

Algorithm User Guide

This document contains descriptions on how to compile and use the algorithm for school bus routing described in the following paper:

Lewis, R. and K. Smith-Miles (2018) "A Heuristic Algorithm for Finding Cost-Effective Solutions to Real-World School Bus Routing Problems", Journal of Discrete Algorithms, doi: [10.1016/j.jda.2018.11.001](https://doi.org/10.1016/j.jda.2018.11.001)

This program has been written in C++ (V11) and is available at <http://rhydlewis.eu/resources/busCode.zip>. It has been successfully compiled in Windows using Microsoft Visual Studio and in Linux using the GNU Compiler g++. Instructions on how to do this now follow.

Compilation in Microsoft Visual Studio

To compile and execute using Microsoft Visual Studio the following steps can be taken:

1. Open Visual Studio and click **File**, then **New**, and then **Project from Existing Code**.
2. In the dialogue box, select **Visual C++** and click **Next**.
3. Select the subdirectory containing the **.cpp** and **.h** files, and click **Next**.
4. Finally, select **Console Application Project** for the project type, and then click **Finish**.

The source code can then be viewed and executed from the Visual Studio application. Release mode should be used during compilation to make the program execute at maximum speed.

Compilation with g++

To compile the source code using g++, at the command line use the following command:

```
g++ *.cpp -O3 -o solver
```

This will create a new executable program called **solver** that can then be run from the command line. The optimisation option **-O3** ensures that the algorithms execute at maximum speed. A **makefile** is also supplied if preferred.

Input

Input files must be the correct format. An example input file called **Canberra.bus** has been included with the source code. A full description of this problem format, together with ten real world problem instances can be found at <http://rhydlewis.eu/resources/busprobs.zip>.

Note to Linux/Unix users: The file **Canberra.bus** was created on a DOS/Windows machine and therefore uses a carriage return and line feed for line endings. If you are using a Linux/Unix machine you may find that you first need to convert this file to the correct format for it to be read correctly by the implementation. The command **dos2unix** is suitable for this (<https://en.wikipedia.org/wiki/Unix2dos>).

Usage

Once generated, the executable file should be run from the command line. If the program is called with no arguments, the following usage information is printed to the screen.

```

School Bus Optimiser
-----
USAGE:
---- Compulsory -----
-i <inFileName>      (must be a .bus file in the correct format. Do not include extension)

---- Optional -----
-m <double>          (Maximum bus journey time in minutes. Default = 45.0)
-c                  (Maximum bus capacity. Default = 70)
-t <int>             (CPU time limit per-k in seconds. Default = 10. If negative value used, defines number of
                     calls to LS (iterations) per k instead.)
-D <double>          (Discrete level. Minimum number of secs between each solution in the Pareto front. Larger
                     values make runs faster but less accurate. Default = 10.0)
-M                  (If present, bus stop subsets in Stage-1 must be minimal set coverings; else not.)
-S                  (If present, only Stage 1 of the algorithm is run.)

-----
-d <double> <double> (Dwell time coefficients, seconds-per-passenger and seconds-per-stop resp. Defaults = 5.0 and
                     15.0)
-r <int>             (Random seed. Default = 1)
-k <int>             (Number of buses k to start at. Default is the lower bound (numStudents divided by
                     busCapacity, rounded up to nearest integer))
-v                 (Verbosity. Repeat for more output to the screen)
-----
```

A valid command contains the name of the input file to be used. This is the only mandatory argument. The remaining arguments are optional and are allocated default values if left unspecified. Here are some example commands

```
solver -i Canberra
```

This will execute the algorithm on the problem given in the file **Canbeerra.txt** using all of the defaults given above. To produce more output for the same runs, we could also use

```
solver -i Canberra -v
```

or

```
solver -i Canberra -v -v
```

Another example command is

```
solver -i Canberra -m 60 -t -1000 -M -v
```

This runs the program using a maximum journey time of 60 minutes. In the first stage, the iterated local search algorithm will be run for 1000 iterations and it will use only minimal coverings. Some output will also be produced to the console.

Output Logs

In addition to any output written to the console, each run of this program will also append details of the completed run to the files **log-results.txt** and **log-archive.txt**. The file **log-results.txt** contains, in a single line, the following information separated by tabs in the following order.

(Information on the problem instance and run settings used)

- Input file name
- Number of stopping points in the problem (i.e. the school and all bus stops)
- Number of addresses in the problem

- Number of passengers in the problem
- Distance units used
- Maximum walk distance m_w
- Minimum eligibility distance m_e
- Average number of stops within walking distance of each address
- Average number of addresses within walking distance of each bus stop
- Number of seconds spent dwelling at a bus stop for each passenger
- Number of seconds spent dwelling at each stop
- Maximum bus capacity
- Maximum journey time in minutes
- Random seed
- Time permitted per-k
- Setting used for -D
- Are non-minimum coverings allowed in the solution space? (minCoveringsOnly/AllCoverings)

(Information on the solution produced by the iterated local search algorithm)

- Number of buses needed in the solution final solution
- Cost of this solution
- Number of different bus stops used in this solution
- Number of multistops used in this solution
- Total number of stops in the solution (multistops are counted multiple times)
- Number of routes containing outlier stops
- Was feasibility achieved? (foundFeas/NoFeasFound)

(If used, information on the archive set returned by the multiobjective algorithm is also given)

- Number of solutions, feasible and infeasible, in the final archive set
- Number of feasible solutions in the final archive set
- Total run time

If the multiobjective algorithm has also been employed (i.e. the run setting **-S** was not used in the command line), then the costs of all feasible solutions in the final archive set are also added to the end of the file **log-archive.txt** in two lines.

Copyright notice

Redistribution and use in source and binary forms, with or without modification, of the code associated with this document are permitted provided that citations are made to the publication mentioned at the start of this document. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. This software is provided by the contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage. This software is supplied without any support services.

Please direct any queries/comments to Rhyd Lewis: web: www.rhydLewis.eu, email: LewisR9@cf.ac.uk

R. Lewis (Last updated Tuesday, 13 November 2018)