Supplementary Material for the paper *Finding Fixed-Length Circuits and Cycles in Undirected Edge-Weighted Graphs: An Application with Street Networks*

R. Lewis[1] and P. Corcoran[2]
[1]School of Mathematics, Cardiff University, CF24 4AX, Wales.
[2]School of Computer Science and Informatics, Cardiff University, CF24 4AX, Wales.
LewisR9|CorcoranP@cardiff.ac.uk

September 15, 2022

Abstract

This document contains supplemntary material for the paper:

- Lewis, R. and P. Corcoran (2022) "Finding Fixed-Length Circuits and Cycles in Undirected Edge-Weighted Graphs: An Application with Street Networks" in *Journal of Heuristics*, vol. 28, pp. 259-285. `https://link.springer.com/content/pdf/10.1007/s10732-022-09493-5.pdf`.

It contains further figures from the above paper and includes results for both circuits and cycles. It also gives instructions on how to compile and use the C++ algorithms described in the paper.

1 Algorithm Compilation and Usage

This section describes how to compile and use the algorithms proposed in the above publication. The implementations have been written in C++ and are available for download at `http://rhydlewis.eu/resources/kCircuit.zip`. The code can be compiled in Windows using Microsoft Visual Studio and in Linux using g++.

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 source code 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 allow the program execute at maximum speed.

To compile the source code in Linux, please use the included makefile.

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

```
Approximation Algorithms for finding an s-circuit/cycle of length k. (R. Lewis
    2021, www.rhydlewis.eu)
If a k-length solution is not found, lower- and upper-bound candidate solutions
    are output.

Usage:
----------------------
-i <string> (Required. Must define a file in the recognised format -- see the
             documentation for further details.)
----------------------
-k <int>    (Desired length (default = 1000).)
-s <int>    (Source Vertex (default = 0).)
-c <int>    (Solution choice
              1: Circuit (default)
              2: Cycle.)
-a <int>    (Solution generation algorithm:
              1: Double-path heuristic w/ filtering: choose furthest remaining
                 vertex at each step (default)
```

```
                2: Double-path heuristic w/ filtering: choose closest remaining
                   vertex at each step
                3: Double-path heuristic w/ filtering: choose random remaining
                   vertex at each step
                4: Double-path heuristic w/o filtering: choose random remaining
                   vertex at each step
                5: Make a random circuit/cycle (uses Option 4, but halts with
                   the first observed solution)
                6: Split s into <s> and <s'> and run Yen's algorithm until an
                   s-s'-path of length k has been reached)
                7: Make a long(ish) circuit/cycle (uses Option 1 and halts with
                   first observed solution).)
-r <int>     (Random seed (default = 1).)
-LS <int>    (Once a solution has been formed using the option selected with '-a',
              carry out local search using one of these options:
                0: Do not do any local search (default)
                1: Use Dijkstra's algorithm with local search until a local optimum
                   is reached
                2: Use BFS with local search until a local optimum is reached
                3: Use BFS with local search until a local optimum is reached then,
                   from this, use Dijkstra's algorithm with LS until a local optimum
                   is reached).
-sp <int>    (Shortest path algorithm to be used with double-path heuristic:
                1: Moore's BFS shortest path algorithm
                2: Modified Dijkstra's algorithm (default))
-v           (Verbosity. If present, output is sent to the console. If -v is
              repeated, more output is given.)
-o           (If present, remove all degree-one vertices before running the chosen
              algorithm.)
-h           (If present, the double path heuristic halts as soon (and if) a
              solution with length k is found.)
----------------------
NOTES: * When option '-a 6' is used (Yen's algorithm):
       - The file <YenEppstein.jar> should be in the same folder as this
         executable
       - The program will run for a maximum of 30 minutes. In this case
         it will halt with the best solution found.
       - The parameters -c, -sp, -h, -r, and -LS have no effect. Only cycles
         are considered.
       - Dummy vertices and edges are introduced for every articulation point.
         This means that in the original graph there may be non-bridge edges
         that are used twice (this will not be picked up by the solution
         checker due to the dummies). The solutions from this algorithm are
         therefore not subject to the same constraints as the other algorithms.
         The algorithm can also be VERY slow for large values of -k
```

The above provides the information needed to produce valid commands for executing the algorithm at the command prompt. Further information on these parameters can be found by consulting the above publication.

Here is an example command.

```
kcircuit -i planar.txt
```

This will execute the algorithm on the supplied problem instance **planar.txt** using the default settings (as specified above). Here is another valid command:

```
kcircuit -i planar.txt -k 5000 -c 2 -a 1 -v -v
```

This will operate similarly to the previous example, but the algorithm is executed using $k = 5000$, considers cycles instead of circuits, uses the double path heuristic (with filtering and furthest-first vertex selection) and produces a moderate amount of output to the console.

1.1 Input Format

The input file specified by `-i` in the above commands contains the problem instance. This is the only mandatory argument. The format of the input files is a text file based on the DIMACS format. Initial lines in the text file will begin with the character `c`. These are used for comments but are otherwise ignored.

After the comments, the next line in the file should start with the character `p`. This is followed by the number of vertices n and edges m. Next, there follows a series of n lines starting with the character `v`, specifying text labels for each vertex. The first label is associated with the vertex with index 0, the second with vertex index 1, and so on.

Finally, there follows a series of m lines beginning with the character `e`. Each of these lines specifies a single edge of the graph by giving its two endpoints and its (nonnegative) edge weight. Endpoints are given using the vertex indices $0, \ldots, n-1$. Note that each edge `e u v w` should appear exactly once in the input file. It is therefore not repeated as `e v u w`.

The following example shows parts of the supplied problem file **planar.txt**. This contains $n = 40,000$ vertices and $m = 50,000$ edges. The vertex labels and indices are equivalent here.

```
c This is a random, connected, edge-weighted planar graph
c Number of nodes = 40000
c Number of edges = 50000
p 40000 50000
v 0
v 1
v 2
...
v 39998
v 39999
e 0 17724 22
e 0 24110 10
e 0 5390 40
e 0 11778 37
e 0 4012 32
...
```

1.2 Output Format

In addition to any output written to the console, output text files are also produced by this program. The first of these contains the two best solutions found by the algorithm, corresponding to the upper and lower bounds. If a solution of length k has been found, then just one solution is returned. The solution(s) are contained in a list of lists, where each list is a sequence of the vertices specifying a circuit or cycle. Items between the commas in these lists refer to the vertex labels specified in the input file.

Here is some example output corresponding to the supplied input file **planar.txt**. This specifies two solutions that both start and end at the vertex with label 0.

```
[
[0, 5390, 1274, 28494, 2171, 20661, 28562, 4886, 33346, 11746, 4817, 29992, 19491,
     10188, 14817, 11254, 14178, 31612, 20907, 1180, 24621, 1477, 34684, 231,
    20943, 37224, 6004, 20560, 1558, 17712, 2502, 4984, 34829, 10306, 11235, 8718,
    11573, 38046, 13083, 16477, 5941, 7558, 16055, 7558, 32028, 19906, 7216, 9594,
    28927, 36226, 1034, 3093, 38362, 7746, 4360, 7050, 18431, 4216, 7749, 34676,
    8351, 34875, 23155, 11097, 5231, 5534, 21475, 12009, 3263, 13162, 26614, 1015,
    38076, 2171, 28494, 1274, 5390, 24110, 0],
[0, 5390, 1274, 28494, 2171, 16101, 14713, 24223, 39495, 11355, 8763, 13044,
    26196, 328, 34874, 6756, 8681, 9238, 9079, 17137, 15040, 5424, 1481, 36190,
    6069, 3077, 7787, 3776, 11407, 4779, 8490, 6995, 10409, 2106, 33896, 16874,
    9168, 1181, 14399, 7327, 15379, 7327, 35555, 29097, 2677, 32050, 38208, 1989,
    11328, 9426, 30830, 5224, 11978, 25437, 19754, 5079, 2785, 3800, 10234, 2385,
    4231, 32460, 6179, 5923, 6350, 34746, 4633, 17061, 4566, 31633, 1215, 6809,
    30963, 28622, 4886, 28562, 20661, 2171, 28494, 1274, 5390, 24110, 0]
]
```

Each time the algorithm terminates, a single line of information is also written to the text file **resLog.txt**. This contains a record of all of the input parameters, together with information on the resultant solutions and run times (in seconds). In order, these are:

1. Input file name `-i`
2. Number of vertices in the input file
3. Number of edges in the input file
4. Source vertex `-s`
5. Target length `-k`
6. Solution choice `-c`

7. Shortest path algorithm used `-sp`
8. Algorithm choice `-a`
9. Removal of degree-one vertices? `-o`
10. Random seed `-r`
11. Specified verbosity `-v`
12. Should the algorithm halt at the optimal? `-h`
13. Local search option `-LS`
14. Number of vertices in the reduced graph
15. Number of edges in the reduced graph
16. Number of dummy vertices that were added
17. Number of iterations performed by algorithm `-a`
18. Lower bound after execution of algorithm `-a`
19. Upper bound after execution of algorithm `-a`
20. Time at which the best gap was found by algorithm `-a`
21. Execution time of algorithm `-a`
22. Lower bound after execution of local search `-LS`
23. Upper bound after execution of local search `-LS`
24. Number of calls to the shortest path algorithm made by local search
25. Overall run time.

1.3 Software Copyright Notice

1.4 Data Sets

The data sets used to generate the results reported in the above paper are also included with the source code at the link above. These are stored as comma separated value (csv) files. The data from these files was used to generate the charts and tables below.

2 Double Path Heuristic

The figures in this section correspond to the results given in Section 4.2 of the above paper. Here, results for cycles are included in addition to those of circuits.

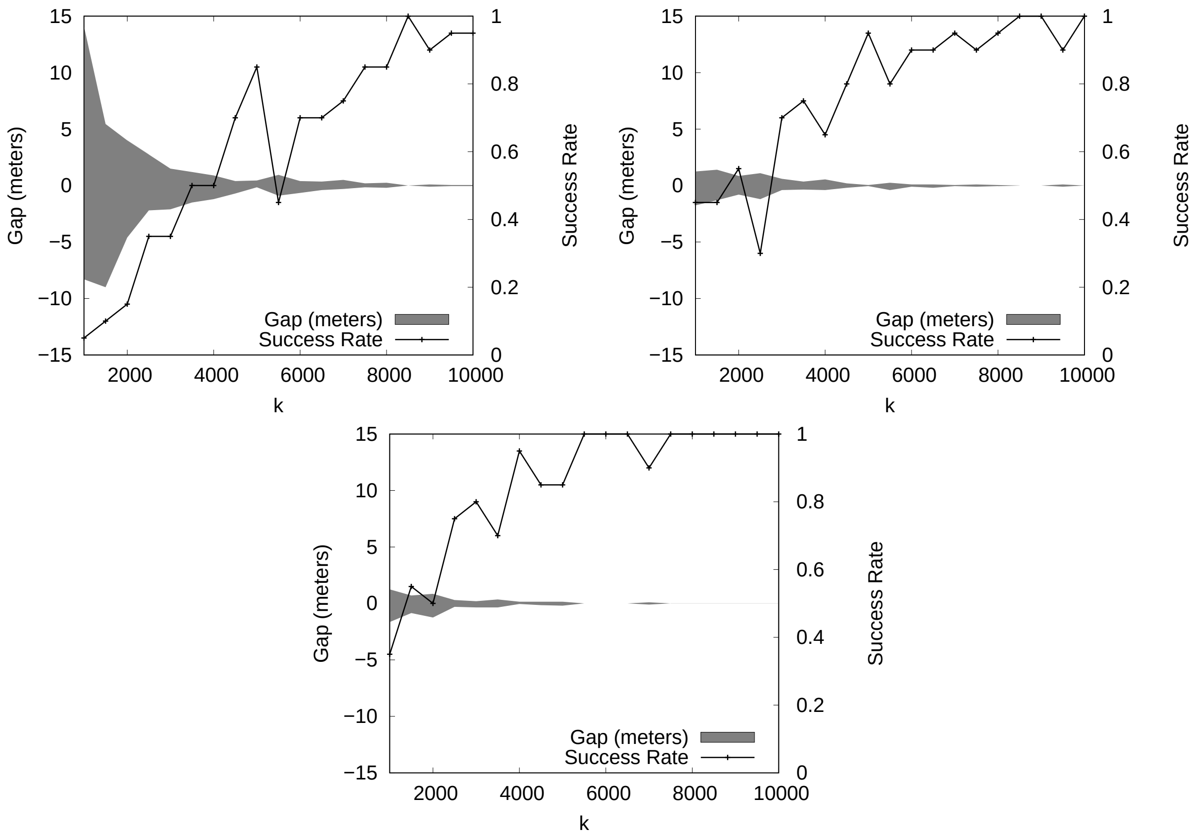

Figure 1: The shaded areas show the gap (in meters) between the two solutions returned by the double path heuristic for differing values of k (using circuits). The lines show the corresponding success rates. Each point is the mean across twenty problem instances for, respectively, sparse, medium, and dense planar graphs.

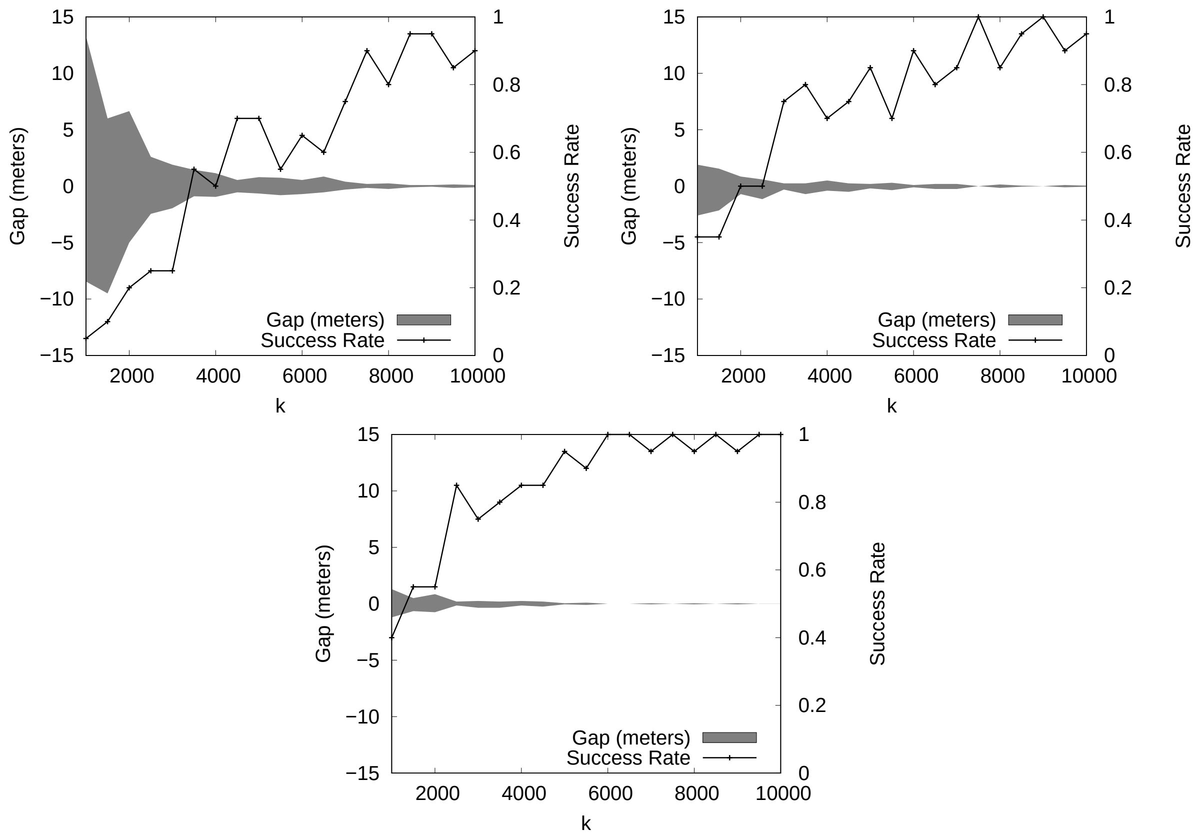

Figure 2: This shows the same information as Figure 1, but considers cycles instead of circuits.

City	$k = 1000$			$k = 5000$			$k = 10,000$		
	LB$-k$	UB$-k$	CPU Time (s)	LB$-k$	UB$-k$	CPU Time (s)	LB$-k$	UB$-k$	CPU Time (s)
Circuits									
London	-1.2	1.8	0.046 ± 0.016	-0.1	0.1	11.835 ± 0.867	0.0	0.0	193.484 ± 9.511
Melbourne	-1.4	2.1	0.031 ± 0.016	-0.1	0.1	8.285 ± 1.591	-0.2	0.2	63.800 ± 7.236
Amsterdam	-3.1	3.0	0.010 ± 0.003	-0.7	0.9	1.997 ± 0.264	-0.4	0.4	21.682 ± 2.009
New York	-5.7	4.9	0.004 ± 0.001	-0.4	0.5	2.116 ± 0.253	-0.2	0.2	25.934 ± 2.099
Kolkata	-6.1	9.2	0.002 ± 0.001	-1.3	1.0	0.976 ± 0.176	-0.5	0.5	13.038 ± 1.388
Circuits (Remove degree-1 vertices)									
London	-2.2	3.8	0.022 ± 0.008	-0.7	0.6	3.518 ± 0.324	-0.3	0.3	48.219 ± 3.425
Melbourne	-3.4	4.9	0.014 ± 0.008	-0.3	0.3	3.564 ± 1.376	-0.4	0.4	22.935 ± 5.215
Amsterdam	-5.7	5.3	0.006 ± 0.002	-1.1	1.2	0.893 ± 0.132	-0.6	0.8	7.924 ± 0.791
New York	-9.7	8.4	0.003 ± 0.001	-0.4	0.7	1.089 ± 0.193	-0.3	0.3	13.983 ± 1.534
Kolkata	-13.9	14.3	0.001 ± 0.001	-1.8	1.8	0.485 ± 0.107	-1.3	1.2	4.604 ± 0.622
Cycles									
London	-1.2	1.9	0.057 ± 0.019	-0.1	0.2	12.023 ± 0.871	0.0	0.0	180.077 ± 8.460
Melbourne	-1.4	2.6	0.034 ± 0.017	-0.1	0.1	8.343 ± 1.582	-0.2	0.2	60.317 ± 6.881
Amsterdam	-2.9	2.9	0.012 ± 0.004	-0.6	0.8	2.031 ± 0.264	-0.5	0.5	21.991 ± 1.916
New York	-5.2	4.6	0.005 ± 0.001	-0.3	0.4	2.126 ± 0.243	-0.1	0.1	25.056 ± 1.797
Kolkata	-6.6	8.9	0.003 ± 0.001	-1.5	0.9	1.022 ± 0.182	-0.4	0.3	13.261 ± 1.420
Cycles (Remove degree-1 vertices)									
London	-2.4	4.4	0.027 ± 0.011	-0.5	0.6	3.621 ± 0.328	-0.3	0.2	45.266 ± 2.991
Melbourne	-3.2	4.9	0.015 ± 0.009	-0.3	0.3	3.626 ± 1.345	-0.4	0.3	22.064 ± 4.960
Amsterdam	-5.6	5.6	0.006 ± 0.002	-1.0	1.1	0.924 ± 0.135	-0.7	0.9	8.085 ± 0.866
New York	-9.1	7.0	0.003 ± 0.001	-0.4	0.6	1.102 ± 0.183	-0.1	0.1	13.466 ± 1.479
Kolkata	-15.8	15.4	0.002 ± 0.001	-2.2	1.4	0.521 ± 0.116	-1.1	1.3	4.737 ± 0.652

Table 1: Accuracy and speed of the double path heuristic on five cities. Each figure is a mean across 50 runs using a randomly selected source vertex within 1 km of the city centre.

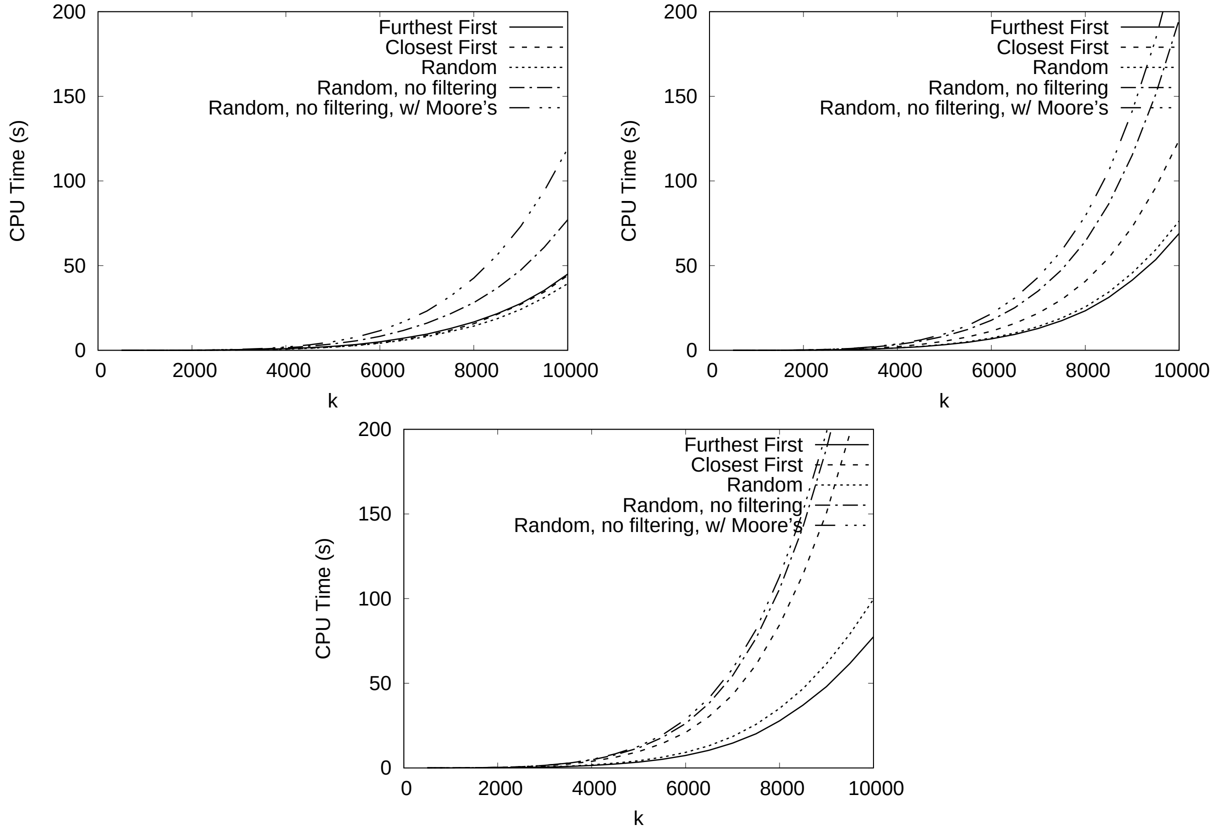

Figure 3: Run times for differing variants of the double path heuristic (for circuits) using various values of k for (respectively) sparse, medium, and dense planar graphs. All points are the mean across twenty problem instances.

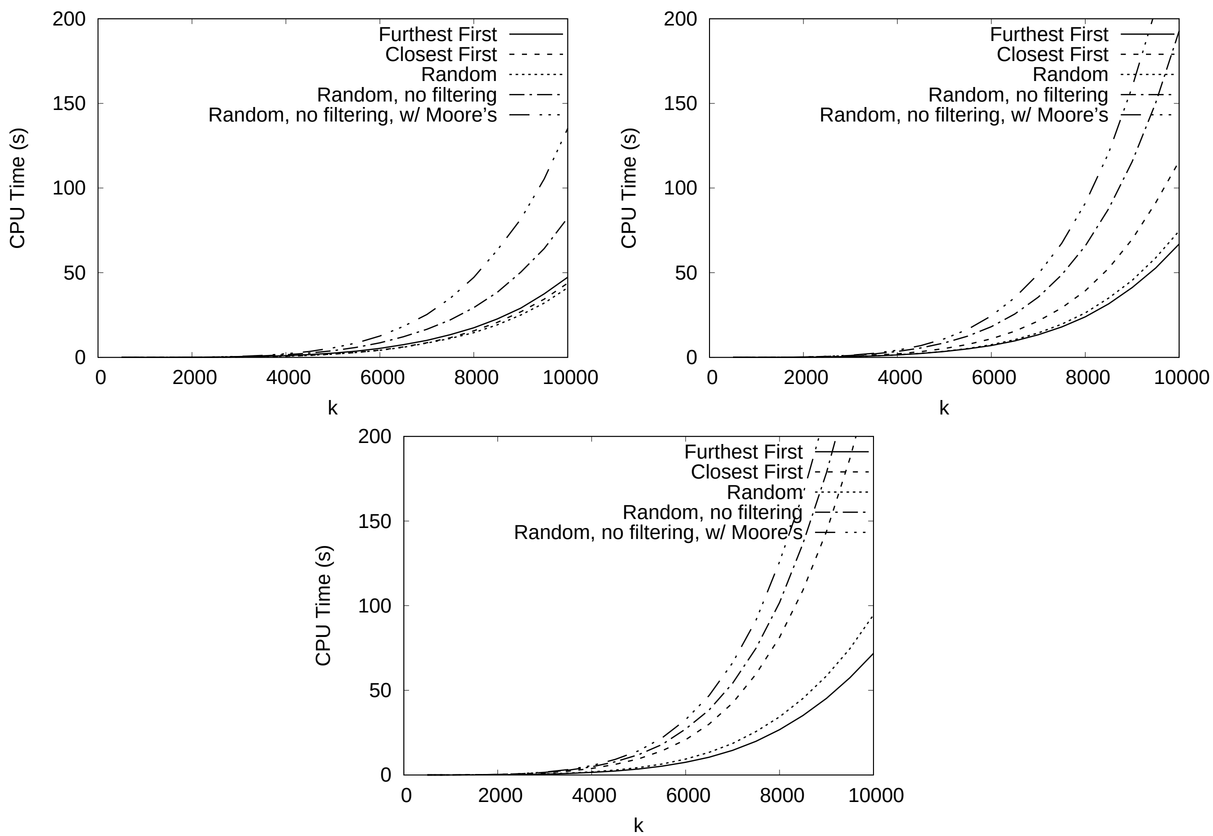

Figure 4: This shows the same results as Figure 3, but considers cycles instead of circuits.

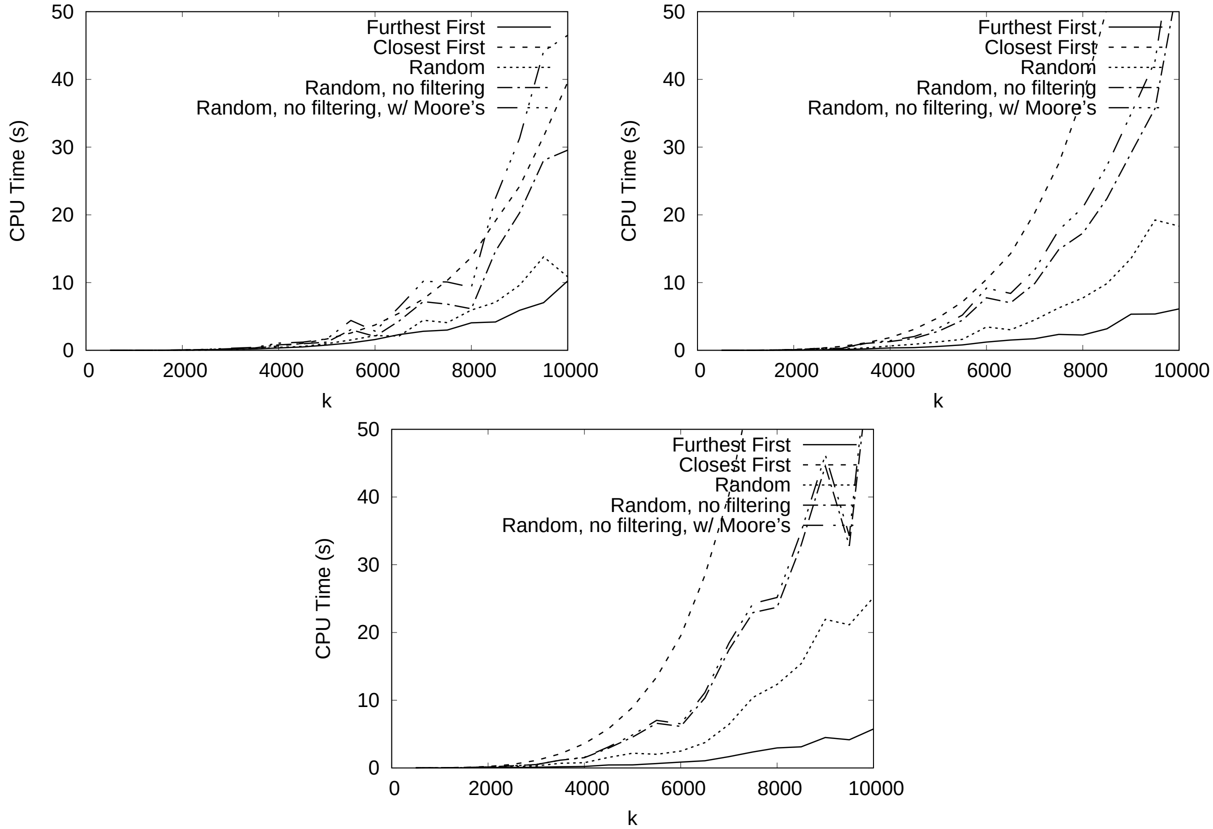

Figure 5: Times at which the best gap was achieved for differing variants of the double path heuristic using circuits. Experimental details are the same as those in Figure 3.

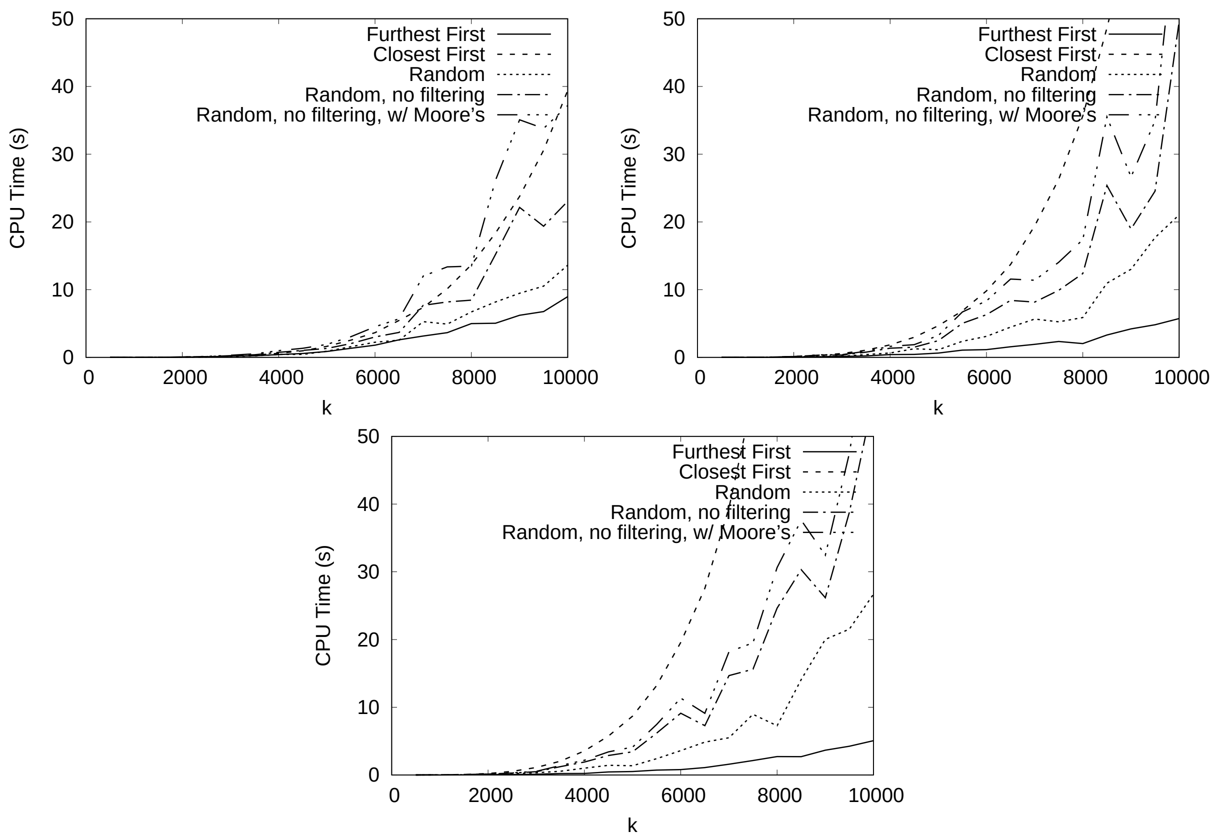

Figure 6: This shows the same results as Figure 5, but considers cycles instead of circuits.

3 Local Search Heuristic

The figures in this section correspond to the results given in Section 4.3 of the above paper. Results for cycles are included in addition to those of circuits.

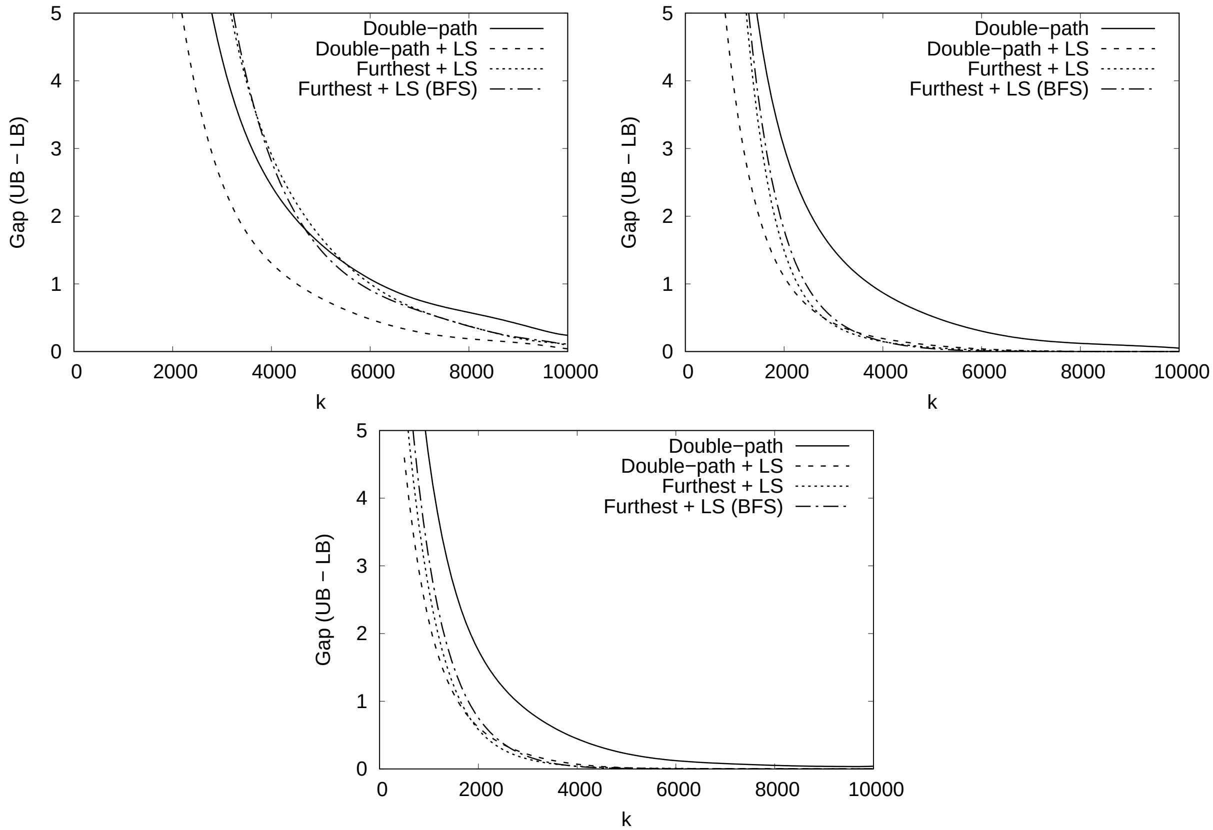

Figure 7: Accuracy of the double path and three local search heuristics for differing values of k using (respectively) sparse, medium, and dense planar graphs. All points are the mean across 100 problem instances. (Circuits)

k	Double-Path	Double-Path + LS	Furthest + LS	Furthest + LS (BFS)
Sparse				
1000	25.20 ± 79.27	16.64 ± 42.39	39.15 ± 54.65	39.52 ± 54.85
2500	4.20 ± 4.70	2.40 ± 3.35	4.71 ± 5.19	5.74 ± 6.57
4500	1.86 ± 2.78	1.00 ± 1.94	1.55 ± 2.51	1.56 ± 2.41
7500	0.66 ± 1.19	0.15 ± 0.56	0.48 ± 1.02	0.54 ± 1.12
10000	0.24 ± 0.65	0.04 ± 0.28	0.11 ± 0.49	0.09 ± 0.45
Medium				
1000	4.87 ± 4.61	2.06 ± 2.51	2.78 ± 3.26	3.59 ± 4.26
2500	1.87 ± 2.10	0.60 ± 1.21	0.42 ± 1.03	0.42 ± 0.97
4500	0.56 ± 1.15	0.10 ± 0.44	0.08 ± 0.39	0.00 ± 0.00
7500	0.17 ± 0.59	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
10000	0.05 ± 0.36	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Dense				
1000	2.98 ± 3.02	1.02 ± 1.50	1.47 ± 2.29	1.59 ± 2.63
2500	0.90 ± 1.42	0.27 ± 0.76	0.22 ± 0.84	0.06 ± 0.34
4500	0.19 ± 0.61	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
7500	0.06 ± 0.34	0.02 ± 0.20	0.00 ± 0.00	0.00 ± 0.00
10000	0.04 ± 0.28	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

Table 2: Accuracy of the double-path and three local search heuristics for differing values of k using sparse, medium, and dense planar graphs with circuits. Entries correspond to a selection of those shown in Figure 7 and show the mean across 100 problem instances, plus/minus the standard deviation.

k	Double-Path	Double-Path + LS	Furthest + LS	Furthest + LS (BFS)
Sparse				
1000	26.09 ± 79.72	16.86 ± 42.51	51.43 ± 112.00	51.74 ± 111.96
2500	4.93 ± 5.61	2.67 ± 3.32	5.56 ± 5.80	6.13 ± 6.81
4500	1.50 ± 2.08	0.64 ± 1.13	1.61 ± 2.37	1.95 ± 3.07
7500	0.48 ± 1.02	0.18 ± 0.64	0.86 ± 1.65	0.52 ± 1.20
10000	0.27 ± 0.71	0.08 ± 0.39	0.42 ± 0.96	0.21 ± 0.64
Medium				
1000	6.03 ± 5.35	3.22 ± 3.59	5.98 ± 7.28	5.42 ± 6.06
2500	1.47 ± 2.09	0.50 ± 1.01	0.58 ± 1.44	1.02 ± 2.28
4500	0.52 ± 1.07	0.12 ± 0.48	0.04 ± 0.28	0.14 ± 0.59
7500	0.17 ± 0.59	0.00 ± 0.00	0.03 ± 0.30	0.00 ± 0.00
10000	0.06 ± 0.34	0.00 ± 0.00	0.00 ± 0.00	0.02 ± 0.20
Dense				
1000	2.25 ± 2.50	1.12 ± 1.71	2.21 ± 3.00	3.47 ± 4.05
2500	0.73 ± 1.32	0.23 ± 0.66	0.29 ± 0.89	0.36 ± 1.06
4500	0.28 ± 0.83	0.06 ± 0.34	0.04 ± 0.28	0.06 ± 0.34
7500	0.02 ± 0.20	0.00 ± 0.00	0.02 ± 0.20	0.00 ± 0.00
10000	0.04 ± 0.28	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

Table 3: This shows the same information as Table 2, but considers cycles instead of circuits.

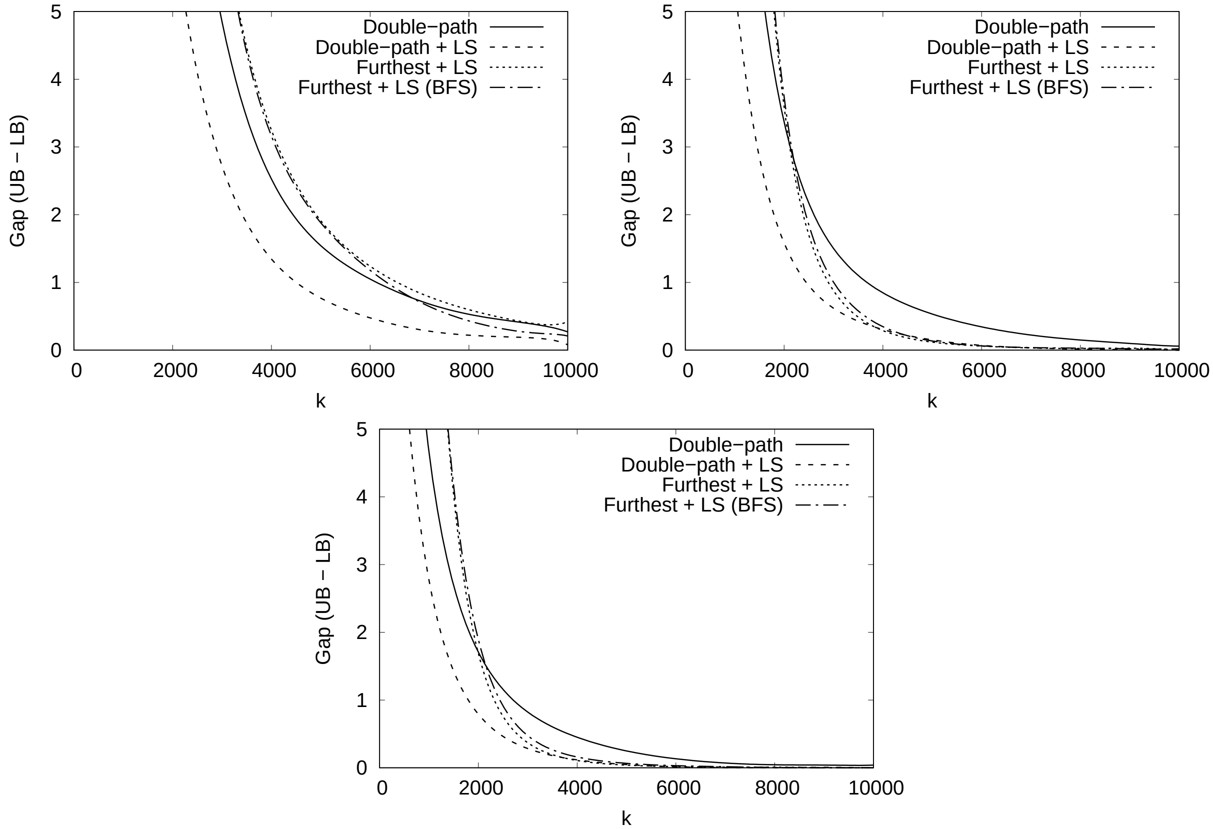

Figure 8: This shows the same results as Figure 7, but considers cycles instead of circuits.

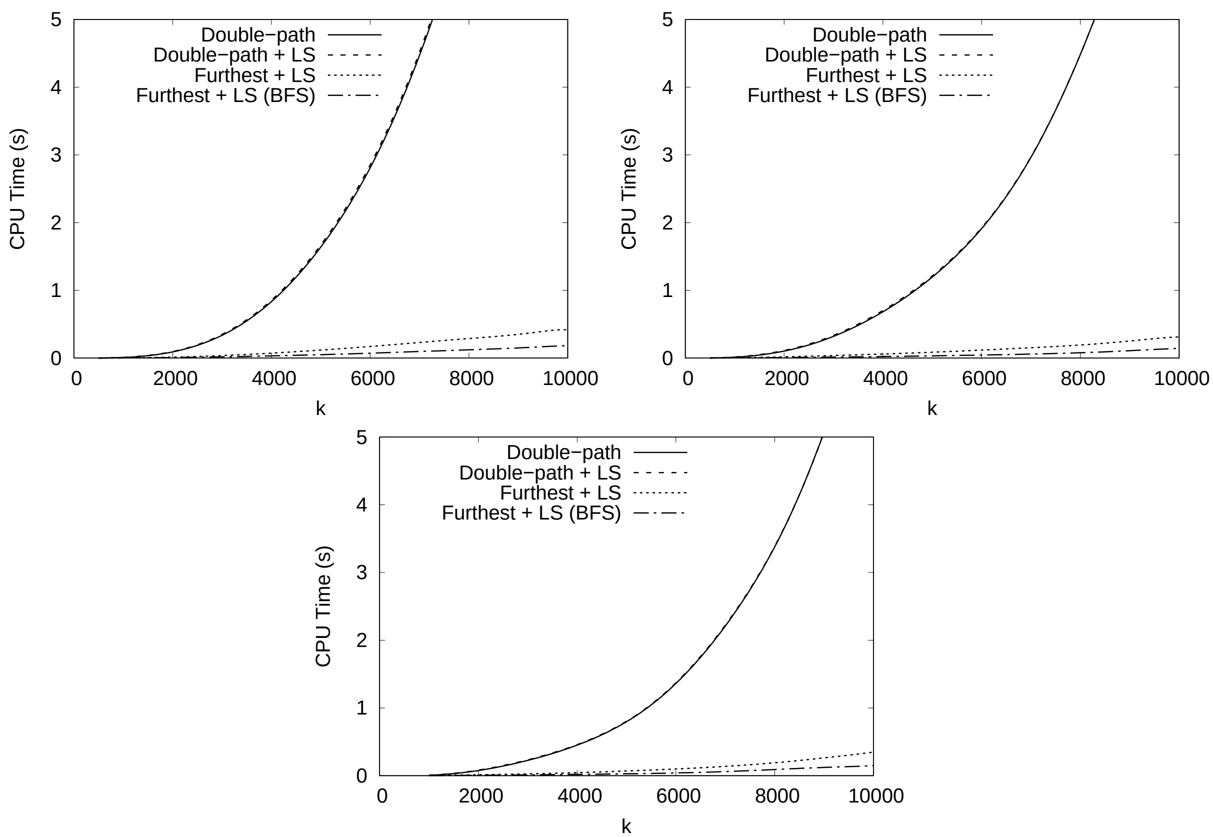

Figure 9: Mean execution times of the trials given in Figure 7 for (respectively) sparse, medium, and dense problem instances.

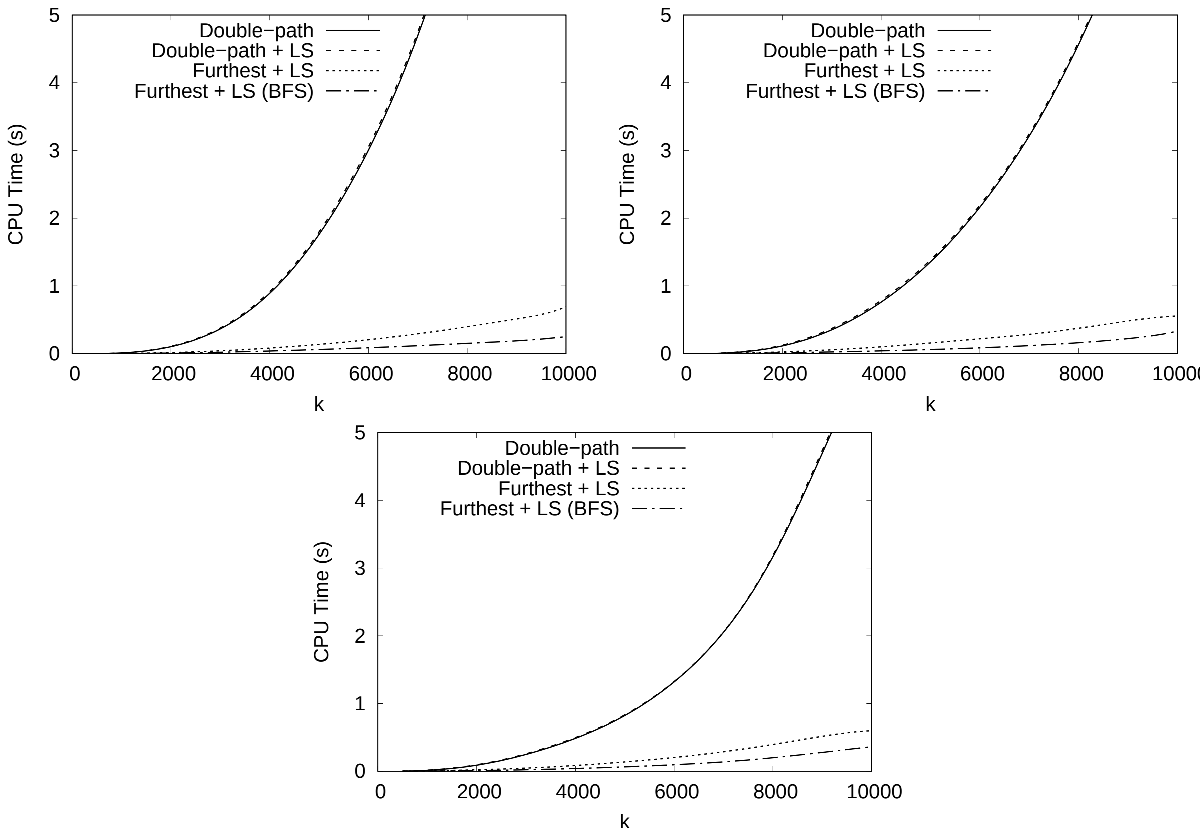

Figure 10: Mean execution times of the trials given in Figure 8 for (respectively) sparse, medium, and dense problem instances.

	$k = 1000$			$k = 5000$			$k = 10,000$		
City	LB$-k$	UB$-k$	CPU Time (s)	LB$-k$	UB$-k$	CPU Time (s)	LB$-k$	UB$-k$	CPU Time (s)
Double Path + LS									
London	-0.4	1.0	0.034 ± 0.020	0.0	0.0	2.947 ± 3.071	0.0	0.0	23.795 ± 12.616
Melbourne	-0.7	0.6	0.014 ± 0.016	0.0	0.0	2.080 ± 2.586	0.0	0.0	13.234 ± 20.040
Amsterdam	-0.8	0.7	0.010 ± 0.005	0.0	0.0	0.983 ± 0.763	0.0	0.0	8.913 ± 8.031
New York	-2.1	1.5	0.005 ± 0.003	0.0	0.0	0.830 ± 0.862	0.0	0.0	7.708 ± 8.425
Kolkata	-4.3	4.6	0.003 ± 0.001	-0.2	0.2	0.622 ± 0.429	0.0	0.0	6.209 ± 5.448
Furthest + LS									
London	-0.8	0.5	0.010 ± 0.007	0.0	0.0	0.150 ± 0.169	0.0	0.0	0.641 ± 1.004
Melbourne	-0.9	2.4	0.006 ± 0.005	0.0	0.0	0.129 ± 0.130	0.0	0.0	0.676 ± 1.015
Amsterdam	-1.2	1.3	0.004 ± 0.002	0.0	0.0	0.071 ± 0.074	0.0	0.0	0.217 ± 0.212
New-York	-8.6	2.3	0.003 ± 0.002	0.0	0.0	0.061 ± 0.063	0.0	0.0	0.146 ± 0.113
Kolkata	-16.1	11.6	0.002 ± 0.001	-0.1	0.1	0.060 ± 0.058	-0.1	0.1	0.246 ± 0.248
Furthest + LS using BFS									
London	-0.9	0.5	0.004 ± 0.003	0.0	0.0	0.053 ± 0.046	0.0	0.0	0.254 ± 0.234
Melbourne	-0.7	2.0	0.003 ± 0.002	0.0	0.0	0.044 ± 0.041	0.0	0.0	0.235 ± 0.324
Amsterdam	-1.9	1.1	0.002 ± 0.001	0.0	0.0	0.023 ± 0.017	0.0	0.0	0.084 ± 0.092
New-York	-5.2	2.7	0.002 ± 0.001	0.0	0.0	0.022 ± 0.019	0.0	0.0	0.076 ± 0.048
Kolkata	-15.4	13.5	0.001 ± 0.001	-0.2	0.2	0.023 ± 0.020	-0.1	0.1	0.088 ± 0.074
Double Path + LS (Remove degree-1 vertices)									
London	-0.5	1.6	0.021 ± 0.012	0.0	0.0	1.751 ± 1.314	0.0	0.0	19.726 ± 16.717
Melbourne	-0.6	0.6	0.009 ± 0.009	0.0	0.0	1.627 ± 1.854	0.0	0.0	8.397 ± 9.301
Amsterdam	-1.1	1.1	0.007 ± 0.003	0.0	0.0	0.541 ± 0.338	0.0	0.0	4.265 ± 3.009
New York	-1.4	1.2	0.003 ± 0.002	0.0	0.0	0.511 ± 0.459	0.0	0.0	6.005 ± 4.643
Kolkata	-8.6	5.6	0.002 ± 0.001	-0.2	0.2	0.376 ± 0.228	-0.1	0.1	3.169 ± 1.899
Furthest + LS (Remove degree-1 vertices)									
London	-1.2	0.4	0.008 ± 0.007	0.0	0.0	0.119 ± 0.097	0.0	0.0	0.313 ± 0.275
Melbourne	-1.4	1.6	0.005 ± 0.004	0.0	0.0	0.096 ± 0.098	0.0	0.0	0.463 ± 0.745
Amsterdam	-1.5	1.0	0.003 ± 0.002	0.0	0.0	0.062 ± 0.067	0.0	0.0	0.264 ± 0.292
New-York	-5.4	2.0	0.003 ± 0.002	0.0	0.0	0.053 ± 0.052	0.0	0.0	0.127 ± 0.095
Kolkata	-16.3	11.7	0.002 ± 0.001	-0.1	0.2	0.053 ± 0.048	-0.1	0.1	0.244 ± 0.302
Furthest + LS using BFS (Remove degree-1 vertices)									
London	-1.3	0.5	0.004 ± 0.003	0.0	0.0	0.041 ± 0.037	0.0	0.0	0.188 ± 0.178
Melbourne	-2.6	1.7	0.003 ± 0.002	0.0	0.0	0.030 ± 0.028	0.0	0.0	0.153 ± 0.210
Amsterdam	-2.7	1.5	0.002 ± 0.001	0.0	0.0	0.017 ± 0.012	0.0	0.0	0.075 ± 0.078
New-York	-7.4	2.7	0.002 ± 0.001	0.0	0.0	0.024 ± 0.023	0.0	0.0	0.064 ± 0.046
Kolkata	-16.5	13.2	0.001 ± 0.001	-0.1	0.2	0.023 ± 0.019	0.0	0.0	0.083 ± 0.108

Table 4: Accuracy and speed of the LS heuristic on five cites (using circuits). Each figure is a mean across 50 runs using randomly selected source vertices within 1 km of the city centre.

City	$k = 1000$			$k = 5000$			$k = 10,000$		
	LB$-k$	UB$-k$	CPU Time (s)	LB$-k$	UB$-k$	CPU Time (s)	LB$-k$	UB$-k$	CPU Time (s)
Double Path + LS									
London	-0.5	1.1	0.039 ± 0.023	0.0	0.0	2.961 ± 3.235	0.0	0.0	20.708 ± 11.396
Melbourne	-0.6	0.7	0.017 ± 0.019	0.0	0.0	2.133 ± 2.749	0.0	0.0	13.282 ± 20.467
Amsterdam	-0.8	0.7	0.011 ± 0.006	-0.1	0.0	1.036 ± 0.791	0.0	0.0	10.013 ± 8.914
New-York	-2.7	2.1	0.005 ± 0.003	0.0	0.0	0.770 ± 0.813	0.0	0.0	6.241 ± 6.614
Kolkata	-5.4	5.7	0.003 ± 0.001	-0.4	0.5	0.706 ± 0.451	-0.1	0.1	5.999 ± 5.260
Furthest + LS									
London	-0.7	2.4	0.012 ± 0.008	0.0	0.0	0.319 ± 0.284	0.0	0.0	1.773 ± 1.734
Melbourne	-1.3	2.7	0.009 ± 0.007	0.0	0.0	0.295 ± 0.249	0.0	0.0	0.937 ± 1.184
Amsterdam	-1.6	1.2	0.006 ± 0.003	0.0	0.0	0.105 ± 0.076	0.0	0.0	0.678 ± 0.637
New-York	-9.3	9.9	0.004 ± 0.002	0.0	0.0	0.115 ± 0.099	0.0	0.0	0.355 ± 0.514
Kolkata	-17.4	13.8	0.002 ± 0.001	-0.1	0.1	0.089 ± 0.073	-0.3	0.2	0.422 ± 0.401
Furthest + LS using BFS									
London	-0.7	2.5	0.005 ± 0.003	0.0	0.0	0.059 ± 0.042	0.0	0.0	0.449 ± 0.443
Melbourne	-1.1	2.5	0.004 ± 0.003	0.0	0.0	0.085 ± 0.077	0.0	0.0	0.304 ± 0.349
Amsterdam	-2.2	1.2	0.003 ± 0.002	0.0	0.0	0.046 ± 0.035	0.0	0.0	0.187 ± 0.196
New-York	-6.7	9.6	0.002 ± 0.001	0.0	0.0	0.043 ± 0.037	0.0	0.0	0.108 ± 0.119
Kolkata	-17.4	13.8	0.002 ± 0.001	-0.2	0.3	0.033 ± 0.024	-0.1	0.1	0.142 ± 0.170
Double Path + LS (Remove degree-1 vertices)									
London	-0.6	0.6	0.024 ± 0.013	0.0	0.0	1.814 ± 1.397	0.0	0.0	16.034 ± 14.259
Melbourne	-0.7	0.7	0.010 ± 0.010	0.0	0.0	1.529 ± 1.934	0.0	0.0	7.901 ± 8.866
Amsterdam	-1.6	1.5	0.008 ± 0.003	-0.1	0.0	0.588 ± 0.360	0.0	0.0	4.645 ± 3.107
New-York	-2.7	2.5	0.004 ± 0.002	0.0	0.0	0.529 ± 0.455	0.0	0.0	4.689 ± 3.725
Kolkata	-10.1	7.0	0.002 ± 0.001	-0.7	0.5	0.427 ± 0.247	-0.1	0.2	3.220 ± 1.916
Furthest + LS (Remove degree-1 vertices)									
London	-0.7	2.9	0.011 ± 0.007	0.0	0.0	0.215 ± 0.206	0.0	0.0	1.022 ± 1.003
Melbourne	-1.7	1.6	0.007 ± 0.006	0.0	0.0	0.238 ± 0.214	0.0	0.0	0.984 ± 1.098
Amsterdam	-2.2	1.4	0.005 ± 0.003	0.0	0.0	0.117 ± 0.086	0.0	0.0	0.484 ± 0.433
New-York	-12.3	9.1	0.003 ± 0.002	0.0	0.0	0.095 ± 0.087	0.0	0.0	0.311 ± 0.402
Kolkata	-20.6	13.6	0.002 ± 0.001	-0.2	0.1	0.067 ± 0.054	-0.1	0.1	0.371 ± 0.328
Furthest + LS using BFS (Remove degree-1 vertices)									
London	-0.6	2.4	0.005 ± 0.003	0.0	0.0	0.062 ± 0.053	0.0	0.0	0.283 ± 0.288
Melbourne	-1.2	1.4	0.003 ± 0.002	0.0	0.0	0.066 ± 0.062	0.0	0.0	0.289 ± 0.319
Amsterdam	-1.8	2.2	0.003 ± 0.001	-0.1	0.0	0.034 ± 0.030	0.0	0.0	0.151 ± 0.150
New-York	-9.2	9.1	0.002 ± 0.001	0.0	0.0	0.040 ± 0.032	0.0	0.0	0.109 ± 0.127
Kolkata	-20.0	14.3	0.001 ± 0.001	-0.3	0.2	0.027 ± 0.024	-0.1	0.1	0.118 ± 0.110

Table 5: Accuracy and speed of the LS heuristic on five cites (using cycles). Each figure is a mean across 50 runs using randomly selected source vertices within 1 km of the city centre.

4 Random Graphs

The figures in this section correspond to the results given in Section 4.4 of the above paper. Results for cycles are included in addition to those of circuits.

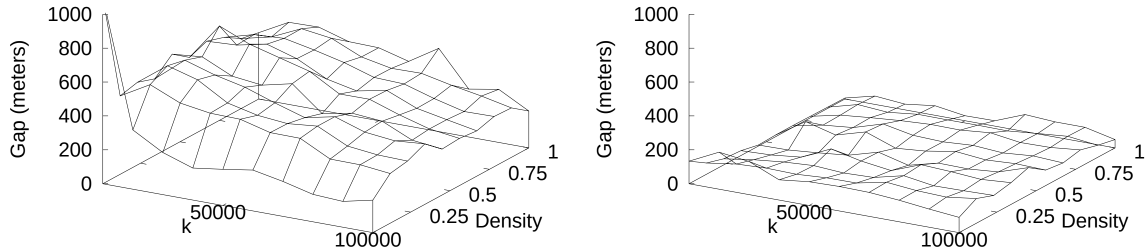

Figure 11: Accuracy of the local search algorithms for circuits, using random graphs of varying densities and different values of k. Accuracy is measured using the difference between the lower and upper bound, averaged across 20 instances. The first chart shows the local search algorithm based on BFS; the second shows the algorithm augmented with a second step of local search using shortest paths.

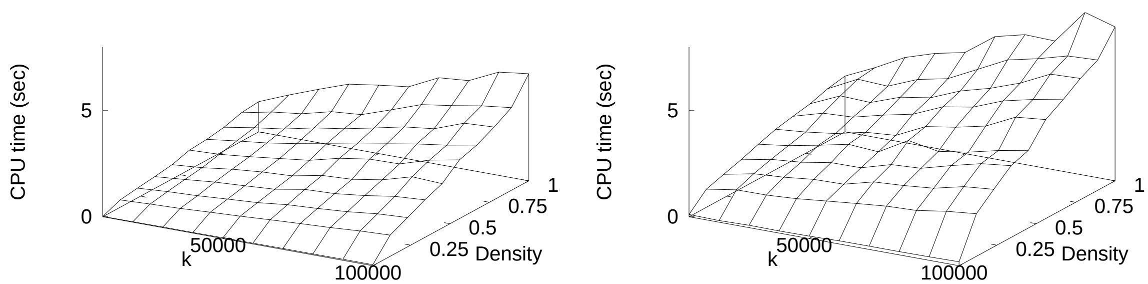

Figure 12: Mean execution times of the local search algorithms using random graphs of varying densities, for different values of k. Other details are the same as Figure 11.

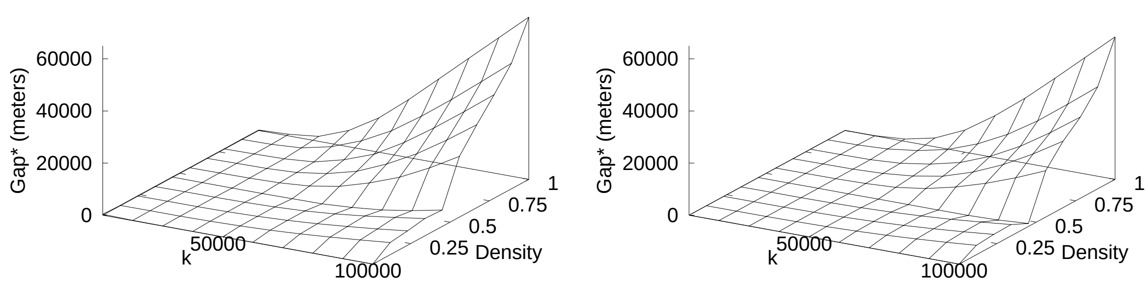

Figure 13: Accuracy of the local search algorithms for cycles, using random graphs of varying densities and different values of k. Accuracy is measured using the difference between the lower and upper bound, averaged across 20 instances. The first chart shows the local search algorithm based on BFS; the second shows the algorithm augmented with a second step of local search using shortest paths.

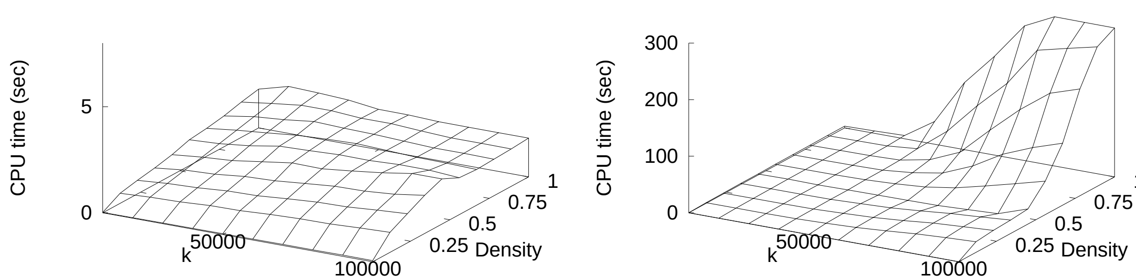

Figure 14: Mean execution times of the local search algorithms using random graphs of varying densities, for different values of k. Other details are the same as Figure 13.