System environment:

The software has been compiled and run successfully on a Windows 7 Enterprise machine using Microsoft Visual Studio 2010 (http://www.microsoft.com/visualstudio/en-us). An earlier version of the software has been compiled and run successfully on a Windows XP Professional machine with cygwin (http://cygwin.com/) providing the simulated Linux environment. The compiler was GCC (http://gcc.gnu.org/). The development platform was Eclipse (http://www.eclipse.org/).

ISC uses an open-source linear programming solver library called lp_solve (http://tech.groups.yahoo.com/group/lp_solve/). The version used during the development of this software was lp_solve 5.1. The user should download the proper library from the lp_solve website and set up the linker accordingly. We suggest that lp_solve 5.1 be used together with ISC because we had memory leakage problems with lp_solve 5.5. The library file liblpsolve51.a is needed if the platform is Cygwin and lpsolve51.lib and lpsolve51.dll for Microsoft Visual Studio.

The software is written in standard C++. No vendor-specific C++ feature is used. In principle, any C++ compiler supporting standard C++ should be able to compile it. However, it is up to the user to figure out how to do that in their own computing environment. In the authors' own experience, VC++ and GCC have slightly different rules. The current version supports VC++ but should also be compatible with GCC, or at least with very minor changes.

Below we provide brief instructions (taken from help documents from Visual Studio and Eclipse) for Visual Studio and Eclipse. What we provide is a way that works, but may not be the "best" way among many possible ways. Other than the following descriptions, no support will be provided.

1. Visual Studio 2010

a. Download and save the source files (.cpp) and header files (.h) into one directory on your local machine.

b. Obtain lp_solve from http://tech.groups.yahoo.com/group/lp_solve/. For a Visual Studio user, he/she can download **lp_solve_5.1_dev.zip**. Then unzip the file and copy header files into the directory containing ISC files.

c. Create a Visual C++ project with the source files and header files downloaded.

   i) Choose File->New->Project From Existing Code. In the wizard window, make sure you choose to create a Visual C++ project. Click next to move to the next window.

a. Project file location: In this field, specify in this field where you want to keep your project files using the Browse button.
b. Project name: give your project a name.
c. Make sure the box next to "Add files to the project from these folders" is checked.
d. Use the "Add" button to add the directory containing the source files and header files. Use the "Remove" button to remove any directory that should not be selected.
e. Click "Next". Make sure "Use Visual Studio" is selected, and select "Console application project".
f. Click "Finish".

ii) Build the project.

a. From **lp_solve_5.1_dev.zip**, copy "lpsolve51.lib" and "lpsolve51.dll" into the VC++ project directory.
b. In the solution explorer window, make sure that the ISC project is selected. Then right click, choose "Properties".
c. In the Property window, expand "Configuration Properties", and then click on Linker to expand it. Then click on "Input", and click on "Additional Dependencies". Type in "lpsolve51.lib".
d. Now build the project F7. You may see some performance warning messages. Please ignore them.

2. Eclipse on a Windows machine with Cygwin. If the user has a Linux/Unix environment, the only difference is how you obtain the appropriate lp_solve library files.

a. Make sure you can compile C++ with your Eclipse (CDT installed)

b. Download and save the source files (.cpp) and header files (.h) into one directory on your local machine.

c. Obtain lp_solve from http://tech.groups.yahoo.com/group/lp_solve/. For Windows user, download **lp_solve_5.1_source.tar.gz**, unzip the file, and inside the root directory "lp_solve_5.1", find the directory "lpsolve51". Run "cgcc.bat" under that directory. This may take a few minutes. So please be patient. After it finishes, the user should be able to see a file "liblpsolve51.a". Copy that file, together with header files from **lp_solve_5.1_dev.zip**, into the directory containing ISC codes.

d. Import source codes into Eclipse

i) Create a new project by File->New->Project. Choose C++, and then Managed Make C++ Project. Click next. Then specify the name and location of the project. Click Next. Then make sure Project Type is "Executable", and at least the "Debug" configuration is marked.
ii) Choose File->Import->General->File Systems. Click Next.
iii) In "From directory", browse and find the directory containing source codes. Check the box besides the name of the directory to select all files. In "Into folder", browse and select the directory where the user wants to keep the source codes. For example, we can use the default, the root directory of the project directory. Depending on your Eclipse configuration, Eclipse may begin building the project, and if so, will report a link error.

e. Build the project
   i) Choose Project->Properties->C++ Build. On the right part of the window, choose “Tool Settings” tab, expand linker options, and click on “Libraries”. In the “Libraries(-l)” field, use the green “Add” button to add “lpsolve51”. In the “Library search path” field, use the green “Add” button to specify the directory where the file “liblpsolve51.a” is stored. If the user stores all source codes and lpsolve files in the project root directory, just click the “Workspace” button.
   ii) Eclipse will build/rebuild the project, and hopefully it is successful.

Using ISC to solve a discrete optimization via simulation problem:

Overview: ISC controls the optimization run. It assumes that there is a simulator object that takes certain input arguments and returns simulation output, as described below. ISC reports progress on a console window and saves the objective values of the best solution found so far into a text file. In ISC terminology, a “solution” is a setting of the integer decision variables.

ISC comes with a random number generator (Copyright: Pierre L'Ecuyer, University of Montreal, http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf) for its own internal algorithms. The user can use this or any other random number generator for their simulation.

1. Code the simulation.

Include “masterheader.h”, which contains all header files used in ISC. At a minimum, the user must include “Simulator.h” and derive from the abstract base class Simulator. The user should implement three public methods:

- double simulation(const Solution* const solution): this method runs the simulator to evaluate the solution represented by the pointer solution. Inside this method, the programmer can use two methods defined for the Solution class to recover values of the integer solution to be evaluated as illustrated below:
  ```
  int dimension;
  dimension = solution->getDimension();
  const int* const x = solution->getDiscreteSolution();
  ```
  In the codes above, dimension gives the size of the integer array that contains the value of the integer solution, with x being the pointer to the array. When this method finishes, it must return a double as the simulated objective value of the integer solution.
- double gettruevalue(const Solution* const solution): The gettruevalue method returns the true expected value of the objective function evaluated at the integer solution represented by the pointer solution if the true value is known (we used this option to study the performance of ISC on problems with known expected values), or simply the sample mean, recovered by double solution->getSampleMean() if the true objective value is unknown and estimated. For most

users who use ISC to solve actual simulation optimization problems with no known true expected value, they can simply write

```
double gettruevalue(const Solution* const solution) {return solution->getSampleMean();}
```

See ResponseSurface.h and ResponseSurface.cpp for examples of writing simple simulations.

- int isGlobalOptimum(const Solution& solution): this method determines if a solution is the true global optimum (1 if yes, 0 if no). Obviously, this is only possible if the global optimum is known in advance. For problems with no known solution, simply return a 0 in this function.

Finally, include the new header file for the simulator in "masterheader.h".

2. Modify the main() function

Once the simulation is written, modify the main() function inside main.cpp. We use the following comment line to indicate that the next line is where to make the change

```
/***********************************************/
```

- Specify the path and name of the file containing control parameters for ISC. For example, in the line below, the input file named inputFileF22.txt is located in the parent directory of the directory containing the source codes.

  ```
  const char* inputFiles = {"./inputFileF22.txt"};
  ```

- Create a simulator object as showed below. Replace "ResponseSurface" with the class name for the user's own simulator.

  ```
  ResponseSurface sim;
  ```

3. Write the input file.

In the directory specified in Step 2, edit a text file with the file name specified in Step 2. The format of the input text file MUST strictly follow the format described here. Below is the content of an input file for a 5 dimensional test problem:

```
OUTFILE ./output/d5/d5co
BUDGET 1000000
NUM_MACRO_REPS 5
BACK_TRACKING_TEST -2
OCBA -1
GA_NO_IMPROVE_GENERATION -3
DOMIN_NICHE -1
CLEANUP -1
STOC_SIM 1
DELTA_GLOBAL -20
BACK_TRACKING_ALPHA -0.1
BACK_TRACKING_DELTA -1
LOCAL_OPT_ALPHA -0.05
LOCAL_OPT_DELTA -0.5
CLEANUP_ALPHA -0.05
CLEANUP_DELTA 10
INIT_NUM_OBS 5
```

```
NUM_LOCAL_CANDS 5
ELITISM -1
CSTR_PRUNING_FREQ 50
MPA_MTHD 0
SAMPLING_MTHD 0
GA_BUDGET 0
DIMENSION 5
NUM_CONSTRAINTS 10
25 25 25 25 25
1 0 0 0 0 -5000
-1 0 0 0 0 -5000
0 1 0 0 0 -5000
0 -1 0 0 0 -5000
0 0 1 0 0 -5000
0 0 -1 0 0 -5000
0 0 0 1 0 -5000
0 0 0 -1 0 -5000
0 0 0 0 1 -5000
0 0 0 0 -1 -5000
```

There are two parts of this input file.

- The first part has two columns: the first column is a descriptive name (no blank or tab), the second column, separated by a white space or tab from the first column, gives the value for the parameter described by the first column. The order of the rows must not be changed. The last row is "NUM_CONSTRAINTS 10", which tells ISC that the number of constraints for this problem is 10. For a detailed discussion of these parameters, please refer to the paper titled ""Industrial Strength COMPASS: A Comprehensive Algorithm and Software for Optimization via Simulation," *ACM TOMACS* **20** (2010), 1-29, available on http://users.iems.northwestern.edu/~nelsonb/ISC/. We briefly describe them below. In general, we can use a negative number to use default values for most parameters, except for CLEANUP_DELTA.
  - OUTFILE: the location and prefix for the names of the ISC output files
  - BUDGET: number of simulation replications allowed for the entire ISC experiment (including all macro replications), default to 10,000
  - NUM_MACRO_REPS: number of macro replications of ISC runs, default to 5
  - BACK_TRACKING_TEST: whether to use the backtracking test when COMPASS is about to take a backtracking move, default to 0 (no test)
  - OCBA: whether to use OCBA to allocate computing budget, default to 1 (yes)
  - GA_NO_IMPROVE_GENERATION : maximum number of generations without any improvement before the global phase niching genetic algorithm is terminated and ISC moves on to local phase, default to 3
  - DOMIN_NICHE: whether the dominant niche test is used to transition from global to local search, default to 0 (no)
  - CLEANUP: whether to perform final clean-up, default to 1 (yes)
  - STOC_SIM: whether the simulation is stochastic (any nonzero integer) or deterministic (0)
  - DELTA_GLOBAL : the indifference zone parameter delta for the global phase, default to cleanup delta

    - BACK_TRACKING_ALPHA : the significance level of the back tracking test, default to 0.1
    - BACK_TRACKING_DELTA : the indifference zone parameter for backtracking test, default to cleanup delta
    - LOCAL_OPT_ALPHA: the significance level of the local optimality test, default to 0.05
    - LOCAL_OPT_DELTA : the indifference zone parameter delta for the local optimization phase, default to cleanup delta
    - CLEANUP_ALPHA : the significance level of the final cleanup phase, default to 0.05
    - CLEANUP_DELTA: the indifference zone parameter delta for the final cleanup phase, no default value. ISC user must provide a nonnegative value for this parameter to reflect the minimum difference in objective value worth detecting
    - INIT_NUM_OBS: initial number of simulation replications assigned to a new solution, default to 5
    - NUM_LOCAL_CANDS: number of solutions sampled in one local optimization iteration, default to 5
    - ELITISM : whether to use elitism in global genetic algorithm search phase, default to 1 (yes), use 0 to turn it off
    - CSTR_PRUNING_FREQ: constraint pruning frequency for COMPASS, default to 5
    - MPA_MTHD: the geometry of the most promising area (MPA), use 0 (also the default) for the COMPASS MPA, and 1 for the adaptive hyperbox algorithm (AHA)
    - SAMPLING_MTHD: use uniform sampling distribution if 0 (also the default) or 1 for coordinate sampling. Note that if AHA is used, we should set this parameter to 0
    - GA_BUDGET: the portion (between 0 and 1) of simulation budget allocated for GA, default to 0.18
    - DIMENSION: dimension of the test problem, no default
    - NUM_CONSTRAINTS: number of deterministic linear inequality constraints, no default
- The second part gives the initial solution and the linear inequality constraints
  - The first line in the initial solution to start the search. Here we start with a solution of (25, 25, 25, 25, 25)
  - The lines after provide the linear inequality constraints in the format of AX>=B. So the number of columns is problem dimension + 1. The first line is “1 0 0 0 0 -5000”, which gives X_1 >= -5000.

ISC Output

After the user has successfully added the simulator to the ISC software, and created the input text file, ISC is ready to run. ISC generates 4 types of output text files, all starting with the prefix specified in the “OUTFILE” parameter (the first line of the input text file). ISC does not yet have the capability to “flush” out the output when it is killed by the user. So if you decide to kill ISC because it is taking too long to finish, you will lose all information except for the console (screen) output and individual macro replication files (if some macro replications have finished).

Continue with the previous example. All output files start with “d5co”.

1. d5co.dat: this file contains the average of all macro replications of ISC runs. The first column gives the number of simulations, and the second one is the average objective value across different ISC runs. This file can be used to generate a plot showing the average progress of ISC. If the expected value of the objective function is known for each solution, this plot gives the actual objective value

for the best solutions found up to that point (measured by the number of simulation replications consumed so far). If not, it is the sample average objective value.

2. d5co0.dat: this is the plot for the first macro replication. The content is the same as the previous file otherwise. There is one such file for each macro replication. So if you choose to run 5 macro replications, you will see d5co0.dat up to d5co4.dat.
3. d5coresult.dat: this file records the screen output ISC generates. It contains text messages describing ISC progress. The output is self-explanatory.
4. d5co_summary.txt: this file report the statistics for all macro replications. An example is given below. The first line is a header. For each of the following 4 rows, we report the number of macro replications, the mean, standard deviation, max and min for the number of simulation replications used, whether the best solution found is indeed a local optimum, whether the best solution found is indeed a global optimum (if we know the global optimum a priori), and the objective value of the best solution found. The last column is simply gives the prefix of the output file names as a case tag. In the example below, we know for all 5 macro replications, we find the global optimum in all 5 macro replications (and thus the std is 0, mean is 1, max and min are both 1). We also know on average, each macro replication uses 7774.6 simulation replications with a standard deviation of 822.92. The maximum number of replication used is 9246 and the minimum number of replications used is 6778.

```
Row         N     Mean    Std     Max     Min     Case
NumReps     5     7774.6  822.92  9246    6778    d5co
IsLocalOpt  5     1       0       1       1       d5co
IsGlobalOpt 5     1       0       1       1       d5co
ObjValue    5     -10000  0       -10000  -10000  d5co
```