Apptopinv
User's Guide

Timo Poranen
Department of Computer Sciences
P.O. Box 607
FIN-33014 University of Tampere, Finland
tp@cs.uta.fi
http://www.cs.uta.fi/~tp/apptopinv/

21st October 2003

Abstract

The maximum planar subgraph, maximum outerplanar subgraph, the thickness and outerthickness of a graph are all NP-complete optimization problems. *Apptopinv* is a program that contains different heuristic algorithms for these four problems: algorithms based on Hopcroft-Tarjan planarity testing algorithm, the spanning-tree heuristic and various algorithms based on the cactus-tree heuristic. *Apptopinv* contains also a simulated annealing algorithm that can be used to improve the solutions obtained from other heuristics. Most of the heuristics have also a greedy version.

We have implemented graph generators for complete graphs, complete k-partite graphs, complete hypercubes, random graphs, random maximum planar and outerplanar graphs and random regular graphs. *Apptopinv* supports three different graph file formats.

Apptopinv is written in C++ programming language for Linux-platform and GCC 2.95.3 compiler. To compile the program, a commercial LEDA algorithm library (version 4.3 or newer) is needed.

Contents

1 Introduction **2**
- 1.1 What is *apptopinv*? . . . 2
- 1.2 Why? . . . 2
- 1.3 History . . . 3
- 1.4 Web page and updates . . . 3
- 1.5 User rights . . . 3

2 Compilation **4**
- 2.1 Distributed files and compilation . . . 4
- 2.2 Simple example . . . 5

3 Usage **6**
- 3.1 Repetitions and the result file . . . 7
- 3.2 Supported optimization problems . . . 7
- 3.3 Different algorithms and solution optimization . . . 8
- 3.4 SA parameters and SA algorithm . . . 9
- 3.5 Supported graph formats . . . 10
- 3.6 Graph generators . . . 11
- 3.7 Saving random graphs . . . 11
- 3.8 Experiments and scripts . . . 12
- 3.9 Used bounds in SA . . . 13

4 More examples **15**
- 4.1 Is a graph planar or outerplanar? . . . 15
- 4.2 Cactus-tree heuristic example . . . 15
- 4.3 $Thick_{HT}$ heuristic for the graph thickness . . . 16
- 4.4 Greedy vs. running time . . . 16

5 Future plans **18**

6 Bugs and known features **19**

7 Acknowledgements **20**

Bibliography **20**

Chapter 1

Introduction

1.1 What is *apptopinv*?

Apptopinv (**app**roximation algorithms for the **top**ological **inv**ariants of graphs) is a software that can be used to approximate the following four topological invariants of graphs:

- maximum planar subgraph (MPS),
- maximum outerplanar subgraph (MOPS),
- thickness (TH) and
- outerthickness (OTH) of a graph.

Apptopinv contains implementations of different algorithms for these optimization problems. The program can be used also for testing the planarity and outerplanarity of a graph.

Apptopinv is written in C++ programming language for Linux platform. It can be compiled with GCC 2.95.3 compiler. You also need LEDA 4.3 (or newer) algorithm library [23] to compile it successfully. LEDA is commercial, but it has inexpensive academic versions for research purposes. *Apptopinv* uses the Hopcroft-Tarjan [18] planarity test algorithm that is implemented in LEDA.

1.2 Why?

The aim of this work is to improve the reproducibity [20] of the experimental comparison of various algorithms for MPS, MOPS, TH and OTH problems. This also makes it possible to implement new algorithms for these optimization problems with less work than starting from the scratch. This manual answers how *apptopinv* can be used, for detailed description of implemented algorithms and their analyses we give only references. An excellent survey concerning graph planarization is written by Liebers [24]. For the thickness we recommend a survey by Mutzel et al. [26]. Different algorithms and experiments for maximum outerplanar subgraph problem can be found from [29] and theoretical results for the outerhickness can be found from [14, 15, 16, 33].

1.3 History

Apptopinv was originally written for the graph thickness [25, 28] problem. Then it was modified to approximate the maximum planar subgraphs [32, 31] and maximum outerplanar subgraphs [29]. On summer 2003 these different programs were combined in a single program, since many data structures, algorithms and other functions were same for all these optimization problems. Finally, the thickness and outerthickness problems were approximated with new heuristics during the implementation phase of *apptopinv* [30].

1.4 Web page and updates

The newest version of *apptopinv* is downloadable from the url `http://www.uta.fi/~tp/apptopinv/`. Also this user's guide can be found from the same address. The current version number of *apptopinv* is 1.0.

1.5 User rights

You may use and modify the source codes of *apptopinv* freely for any research and testing purposes. But notice, you need LEDA version 4.3 or newer to compile and run *apptopinv*.

Chapter 2

Compilation

2.1 Distributed files and compilation

After unpacking *apptopinv-1.0.tar.gz* file, you should have files and directories listed in Table 2.1 in the same directory. Since *apptopinv* uses LEDA, check before compilation that your $LEDAROOT$ is set correctly (see LEDA's manual).

Table 2.1: *Apptopinv* v. 1.0 - distributed files

README.txt	usage information
Makefile	makefile for compilation
apptopinv.cpp	main file
sa_graph.h	headers
sa_graph.cpp	initialization, algorithms for MPS and MOPS
sa_alg.cpp	simulated annealing algorithms
sa_thickness.cpp	algorithms for the TH and OTH
bounds_th.cpp	upper and lower bounds for the TH and OTH
graph_gen.cpp	graph generators
graphs/	directory for different graphs
scripts/	directory for scripts
docs/	directory for documentation
sa_parameters.txt	parameter file for the SA algorithm

Now you can compile *apptopinv* with the GCC 2.95.3 compiler just by typing:

```
make
```

in the root directory. Now there should be one executable file (*a.out*) and seven object files (*apptopinv.o*, *graph_gen.o*, *sa_alg.o*, *sa_thickness.o*, *bounds_th.o*, *max_outerplanar.o* and *sa_graph.o*).

LEDA 4.3 did not work with GCC 3.0.* compiler in our platform (one processor AMD Athlon $1GHz$ computer with 256 megabytes memory (1992 BogoMips) running under Linux Mandrake 8.1), so it is possible that there are also other problems with LEDA and newer compilers.

2.2 Simple example

After compilation there should be a file named *a.out*. Try the following command to test how *apptopinv* works:

```
./a.out 1 res.txt -th -ca1 -no -c 100
```

Here is a sample printing what should happen if everything is fine:

```
[timo@drill ver1] ./a.out 1 res.txt -th -ca1 -no -c 20
Complete 20 -partite graph: K20
Vertices: 20 Edges: 190
thickness
exp1 cactus-tree heuristic
found subgraph of size: 37 / 190
found subgraph of size: 35 / 153
found subgraph of size: 33 / 118
found subgraph of size: 32 / 85
found subgraph of size: 27 / 53
found subgraph of size: 19 / 26
found subgraph of size: 7 / 7
Degree: 4 Euler : 4 <= thickness <= Alekseev: 4 / Dean: 9 / Halton: 10
 / Edge conj. :3+c
SOLUTION: 7
-----------------------
REPEATS: 1
WORST SOLUTION: 7
BEST SOLUTION: 7
AVERAGE SOLUTION: 7
Total running time: 0.01 seconds.
-----------------------
```

Next we look at this example a little closer. The first parameter, "1", is the number of distinct runs of the algorithm. The second parameter, "res.txt", is the name of the output file. The graph's name and the best found result (and average result, if more than one runs of the algorithm is used) are reported in the output file. The third parameter, "-th", chooses the optimization problem. Here -th stands for the *thickness*. The fourth parameter, "-ca1", specifies the used algorithm and the fifth parameter "-no" checks whether we want to optimize further the obtained solution by a simulated annealing algorithm. The sixth parameter describes the graph for which we are applying the algorithm. The last parameter, "-c 20", stands for the complete graph with 20 vertices.

After the command is entered, *apptopinv* starts. It prints data (graph, problem and used algorithm) to the stdout. Then it shows how large subgraphs are extracted from the input graph (K_{20}). Then the algorithm prints lower and upper bounds for the input graph (see also section "Used bounds in SA" for the origin of these bounds) and the obtained solution (SOLUTION 7). Next the number of repetitions, the best and worst solutions found, and the average of all solutions with the total running time is reported.

Chapter 3

Usage

To get help about the usage of the algorithm write *a.out* (same message is also printed if you give an illegal parameter):

```
[timo@drill ver1] ./a.out

Approximation algorithms for the topological invariants of graphs v. 1.0
University of Tampere, Department of Computer Sciences, 2003

Usage: a.out REPEATS RESULT_FILE PROBLEM ALGORITHM SA GRAPH_SOURCE [PARAMETERS]
  REPEATS              number of repetitions of the algorithm
  RESULT_FILE          file where results are written
  PROBLEM
       -mps            maximum planar subgraph
       -mops           maximum outerplanar subgraph
       -th             thickness
       -oth            outerthickness
       -planar        planar
       -oplanar        outerplanar
  ALGORITHM
       -e              empty set initialization
       -g              greedy initialization
       -ca             cactus-tree initialization
       -gca            greedy cactus-tree initialization
       -ca1            exp1. version of cactus-tree heuristic
       -gca1           greedy exp1. version of cactus-tree heuristic
       -ca2            exp2. version of cactus-tree heuristic (mps/th)
       -gca2           greedy exp2. version of cactus-tree heuristic (mps/th)
       -ht             dfs+HT planarity test initialization
       -st             spanning tree
  SA
       -sa             simulated annealing algorithm
       -no             only initial solution
  GRAPH_SOURCE
       -f FILENAME     graph is read from file. File in edge-list format
       -f1 FILENAME    graph is read from file. File in rmf format
       -fgml FILENAME  graph is read from file. File in gml format
       -c V1 V2 ... Vk complete k-partite graph with vertex set sizes V1,V2...
       -ch k           complete hypercube with 2^k vertices
```

```
      -pl V E         max plan graph with V nodes and E additional edges
      -opl V E        max oplan graph with V nodes and E additional edges
      -r V E          random graph with V vertices and E edges
      -rr V D         random D-regular graph with V vertices
  OPTIONAL SAVEOPTIONS
      -gml FILENAME   (random) graph is saved in gml-format
      -f FILENAME     (random) graph is saved in edge-list format

Examples: ./a.out 1 res.txt -th -ca1 -no -c 20
          ./a.out 1 res.txt -mps -ca -no -fgml graphs/random/r100_1508.gml
          ./a.out 1 res.txt -oth -g -sa -r 80 350 -f graphs/r80_350.gml
          ./a.out 1 res.txt -mops -gca -no -f graphs/data/g19.dat

Notice that the order of parameters matters! See also README.txt.
```

Next we look closer at the different parameter settings their combinations. Finally we give more examples on the usage. The order of command line parameters is as follows:

```
Usage: a.out REPEATS RESULT_FILE PROBLEM ALGORITHM SA GRAPH_SOURCE [PARAMETERS]
```

3.1 Repetitions and the result file

The number of repetitions of the used algorithm should be greater than or equal to 1, there is no upper limit. In RESULT_FILE the best, worst and average results of the run are reported. If RESULT_FILE already exists, new results are appended in the end of the file. A result file obtained with the command line

```
./a.out 100 res1.txt -th -ca1 -no -c 20
```

could be like this:

```
K20 7 6.86 6
```

It simply says that for the complete graph with 20 vertices the worst result over all runs is 7, the average result is 6.86 and the best result is 6. The problem and used algorithm is not mentioned here, since this format made it easier to analyze and use the results. It is very easy to modify the source code (only file *apptopinv.cpp* needs modifications) to get additional information in the result file.

3.2 Supported optimization problems

Apptopinv can be used to approximate the following four topological invariants of graphs:

- maximum planar subgraph problem (-mps),
- maximum outerplanar subgraph problem (-mops),
- thickness (-th) and
- outerthickness (-oth) of a graph.

It is possible to check whether the input graph is planar or outerplanar by using the following parameters instead of supported problems:

- -planar to test planarity, and
- -oplanar to test outerplanarity.

Here is an example to test the outerplanarity of a graph:

```
./a.out 1 rest.txt -oplanar -no -no -f graphs/data/g3.dat
```

The problem type is -planar or -oplanar. Algorithm type is set -no and simulated annealing is set -no. Graph source options can be used normally (see the following sections for details). The result file and number of repeats are still needed in the parameter list. Planarity test is performed only once, and no information about the run is written in the result file (modify file *appropinv.cpp*, if you like to change this).

Basically, it is possible to add other optimization problems, but depending on your problem, there might be more or less difficulties. These four problems are very similar, although in problems MPS and MOPS we try to maximize and in problems TH and OTH the aim is to minimize the solutions.

3.3 Different algorithms and solution optimization

Next we describe the use of different algorithms. In Table 3.1 there is a list of heuristics that can be used with different problems. We have also listed references for the origin of the algorithm and reported experiments. Also some remarks on the running time (time) and relative performance (sol'n) of algorithms are reported. There might be graphs for which the suggestions are completely wrong.

Table 3.1: Supported algorithm and problem types.

Algorithm	MPS	MOPS	TH	OTH	time	sol's	references
-g -sa/-no	x	x	x	x	slow	average/poor	[8, 29]
-ca -sa/-no	x	x	x	x	fast	good	[7, 29, 30]
-gca -sa/-no	x	x	x	x	slow	good	[29, 29, 30]
-ca1 -sa/-no	x	x	x	x	fast	very good	[31, 29, 30]
-gca1 -sa/-no	x	x	x	x	slow	good/best	[29]
-ca2 -sa/-no	x		x		fast	good?	[31, 30]
-gca2 -sa/-no	x		x		slow	good?/average	[30]
-ht -sa/-no	x	x	x	x	slow	good	[21, 25, 26, 8, 9]
-st -sa/-no	x	x	x	x	fast	poor	[10, 30]
-e -sa	x	x			slow	average/good	[28, 32, 29]

After you choose an algorithm to get initial solution, you have to decide whether you try to optimize your solution with simulated annealing (-sa / -no). Usually simulated annealing (SA) improves the solutions (this depends of course

on the used annealing parameters and the input graph). The running time of the SA is very slow for large graphs.

If you take only the initial solution, ca, ca1, ca2 and st are very fast even if you consider large graphs. Greedy algorithms (g, gca, gca1, gca2 and ht) are slow in general for large instances, but very useful for small and sparse graphs (depends also on the problem type).

All algorithms are randomized by choosing vertices and edges randomly whenever it is possible.

3.4 SA parameters and SA algorithm

The parameters for the simulated annealing are read from the file *sa_parameters.txt*. Only the first four lines of this file are read. The first value is the initial temperature (t0), the second value is the frozen temperature (t1), the third value is the cooling ratio (alpha) and the last value is the equilibrium detection rate (innerloop). Here is a sample parameter file. Values are taken from experiments reported in [29] for the maximum outerplanar subgraph problem.

```
[timo@drill ver1] more sa_parameters.txt
t0        0.25
t1        0.20
alpha     0.999
innerloop 5

//Only first four lines are read by sa algorithm
// inner-loop parameters:
// innerloop 0  - vertices
//           1  - 2*vertices
//           2  - vertices*vertices
//           3  - vertices*vertices/2
//           4  - edges/2
//           5  - edges
//           6  - 2*edges
```

The initial and frozen temperatures should be greater than zero and the cooling ratio should be more than 0 and less than 1. There are seven possible parameters for the equilibrium detection rate. Different options are listed in the previous parameter file.

Usually many experiments are needed to find out good cooling parameters. General guidelines on how good parameters can be detected are given in [2, 19]. More specific examples related on *apptopinv* are reported in [29, 28, 32]

If simulated annealing is used, optimization starts after the construction of the initial solution. Simulated annealing prints information on how the current solution changes during cooling process. Below is such a printing example for the graph thickness (printouts for outerhickness are similar).

```
:
Simulated annealing algorithm started
Reading parameter file!
:
58 / 105 subgraphs: 4 smallest: 2 dev: 14.0446
59 / 105 subgraphs: 4 smallest: 2 dev: 14.0268
```

```
60 / 105 subgraphs: 4 smallest: 2 dev: 14.0268
61 / 105 subgraphs: 4 smallest: 2 dev: 14.0089
62 / 105 subgraphs: 4 smallest: 2 dev: 14.0089
:
```

The leftmost numbers are the number of iterations before reaching equilibrium detection rate (58 / 105). Third number, *subgraphs: 4*, shows the current solution, *smallest: 2* is the size of the smallest subset (if this gets zero, the number of subsets decreases to 3) and *dev: 14.0446* shows the standard deviation of the sizes of planar subgraphs. The number of subgraph can only decrease.

The printings for the maximum planar and outerplanar subgraph problems are slightly different:

```
[timo@drill ver1]./a.out 1 res.txt -mps -e -sa -f graphs/data/g1.dat
graph: graphs/data/g1.dat
Vertices: 10 Edges: 22
maximum planar subgraph problem
Empty set initialization
Simulated annealing algorithm started
Reading parameter file!
1 / 22 current: 1 best: 1
2 / 22 current: 2 best: 2
3 / 22 current: 3 best: 3
:
```

Also the leftmost number is the number of iterations before reaching the equilibrium detection rate (second number). Current solution is *current: 1* and the best solution obtained so far is *best: 1*. Since empty-set initialization is used, optimization starts from an empty subgraph. The current solution and best found solution are printed since the current solution may decrease during the annealing process.

Descriptions of the simulated annealing algorithm and suitable cooling parameters for the maximum planar subgraph and maximum outerplanar subgraph problems can be found from [32, 29] and for the thickness from [28].

3.5 Supported graph formats

Apptopinv can read three different file formats. It supports Graph Modelling Language (-fgml) format [13], that is also used in LEDA [23]. The other two formats are edge-list format (-f) used in the experiments of Resende and Ribeiro [34] and rmf-format (-f1) used by Petit [27] for the optimal linear arrangement problem. See Table 3.2 for a list of these file formats and their references. Methods for different graph formats are implemented in file *apptopinv.cpp*.

Table 3.2: Supported file formats.

Parameter	Format description
-fgml	GML [23, 13]
-f	"edge-list" (see source codes by Resende and Ribeiro [34])
-f1	rmf (see source codes and graphs used by Petit [27])

3.6 Graph generators

Apptopinv can generate different graphs for testing purposes. Implementations of graph generators can be found from the file *graph_gen.cpp*.

Complete graphs can be generated with parameters -c V, where V is the number vertices. Complete k-partite graphs can be generated with parameters -c V1 V2 .. Vk, where Vi is the number of vertices in set i. To generate complete hypercubes, use parameter -ch k to get hypercube with 2^k vertices. If $k \geq 12$, then the construction of the hypercube gets very slow.

There are four different types of random graph generators that are implemented. The first random graph class is the class of "normal random graphs", where each edge has the same probability to exist. Using parameter -r V E a random graph with V vertices and E edges is generated using LEDA's function [23] *random_graph(G,V,E,true,true,true)*.

The second class is the class of random maximum planar graphs. It is also possible to randomly add edges that violate planarity for the constructed maximum planar graph. Using parameters *-pl V E* a random maximum planar graph with V vertices and $3V - 6$ edges is generated first and then E additional edges randomly chosen edges that violate planarity are added. If E is set to be 0, no planarity violating edges is added. Third, to construct random maximum outerplanar graphs, use parameters *-opl V E*.

The fourth class is the class of random regular graphs. We have implemented an Las Vegas algorithm given by Steger and Wormald [36]. It can be used with parameters *-rr V D*, where V is the number of vertices and D is the degree of the graph. The algorithm runs very fast in practice.

Table 3.3: Graph generators.

parameter	graph description
-c V	complete graph with V vertices
-c V1 V2 ... Vk	complete k-partite graph sets V1 V2 ... Vk
-ch k	complete hypercube with 2^k vertices
-r V E	random graph with V vertices and E edges [23]
-pl V E	random planar graph with E edges that violate planarity
-opl V E	random outerplanar graph with E edges that violate outerplanarity
-rr V D	random D-regular graph with V vertices [36]

3.7 Saving random graphs

It is possible to save all randomly generated graphs by giving the saving format and the name of the target file after graph generating parameters. File is saved in edge-list (-f) or gml (-gml) format.

If you generate random graphs and you like to make some experiments with them, it is a good idea to generate them first and then save. Now you can use -f or -fgml option to load it. See the following example for saving random graphs.

```
[timo@drill ver1] ./a.out 1 res.txt -mops -ca1 -no -opl 20 10 -gml gopl1.gml
creating maximum outerplanar graph...
```

```
Maximal outerplanar graph with 20 vertices and 10 edges that violates outerplanarity created
Vertices: 20 Edges: 47
maximum outerplanar subgraph problem
exp1 cactus-tree heuristic
SOLUTION: 35
graph file saved in file gopl1.gml
-----------------------
REPEATS: 1
WORST SOLUTION: 35
BEST SOLUTION: 35
AVERAGE SOLUTION: 35
Total running time: 0.08 seconds.
-----------------------
[timo@drill ver1] ./a.out 1 res.txt -mops -ca1 -no -fgml gopl1.gml
graph: gopl1.gml
Vertices: 20 Edges: 47
maximum outerplanar subgraph problem
exp1 cactus-tree heuristic
SOLUTION: 36
-----------------------
REPEATS: 1
WORST SOLUTION: 36
BEST SOLUTION: 36
AVERAGE SOLUTION: 36
Total running time: 0 seconds.
-----------------------
```

In this example a random graph with 20 vertices and 47 $(2 * 20 - 3 + 10)$ edges containing a maximum outerplanar subgraph with 37 $(2 * 20 - 3)$ edges is constructed and saved in file *gopl1.gml*. Then this new graph is loaded with option *-fgml gopl1.gml*. In this example the second try gave result 36 instead of 35 as in the first run (ca1 is a randomized algorithm).

We have noticed that it is preferable to use -f format for large graphs instead of gml, since the reading of a gml-graph takes much longer time.

3.8 Experiments and scripts

There are many possible ways to automatize computational experiments. Next we describe a possible way for performing many testruns with *apptopinv*. This example works only with Linux platform.

The basic idea is to create a script file that tells to computer what to do and what information about testruns is saved. All runs are performed as background processes and for the timing it is better to use *time* command (this can save running time correctly, or almost correctly, even if you have great number of background processes running at the same time). The timing mechanism programmed inside *apptopinv* fails if many background processes are running at the same time. See the script-file *test1* below for an example on how experiments can be automatized.

```
[timo@drill ver1] more test1
#!/bin/sh
echo g17 > times.txt
```

```
time ./a.out 25 tmpres/resg17.txt -oth -ca1 -no -f graphs/data/g17.dat 2>>times.txt
echo g18 >> times.txt
time ./a.out 25 tmpres/resg18.txt -oth -ca1 -no -f graphs/data/g18.dat 2>>times.txt
echo g19 >> times.txt
time ./a.out 25 tmpres/resg19.txt -oth -ca1 -no -f graphs/data/g19.dat 2>>times.txt
```

This script performs 25 runs of the heuristic ca1 for graphs g17, g18 and g19. The problem is OTH. All running times are saved in file *times.txt* and results for graph g17 are saved in file *tmpres/resg17.txt*, results for graph $g18$ are saved in file *tmpres/resg18.txt* and so forth. Here is a sample *times.txt* file.

```
[timo@drill ver1] more times.txt
g17
0.96user 0.05system 0:01.06elapsed 95%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (785major+5515minor)pagefaults 0swaps
g18
1.03user 0.07system 0:01.30elapsed 84%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (785major+5519minor)pagefaults 0swaps
g19
1.52user 0.13system 0:01.74elapsed 94%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (785major+7991minor)pagefaults 0swaps
```

The user time is the exact running time of your experiment. To get the average time of one run, it is enough to divide the user time by the number of repeats. For example, the average time of one run of heuristic ca1 for graph g19 is $1.52/25 = 0.0608$ seconds.

Here is a sample result file:

```
[timo@drill ver1] more tmpres/resg19.txt
graph: graphs/data/g19.dat 7 6.88 6
```

This file shows that for graph *graphs/data/g19.dat* the worst result is 7, average result is 6.92 and the best found result is 6. Using files *times.txt* and result files it is quite simple to produce different diagrams, tables and figures on experiments. To modify the contents of the result files only source file *apptopinv.cpp* need changes.

3.9 Used bounds in SA

In the SA algorithm we have added a test to recognize the optimal solutions. It is well known that a maximum planar subgraph of n-vertex graph could contain at most $3n-6$ edges and the maximum outerplanar subgraph could contain at most $2n-3$ edges. If the corresponding bound is reached for the MPS or MOPS problems, cooling process is stopped.

If the thickness or outerthickness is approximated using simulated annealing, then the optimal solutions for the complete graphs, complete bipartite graphs and hypercubes are recognized by using the results given in [1, 3, 4, 22] for the thickness and [15, 16] for the outerthickness.

If the thickness is approximated, the following lines are printed:

```
[timo@drill ver1] ./a.out 1 tmpres/res.txt -oth -ca1 -no -c 20
:
```

```
Degree: 4 Euler : 4 <= thickness <= Alekseev: 4 / Dean: 9 / Halton: 10
 / Edge conj. :3+c
SOLUTION: 7
:
```

Also these lower and upper bounds are derived using the previously referred results with the lower and upper bounds given in [11, 17, 37, 38].

If the outerthickness is approximated, bounds are taken from the results given in [33, 11, 17]. This is illustrated in the following example.

```
[timo@drill ver1] ./a.out 1 tmpres/res.txt -oth -ca1 -no -c 20
:
Degree: 5 Euler : 6 <= outerthickness <= Guy: 6 / Halton: 10 / Dean: 9
 / Edge conj. : 4+c
SOLUTION: 7
:
```

Chapter 4

More examples

4.1 Is a graph planar or outerplanar?

In the first example we just check whether input graph is planar or outerplanar:

```
[timo@drill ver1] ./a.out 1 rest.txt -planar -no -no -f graphs/data/g3.dat
graph: graphs/data/g3.dat
Vertices: 10 Edges: 24
Input graph is planar
[timo@drill ver1] ./a.out 1 rest.txt -oplanar -no -no -f graphs/data/g3.dat
graph: graphs/data/g3.dat
Vertices: 10 Edges: 24
Input graph is not outerplanar
```

4.2 Cactus-tree heuristic example

The second example approximates maximum planar subgraph of a graph loaded from the file *graphs/random/r100_1508.gml* using the cactus-tree heuristic. The solution is not optimized. To get exact running time of the algorithm *time* command is used.

```
[timo@drill ver1] time ./a.out 1 res.txt -mps -ca -no -fgml graphs/random/r100_1508.gml
graph: graphs/random/r100_1508.gml
Vertices: 100 Edges: 1508
maximum planar subgraph problem
Cactus-tree heuristic
SOLUTION: 147
-----------------------
REPEATS: 1
WORST SOLUTION: 147
BEST SOLUTION: 147
AVERAGE SOLUTION: 147
Total running time: 0.12 seconds.
-----------------------
0.12user 0.03system 0:00.15elapsed 96%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (791major+1027minor)pagefaults 0swaps
```

4.3 $Thick_{HT}$ heuristic for the graph thickness

The third example approximates the thickness of K_{15}. To get the initial solution, depth-first-search with Hopcroft-Tarjan planarity test is used (-ht). This heuristic is similar to $Thick_{HT}$ heuristic reported in [9, 26]. The initial solution is 4 and it is decreased to 3 (this is the optimal solution) by using simulated annealing.

```
[timo@drill ver1] time ./a.out 1 res.txt -th -ht -sa -c 15
Complete 15 -partite graph: K15
Vertices: 15 Edges: 105
thickness
dfs+(outer)planarity test initialization
found subgraph of size: 39 / 105
found subgraph of size: 32 / 66
found subgraph of size: 22 / 34
found subgraph of size: 12 / 12
Simulated Annealing algorithm started
Reading parameter file!
1 / 105 subgraphs: 4 smallest: 12 dev: 10.2103
2 / 105 subgraphs: 4 smallest: 12 dev: 10.2103
3 / 105 subgraphs: 4 smallest: 12 dev: 10.2103
:
:
:
48 / 105 subgraphs: 4 smallest: 1 dev: 14.6202
49 / 105 subgraphs: 4 smallest: 1 dev: 14.6202
50 / 105 subgraphs: 4 smallest: 1 dev: 14.6544
51 / 105 subgraphs: 3 smallest: 33 dev: 1.63299
Algorithm ended. Got 3 sets
Degree: 3 Euler : 3 <= thickness <= Alekseev: 3 / Dean: 7 / Halton: 7
 / Edge conj. :2+c
SOLUTION: 3
-----------------------
REPEATS: 1
WORST SOLUTION: 3
BEST SOLUTION: 3
AVERAGE SOLUTION: 3
Total running time: 1.69 seconds.
-----------------------
1.68user 0.05system 0:02.59elapsed 66%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (828major+329minor)pagefaults 0swaps
```

4.4 Greedy vs. running time

Finally we give an example on how the usage of the greedy heuristic affects the running time and solution quality. First we approximate the maximum outerplanar subgraph of a given graph by using ca1 [31].

```
[timo@drill ver1] time ./a.out 1 res.txt -mops -ca1 -no -f1 graphs/opt/c1y.rmf
Reading graph from file graphs/opt/c1y.rmf
graph: graphs/opt/c1y.rmf
```

```
Vertices: 828 Edges: 1749
maximum outerplanar subgraph problem
exp1 cactus-tree heuristic
SOLUTION: 1186
-----------------------
REPEATS: 1
WORST SOLUTION: 1186
BEST SOLUTION: 1186
AVERAGE SOLUTION: 1186
Total running time: 0.06 seconds.
-----------------------
0.10user 0.00system 0:00.09elapsed 101%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (785major+369minor)pagefaults 0swaps
```

The running time is quite short (0.1 seconds) since the algorithm is linear and no planarity test is performed. This algorithm can be made greedy by adding as many new edges as possible to the graph obtained using ca1 heuristic. Now planarity test is performed 563 times (outerplanarity can be tested using planarity testing algorithm [39]).

```
[timo@drill ver1] time ./a.out 1 res.txt -mops -gca1 -no -f1 graphs/opt/c1y.rmf
Reading graph from file graphs/opt/c1y.rmf
graph: graphs/opt/c1y.rmf
Vertices: 828 Edges: 1749
maximum outerplanar subgraph problem
Greedy exp1 cactus-tree heuristic
SOLUTION: 1302
-----------------------
REPEATS: 1
WORST SOLUTION: 1302
BEST SOLUTION: 1302
AVERAGE SOLUTION: 1302
Total running time: 94.53 seconds.
-----------------------
94.43user 0.14system 1:39.75elapsed 94%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (807major+491minor)pagefaults 0swaps
```

The running time of gca1 was approximately 940 times the running time of the ca1 heuristic! To test the planarity of a planar graph (*graph/opt/crack.rmf*) with 10240 vertices and 30380 edges it takes 20.81 seconds using LEDA, so any algorithm described in this manual that performs planarity test numerous times (all greedy algorithms and simulated annealing) may have very long running time.

Chapter 5

Future plans

The current version of *apptopinv* uses LEDA, so one goal is to have a version that does not need any commercial algorithm library. To drop LEDA out, planarity test algorithm should be implemented. The algorithms of Shih and Hsu [35] and Boyer and Myrvold [5, 6] seem to be good choices. Also new implementations of data structures and algorithms for sets and their manipulation are needed. Since *apptopinv* was originally a combination of different (but very closely related and similar) programs, some of the data structures and functions of this version can be implemented and designed in better way. The code is not very readable, and the documentation of functions and algorithms need tuning.

There exist two linear time algorithms [12, 35] that can be used to search a maximal planar subgraph in linear time. It would be very interesting to compare these algorithms against different cactus-tree heuristics for very large graphs.

Chapter 6

Bugs and known features

Probably *apptopinv* contains some bugs, so if you meet problems using this program, please report them to Timo Poranen by email (tp@cs.uta.fi). Also all kinds of comments are warmly welcome!

Next we give a list of known features of apptopinv:

- It takes much longer time to read a graph in gml format than in edge-list format (the size of the gml file is usually much bigger).
- All data that is written to results files is now appended in the end of the file. In the earlier versions old result file was overwritten.
- If a random graph is saved, then the old file is overwritten (if it exists).
- The annealing parameters are read every time when the SA is applied. Do not modify *sa_parameters.txt* -file between repeats.
- The maximum number of (outer)planar subsets for the thickness and outerthickness is limited to 1000 (if needed, change the statement "MAX_PARTITIONS=1000" in file *sa_alg.cpp*).
- It is not any more possible to save traces about the runs as in the earlier version (see for example [32]). In file *sa_alg.cpp* the source codes for saving traces are left, so you if you need them, you have to modify a little a bit the main file (*apptopinv.cpp*) and the algorithm files.

Chapter 7

Acknowledgements

Although *apptopinv* is mostly written by the author, Petri Vuorenmaa programmed the first version of this program in a research project at the University Tampere during autumn 1999 [25]. Thanks Petri!

Arto Viitanen gave me many useful hints concerning on timing the computer programs and he showed me a simple way to write scripts for large number of testruns. Thanks Arto!

Professor Erkki Mäkinen from the Department of Computer Sciences at the University of Tampere has encouraged me to finish this work. He has also read many drafts of research papers related on this work. Thanks Erkki!

This work was funded by the Tampere Graduate School in Information Science and Engineering (TISE) and supported by the Academy of Finland (Project 51528).

Bibliography

[1] V.B. Alekseev and V.S. Gonchakov. Thickness for arbitrary complete graphs. *Mat. Sbornik.*, 143:212–230, 1976.

[2] C. R. Aragon, D. S. Johnson, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Oper. Res.*, 3(39):378–406, 1991.

[3] L.W. Beineke and F. Harary. The thickness of the complete graph. *Can. J. Math*, 17:850–859, 1965.

[4] L.W. Beineke, F. Harary, and J.W. Moon. On the thickness of the complete bipartite graphs. *Proc. Camb. Phil. Soc.*, 60:1–5, 1964.

[5] J. Boyer and W. Myrvold. Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm. In *Proc. of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 140–146, 1999.

[6] J.M. Boyer, P.F. Cortese, M. Patrignani, and G. Di Battista. Stop minding your P's and Q's: Implementing a fast and simple DFS-based planarity testing and embedding algorithm. To appear in Proc. Graph Drawing: 11th International Symp. (GD'03), 2003.

[7] G. Călinescu, C. Fernandes, U. Finkler, and H. Karloff. A better approximation algorithm for finding planar subgraphs. *J. Algorithms*, 27(2):269–302, 1998.

[8] R. Cimikowski. An analysis of heuristics for the maximum planar subgraph problem. In *Proc. of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 322–331, 1995.

[9] R. Cimikowski. On heuristics for determining the thickness of a graph. *Info. Sci.*, 85:87–98, 1995.

[10] R. Cimikowski and D. Coppersmith. The sizes of maximal planar, bipartite planar, and outerplanar subgraphs. *Discr. Math.*, 149:303–309, 1996.

[11] J.P. Dean, A.M. Hutchinson and E.R. Scheinerman. On the thickness and arboricity of a graph. *J. Comb. Theory Ser. B*, 52:147–151, 1991.

[12] H. N. Djidjev. A linear algorithm for the maximal planar subgraph problem. In S.G. Akl, F. Dehne, J-R. Sak, and N. Santoro, editors, *Proc. 4th International Workshop on Algorithms and Datastructures*, volume 955 of *LNCS*, pages 369–380, 1995.

[13] GML - Graph Modelling Language. Available at `http://www.infosun.fmi.uni-passau.de/Graphlet/GML/`.

[14] R.K. Guy. *Combinatorics, London Math. Soc. Lecture Notes 13*, chapter Outerthickness and outercoarseness of graphs, pages 57–60. Cambridge University Press, 1974.

[15] R.K. Guy and R.J. Nowakowski. The outerthickness and outercoarseness of graphs I. The complete graph & the n-cube. In R. Bodendiek and R Henns, editors, *Topics in Combinatorics and Graph Theory: Essays in Honour of Gerhard Ringel*, pages 297–310. Physica-Verlag, 1990.

[16] R.K. Guy and R.J. Nowakowski. The outerthickness and outercoarseness of graphs II. The complete bipartite graph. In R. Bodendiek, editor, *Contemporary Methods in Graph Theory*, pages 313–322. B.I. Wissenchaftsverlag, 1990.

[17] J.H. Halton. On the thickness of graphs of given degree. *Info. Sci.*, 54:219–238, 1991.

[18] J. Hopcroft and R.E. Tarjan. Efficient planarity testing. *J. ACM*, 21:549–568, 1974.

[19] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Oper. Res.*, 6(37):865–892, 1989.

[20] D.S. Johnson. A theoretician's guide to the experimental analysis of algorithms. In M.H. Goldwasser, D.S. Johnson, and C.C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250, 2002.

[21] M. Jünger and P. Mutzel. Maximum planar subgraph and nice embeddings: Practical layout tools. *Algorithmica*, 16:33–59, 1996.

[22] M. Kleinert. Die Dicke des n-dimensionale Würfel-Graphen. *J. Comb. Theory*, 3:10–15, 1967.

[23] LEDA version 4.3 (commercial). Available at `http://www.algorithmic-solutions.com`.

[24] A. Liebers. Planarizing graphs - a survey and annotated bibliography. *JGAA*, 5(1):1–74, 2001. Available at `http://www.cs.brown.edu/publications/jgaa/accepted/01/Liebers01.5.1.ps.%gz`.

[25] E. Mäkinen, T. Poranen, and P. Vuorenmaa. A genetic algorithm for determining the thickness of a graph. *Info. Sci.*, 138:155–164, 2001.

[26] P. Mutzel, T. Odenthal, and M. Scharbrodt. The thickness of graphs: a survey. *Graphs Comb.*, 1998.

[27] J. Petit. *Layout Problems.* PhD thesis, Universitat Politècnica de Catalunya, 2001. Available at `http://www.lsi.upc.es/~jpetit/Publications/pdf/thesis.pdf`.

[28] T. Poranen. A simulated annealing algorithm for determining the thickness of a graph. Available at `http://www.cs.uta.fi/~tp/pub/max_planar_sa.ps`, October 2002.

[29] T. Poranen. Experiments on outerplanar subgraphs. Available at `http://www.cs.uta.fi/~tp/pub/max_outerplanar_sa.ps`, July 2003.

[30] T. Poranen. Fast algorithms for the thickness of a graph. Available at `http://www.cs.uta.fi/~tp/pub/fast_thickness.ps`, September 2003.

[31] T. Poranen. More triangles! a note on cactus-tree heuristic. Available at `http://www.cs.uta.fi/~tp/pub/triangles.ps`, July 2003.

[32] T. Poranen. A simple randomized algorithm for determining maximum planar subgraphs. Available at `http://www.cs.uta.fi/~tp/pub/thick.ps`, January 2003.

[33] T. Poranen and E. Mäkinen. Remarks on the outerthickness of a graph. Available at `http://www.cs.uta.fi/~tp/pub/othick.ps`, November 2002.

[34] M.G.C Resende and C.C. Ribeiro. A GRASP for graph planarization. *Networks*, 29:173–189, 1997. Available at `http://www.research.att.com/~mgcr/doc/gmpsg.ps.Z`.

[35] W-K. Shih and W-L. Hsu. A new planarity test. *Theor. Comput. Sci.*, 223:179–191, 1999.

[36] A. Steger and N.C. Wormald. Generating random regular graphs quickly. *Comb. Probab. and Comput.*, 8:377–396, 1999.

[37] O. Sýkora, L.A. Székely, and Vrt'o. A note on Halton's conjecture. To appear in Information Sciences, 2003.

[38] W. Wessel. Über die Abhängigkeit der Dicke eines Graphen von seinen Knotenpunktvalenzen. *Geometrie und Kombinatorik, 2. Kollog., Karl-Marx-Stadt, GDR*, 2(2):235–238, 1984.

[39] M. Wiegers. Recognizing outerplanar graph in linear time. In *Graph-Theoretic Concepts in Computer Science, International Workshop WG'86*, volume 246 of *LNCS*, pages 165–176, 1984.