0.1 Color and Clique: dfamx.c and dmclique.c

0.1.1 Who?

David Applegate and David Johnson wrote the C code.

0.1.2 Where?

The source code could be gotten by ftp from dimacs.rutgers.edu under the directory of pub/challenge/graph. And the test data could be found under pub/djs.

0.1.3 When?

The dfmax.c was created on October 22, 1983. And then, it was modified February 20, 1987 for checkpointing, and was modified August 1988 to find maximum independent sets, modified April 1993 for new data structures and bounding, and modified September 1993 for dimacs .b input.

As for the dmclique.c, it was first written on the same time – October 22, 1983. And then, it was modified February 20, 1987 for checkpointing, August 1988 to find maximum independent sets, Summer 1993 by David S. Johnson and David L. Applegate to speed up exhaustive part, and modified September 1993 to accept DIMACS input .b format.

0.1.4 Why?

To aid in comparisons between papers on the clique problem, the DIMACS Challenge Steering Committee is making available two benchmark programs, dfmax.c and dmclique.c.

Whether you are studying exact or approximate algorithms, dfmax.c can serve as a simple benchmark for calibrating your machine's speed relative to those of the other participants.

The code can also be used as a common point of reference for papers implementing exact clique algorithms. It is easily beaten on dense graphs (average degree exceeding $|V|/2$) but may be hard to beat on sparser graphs, especially random ones.

0.1.5 What?

The dfmax.c implements a simple-minded branch-and-bound algorithm very similar to that of Carraghan and Paradalos [Oper.Res.Lett. 9 (1990), 375-382].

And the dmclique.c is a variant on the simple "semi-exhaustive greedy" scheme for finding large independent sets used in the graph coloring algorithm XRLF described in Johnson, Aragon, McGeoch, and Schevon [Oper.Res. 39 (1991), 378-406], originally suggested by Matula and Johri.

Besides, nmclique.c is a new version of dmclique.c that prints out the best clique found.

0.1.6 How?

dmclique has four integer input parameters, SETLIM, TRIALNUM, CANDNUM, and ITERATIONS, of which the key parameters are SETLIM and CANDNUM. The algorithm proceeds as follows:

At any given time there is a set C (initially empty) of vertices in the current clique, and a set U consisting of all those remaining vertices that are adjacent to all members of C. If $|U| <=$ SETLIM, the branch-and-bound code of dfmax (without the initial C&P pre-ordering) is invoked to find the maximum clique in U, and this is added to C. Otherwise, the program samples CANDNUM vertices (with replacement) and chooses one with maximum degree (ties broken in favor of the earlier index). The chosen vertex is then added to C, U is updated, and the process is repeated. The entire process could be repeated TRIALNUM*ITERATIONS times.

Algorithm dmclique algorithm has much in common with the "GRASP" algorithm of Feo and Resende, except the latter sets SETLIM = 0 and uses a final local optimization phase to improve the clique found, typically with slightly better results than are obtained with dmclique. The value of dmclique as a benchmark code (as with dfmax) is that it represents what can be obtained by a (relatively) efficient implementation of the most straightforward of ideas.

0.1.7 Programming Language?

The two programs were written in C.

I am using cc to make the software.

0.1.8 Operating Environment?

The environment in which we are testing the program is Sun Sparc, SunOS 4.1.3.

0.1.9 Efficiency?

The software is not as efficient as what the author described in the results.dfmax and results.dmclique. They were offered by the author(s) record the results of running dfmax on the DIMACS benchmark graphs for clique, as well as selected random graphs with N from 100 to 2000 and p from .3 to .99 on SGI Challenge, which has 8 MIPS R4400 processors. The processors on this machine are in the same class as SPARC 10's and DEC Alphas, perhaps 2-4 times faster than a SPARC 2.

For example, it is said that brock200_1.clq.b is supposed to be read in 0.03 seconds and run in 381.19 seconds on the authors' system, but it took 0.10 user time to read it and

1643.33 seconds to execute on our system.

The test data in results.dfmax and results.dmclique could be obtained by ftp from dimacs.rutgers.edu under the directory of /pub/challenge/graph/benchmarks/clique and /pub/challenge/graph/benchmarks/volume. I have settled them under "test" subdirectory.

The test file with the largest size is the F1000.5.b under /test directory. It takes 760 min. to delete 46 vertices. Still running ...

0.1.10 Portability?

The dfmax.c can run on a variety of architectures. It assumes a 32-bit word length, but otherwise should be reasonably portable. It appears to compile successfully under both ANSI and non-ANSI C-compilers and has been run on a VAX-8550, various SGI machines, and SPARCstations. On an SGI Challenge computer (comparable to a DEC Alpha or a Sun SPARCstation 10), it can handle a 500 vertex, p=0.5 random graph in 3 minutes. A summary of results of the SGI Challenge for random graphs and the DIMACS benchmarks is in the file results.dfmax.

As with dfmax.c, dmclique.c should compile successfully on a variety of machines.

0.1.11 Availability?

Everyone could get by ftp from the address mentioned in the first two sections. There is no indication about the license of the software.

0.1.12 Documentation?

The documentation provided by the authors is the README file and two output files: results.dfmax and results.dmclique. But some information such as the directory where the test data is located are incorrect.

0.1.13 Robustness?

I have to say this software has excellent robustness because all the test files are binary data which are created by the author(s). And all of them could run well and offer a lot of useful information. Since I couldn't find the generator to convert the graph data (as described in Data Format segment) into binary files, I didn't create test data myself.

0.1.14 Data Formats?

Graphs in this directory are stored in compressed incidence matrix formats, with coordinate information appended to the representations of geometric graphs. The formats are as follows, with <cr> representing a carriage return character:

0) random:
  0 <cr> (matrix type, mattype)
  size <cr>
  density <cr>
  bitmap (integers of INT bits separated by spaces)

1) geometric:
  1 <cr>
  size <cr>
  max. distance <cr>
  bitmap
  coordinate pairs

dfmax is designed to take as input the compressed "clq.b" format used in the DIMACS benchmark directories. Usage: "dfmax filename".

There are five test data under the current directory: r100.5.b, r200.5.b, r300.5.b, r400.5.b, and r500.5.b.

There are also two files: results.dfmax and results.dmclique, that contain the results of running dfmax and dmclique, respectively, on the DIMACS benchmark graphs.

Some data could also be found at "/pub/challenge/graph/benchmarks/color" and "/pub /challenge/graph/benchmarks/clique"

0.1.15 Output and Visualization?

To indicate that dfmax is making progress (always useful in a potentially exponential time algorithm), it prints a line of output each time a vertex is eliminated from further consideration, giving the current biggest clique found and the current running time (assuming your computer's clock ticks 100 times per second). It also reports input time at the beginning of a run and outputs the clique at the end.

0.1.16 Ease of Interface

The defined constant NMAX in dfmax and dmclique, currently set to 2400, gives the maximum number of vertices that can be handled. If you increase this value, note that the constants MAX_NR_VERTICES and MAX_NR_VERTICESdiv8 must also be adjusted as indicated in the code. (The code was kludged together from at least three sources, and so does not set standards for programming style...) To run the code as an algorithm for finding maximum stable sets, place an "!" immediately before the word "bitmap" in the line beginning "#define edge". As the code currently stands, it implements the vertex pre-ordering scheme C&P. This can be turned off by commenting out the "#define REORDER" line.

0.1.17 Usefulness?

8 –