Algorithms for the k-length cycle and k-length s-cycle problem: User Guide.

R. Lewis, School of Mathematics, Cardiff University, Wales,
www.rhydlewis.eu, LewisR9@cardiff.ac.uk
Last Updated: Thursday, 16 November 2023

1. Summary

This document contains descriptions on how to compile and use the algorithms described in the following publication:

Lewis, R., P. Corcoran, and A. Gagarin (2024) "Methods for Determining Cycles of a Specific Length in Undirected Graphs with Edge Weights" Journal of Combinatorial Optimization, doi: 10.1007/s10878-023-01091-w.

2. Compilation Instructions

Full descriptions of these algorithms can be found in the above publication. The main algorithms are programmed in C++ and can be compiled in both Windows (using Microsoft Visual Studio) and Linux (using **g++**).

To compile and execute using Microsoft Visual Studio, please use the following steps.

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 sub-directory containing all of the **.cpp** and **.h** files, give the project a name, and click **Next**.
4. Finally, select **Console Application Project** for the project type, and then click **Finish**.

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

To compile the source code using **g++**, please use the included **makefile**.

3. Algorithm Input

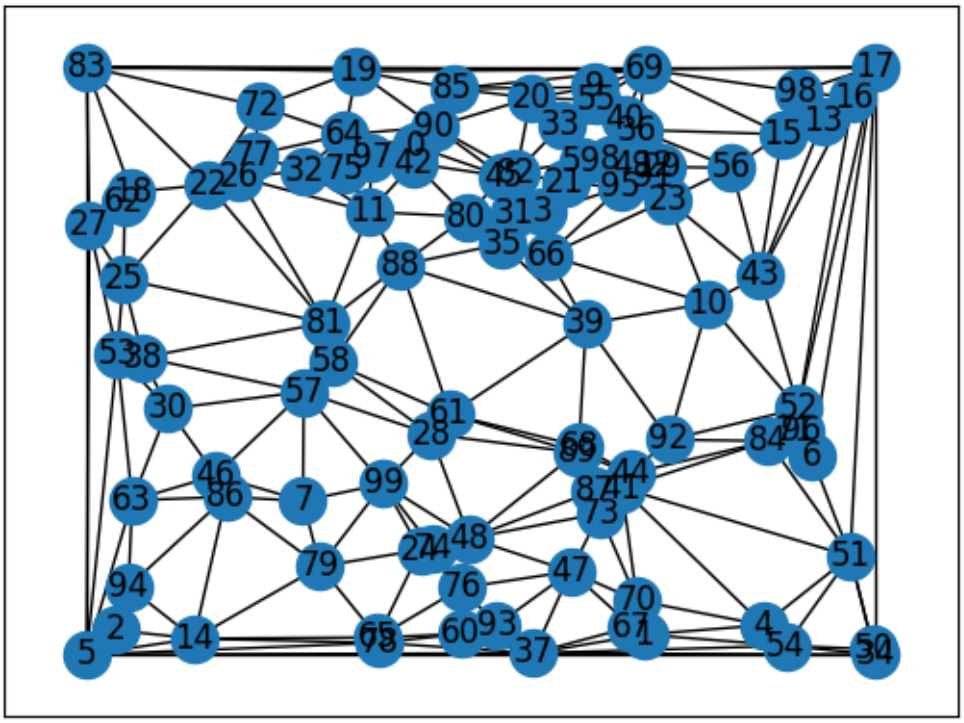
Input files are used to specify individual problem instances for this implementation. These should be undirected, edge-weighted graphs in the correct format. Below are the first few lines of the file **graph.txt**, included as part of this download.

- The initial lines of the file begin with the character "**c**". These lines are comments that give the user information about the input file. They are ignored by the program.
- The single line beginning with "**p**" then specifies the parameters of the graph. In this case, **n** = 100 vertices and **m** = 290 edges.
- There then follows **n** lines, each starting with the character "**v**". These are used to specify the labels of each vertex. In this case, vertex $\mathbf{v_0}$ has the label 0, vertex $\mathbf{v_1}$ has the label 1, and so on.
- Finally, there follows **m** lines that start with the character "**e**". These specify the edges and edge weights of the problem. In the example below, edge $\{\mathbf{v_0}, \mathbf{v_{42}}\}$ has a weight 285, edge $\{\mathbf{v_0}, \mathbf{v_{45}}\}$ has a weight 1206, and so on.

```
c <graph.txt>
c Graph Type      = Planar
c Seed            = 1
c Number of nodes = 100
c Number of edges = 290
p 100 290
v 0
v 1
v 2
v 3
```

```
v 4
...
v 99
e 0 42 285
e 0 45 1206
e 0 90 409
...
e 91 95 439
```

Here is a picture of the planar graph specified in **graph.txt.**



4. Algorithm Usage

Once generated, the executable file should be run from the command line. If the program is called with no arguments, then usage information is printed to the screen. For example, suppose we are using the executable file **kcycle**. Running this program with no arguments from the command line gives the following output:

```
Optimisation algorithm for the k-length cycle problem (KCP) and the k-length s-cycle problem (KSCP) in edge-weighted
undirected graphs. (R. Lewis, Cardiff University, 2023. www.rhydlewis.eu)
Usage:
---
-i <string>   (Required. Must define an input graph in the recognised format. See the documentation for further
               details.)
---
-k <int>      (Desired length of cycle (default = 1000). If -k is given a negative value, the desired length is set to
               infinity and the algorithm will seek the longest cycle)
-s <int>      (Index for the source vertex for KSCP. If a negative value is specified, there is no source vertex and
               the KCP is considered (default = -1).)
-t <double>   (Algorithm time limit in seconds (default = 5).)
-A <int>      (Optimisation algorithm to use (default = 1):
                  1: Great deluge (local search) algorithm
                  2: Yen's algorithm (requires Yen.Jar and a local Java installation)
                  3: Integer programming (requires k-cycle.mos, k-s-cycle.mos, and local installation of Fico Xpress)
-a <int>      (Path-finding algorithm used with local search (default = 1):
                  1: Random first search
                  2: Breadth first search (randomised)
                  3: Depth first search (randomised)
                  4: Wilson's algorithm
```

```
                   5: Dijkstra's algorithm)
-T            (Only has an effect with the KSCP. If -T is present, the graph is not reduced in size during a run.
               When -T is not present, whenever the best solution whose cost > k is improved, graphs are trimmed by
               removing all vertices whose distance from the source is greater or equal to floor(cost / 2) + 1.)
-r <int>      (Random seed (default = 1).)
-v            (Verbosity. If present, output is sent to the console. If -v is repeated, more output is given.)
-stats        (If present, no optimisation is performed. Instead, stats on each BCC are just written to the screen.)
---
```

The string following **-i** should be the name of the file containing the problem instance to be solved. This is the only mandatory argument. The remaining arguments are optional and are allocated default values if left unspecified. Here are some example commands.

```
kcycle -i graph.txt
```

This will execute the algorithm on the problem contained in the file **graph.txt**. The considered problem will be the KCP using a target length of 1000 and a time limit of 5 seconds. Local search will be executed using random first search. Minimal output will be produced. Another example command is:

```
kcycle -i graph.txt -k 10000 -s 0 -t 10 -v -v
```

This uses the same problem instance as before. In this case we are tackling the KSCP with the source vertex $\mathbf{v_0}$ and a desired length of 10000. The time limit is now ten seconds, and a lot of output will be written to the screen. Finally, the following command carries out some simple analyses of the input graph and write this to the screen.

```
kcycle -i graph.txt -stats
```

This produces the following output:

```
Input Graph:         graph.txt
Start Node:          0
Num Non Dyad BCCs:   1
  BCC-0
        NumVertices n:     100
        NumEdges m:        290
        Desired-Len:       1000
        MinEdgeWeight:     77
        MeanEdgeWeight:    0
        MaxEdgeWeight:     10001
        MinDegree:         3
        MinWeightedDegree: 916
        MaxDegree:         10
        MaxWeightedDegree: 47725
        ShortestCycle:     [0, 97, 42, 0]
        LB(WeightedGirth): 1428 (The shortest (s-)cycle in G has this length.)
        LB(BondyAndFan):   0 (= ceil((2 x w(G)) / (n-1)). Must be a cycle of at least this length)
        LB(MinDeg):        1832 (either (a) for all vertices v, there exists a non-Hamiltonian v-cycle of length >=
                            this value; OR (b) all longest cycles in G are Hamiltonian)
        LB(MinDegPair):    2145 (either (a) for all vertices v, there exists a v-cycle of length >= this value; OR (b)
                            G is Hamiltonian)
        UB:                431046 (Sum of all edge weights)
        UB:                279887 (Sum of n heaviest edges)
        UB:                219595 (Two heaviest weights adjacent to a vertex, summed across all vertices, divided by
                                   two)
Stats run on graph.txt
```

5. Algorithm Output

At a minimum, on termination of an optimisation algorithm, the following information is output.

- Specified input file name **-i**.
- Specified target length **-k**.
- Specified time limit **-t**.
- Specified start (source) vertex **-s**.
- Chosen path-finding algorithm **-a**.
- Chosen optimisation algorithm **-A**.

- Was **-T** present in the command? (0/1)
- Specified random seed **-r**.
- Number of occurrences of **-v**.
- Length of best solution observed at termination.
- Was the initial solution improved by the optimiser? (0/1)
- Is this final solution the proven global optimum? (0/1)
- Overall run time (seconds).
- Best cycle produced during the run.

This information is output on a single line, separated by tabs. For example, the first command above gives the following output.

```
graph.txt   1000   5.000000   -1   1   1   0   1   0   1037   1   0   5.000   [96,6,71,96]
```

This information is also appended to the file **resLog.txt.** If the run resulted in an error (due to unrecognised input parameters or an invalid input file), then appropriate single-line error messages are also appended to the file.

6. Copyright Notice