is `CCcut_violated_cuts`. Its purpose is to detect cuts whose capacity is smaller than a certain threshold. We used it to find the violated SEC's. It takes the following arguments:

`ncount` An integer that specifies the number of nodes in the graph.

`ecount` An integer that specifies the number of edges in the graph.

elist	A pointer to a vector of length `2 * ecount` that specifies the two nodes pertaining to each edge. In detail, `elist[2*i]` contains the index of one of the nodes of the i–th edge, and `elist[2*i + 1]` contains the other node index.
x	A pointer to a vector of length `ecount` that specifies the solution to be examined. Each value is the capacity of the corresponding edge.
cutoff	A double that specifies the threshold to determine whether a cut is violated or not.
doit_fn	A callback function that is invoked every time a violated cut is found. See below for more details.
pass_param	A pointer to a user–handle which is passed to the `doit_fn` function.

Note that this function assumes the graph is connected and undirected. The shrinking routines assume that we are working with the TSP and are not interested in cuts of weight 2.0 or more.

The `int (*doit_fn)` callback is invoked every time a violated cut is found, and takes the following argument types:

double	The value of the cut.
int	The number of nodes in the cut.
int *	A pointer to a vector that specifies the indexes of the nodes in the cut.
void *	A pointer to a user–handle passed previously to `CCcut_violated_cuts`.