MTF — A computer program operating on maximal triangle–free graphs

Gunnar Brinkmann Stefan Brandt Thomas Harmuth

10. June 1998

Contents

1 Introduction

The program *MTF* generates maximal triangle–free graphs of an order given by the user. The algorithm is recursive: To produce the **m**aximal **t**riangle–**f**ree graphs (mtf–graphs) with n vertices, the program needs those with $n-1$ vertices. You can just save these mtf–graphs or operate on them. The program supports the following operations:

Local density: You can test if the mtf–graph has a certain *local density*. The local density of an mtf–graph G is tested on subgraphs with x vertices. If every subgraph of G with x vertices contains at least y edges, then G satisfies the condition "G has local density x y".

Ramsey numbers: You can test if a so–called *test graph* T (which is given by the user) is in the complement of the mtf–graph (i. e. the test graph is a subgraph of the complement of the mtf–graph). In this case the mtf–graph is called *Ramsey graph*. This test is a part

of the computation of the *Ramsey number* $R(K_3, T)$. So you can compute the Ramsey number $R(K_3, T)$ for an arbitrary test graph T, where K_3 is a triangle. Furthermore, you can save all Ramsey graphs with a prescribed vertex number.

2 Concepts

MTF generates mtf–graphs in a tree–like structure: It begins with the only mtf–graph with 3 vertices, which is the $K_3 - e$, a triangle with one edge missing. Whenever it finds an mtf–graph with n vertices, the program tries to add another vertex and several edges to obtain one or more mtf–graphs with $n + 1$ vertices. These graphs are the *successors* of the graph with n vertices. In general, each graph has more than one successor. The number of mtf–graphs grows exponentially with n.

MTF generates all mtf–graphs with a given number of vertices. However, there are some options which do not make it necessary to produce all graphs. Sometimes a result which is valid for one graph is also valid for all its successors, and so they do not need to be generated.

Whenever you run the program, *MTF* creates a logfile. All information that is printed on the screen is also written into this logfile. The name of this file depends on the options you use. The same holds for all other output files which are created by the program. The logfile includes the following information:

- the maximum number of vertices up to which mtf–graphs are generated
- the number of generated mtf–graphs, dependent on the number of vertices
- the execution time
- some statistics which is useful only for programmers who know the whole algorithm of the program

Furthermore, one or more graph output files are created. These files contain graph data. The code which is used is normally "*multi_code2_s_old*"[1]. However, it is also possible to use another code for output (see section "Options"). In general, the output files contain the generated mtf–graphs. For each vertex number there is an own output file. You can determine for which vertex numbers output files shall be created. If you use the option "*ramsey*", then the output files may not contain mtf–graphs but test graphs. See section "Options" for details.

The computations for which *MTF* is written may take days or even weeks. For this reason *MTF* writes a so–called *dump file* every 15 minutes (or even more frequently, if necessary). If the system hangs up, this dump file can be used to recover the process. And so the loss of time is at most 15 Minutes. If you run the program, it automatically searches for the dump file. The name of this dump file depends on the options you use so that no dump file deletes the one of another process. When you run the program with certain options for the first time, the respective dump file does not yet exist. The program then prints the message "`Could not find dump file 'xxxx'. I will proceed without.`" and starts the process right from the beginning.

MTF is able to read mtf–graphs from a file. For example, if you want to generate all mtf–graphs with 17 vertices, you can either start with the $K_3 - e$ or you can read in all mtf–graphs with 16 vertices, if you have already generated them. This saves the time you needed to generate the graphs with 16 vertices.

Sometimes a process is so huge that it would take too much time to run it on a single computer. So *MTF* provides the possibility of splitting up a process into independent subprocesses which can be run on different computers. As it was said before, the mtf–graphs are generated in a tree–like structure. Two graphs which are generated in the same depth of the tree have the

[1] See manual of the program *gconv* for the description of this code (Version 3/96 or newer).

same number of vertices. Let the graphs in one depth be numbered consecutively beginning with 0. You can now split up a process by doing the following: You choose a tree–depth called the *level* of the split. Then you determine the number of parts the process must be split into. This number is called the *mod* of the split. Finally, you choose the part of the process which must be executed. The parts are numbered $0, \ldots, mod-1$ and are called *classes*. The program now does the following: Up to the given level, it generates all mtf–graphs. But when it reaches this level, it takes only every *mod*-th graph to generate its successors. For example, if we divide a process into five parts, part 0 takes the graphs $0, 5, 10, \ldots$, part 1 takes the graphs $1, 6, 11, \ldots$, part 2 takes the graphs $2, 7, 12, \ldots$ and so on. This way every mtf–graph on the given level is taken exactly once. So, to split up a process, you must run the program several times with the same level and the same mod, but each time with a different class. You see that up to the level every process does the same, but that does not matter since the execution time of the program grows exponentially with the number of vertices and so the common job of every process is rather small.

3 Compiling and running the program

To compile the program, the following files are needed:

- the source code `mtf.c`
- the header file `nauty.h`
- the files `nauty.c` and `nautil.c`[2]

Copy these files into your current directory and type

```
cc -o mtf nauty.c nautil.c mtf.c
```

where "`cc`" is the name of the C–compiler. You get the executable file `mtf`. You run the program by typing "`mtf [options]`" where `options` is a number of options. All possible options are described in the next section.

4 Options

In the following description v, x and x_i stand for numbers and s and s_i stand for strings.

n v This option is not optional, but necessary. v is the maximum number of vertices up to which mtf–graphs are produced. v must be at least 4. The upper limit depends on the value of the internal constant *MAXN2* which is normally 25, but can be increased to 32 without any problems. If you use the option **ramsey**, the upper limit is 24 instead of 25 (or generally *MAXN2*-1 instead of *MAXN2*).

outputlevel x This option is used to determine the minimum number of vertices an mtf–graph must have to be written into an output file. The default value is the value v which is given after option "**n**". If x is bigger than v or smaller than 4, no output files are created. For each vertex number an own output file is created. So by default, exactly one output file is created.

no_recover If you use this option, then the program starts the process right from the beginning even if an appropriate dump file exists. This dump file will not be read.

no_save If you use this option, then the program does not create dump files. You should not use this option because it does not save much time.

[2]The files `nauty.h`, `nauty.c` and `nautil.c` are written by Brendan McKay, Australian National University.

brandt By default, the graph output is written using the code "*multi_code2_s_old*". If you use the option "*brandt*", the output is written using the code "*brandt_old*".

mod x If you use this option, the process is split up into x processes numbered $0, \ldots, x-1$. Use the options "*mod*" and "*class*" simultaneously. See section "Concepts".

class x Use this option to determine the part of the split process which the program must execute. The parts are numbered beginning with 0. A default value is not defined. If you use a value which is equal or higher than the number of parts, then the program will not continue after it has reached the level. See section "Concepts".

level x Use this option to determine the level on which the process must be split. The default value is $\frac{2}{3}$ of the maximum vertex number up to which mtf–graphs are produced. If you use the option "*level*", then you must also use the option "*mod*".

file s If you use this option, the program reads a file where s is the name of the file. This file must contain mtf–graphs in "multi_code2_s_old". The program generates the successors of these graphs instead of beginning the construction with the $K_3 - e$. Of course, the result of this generation process depends on the input file.

local_density x_1 x_2 If you want the local density to be computed, use this option. See section "Introduction" for the definition of local density. x_1 is the number of vertices of every considered subgraph, x_2 is the number of edges these subgraphs must have. If you use this option, only those mtf–graphs which have the local density x_1 x_2 are generated. The number of mtf–graphs which have a certain local density in general first increases and then decreases with growing number of vertices.

ramsey_kn x Use this option to compute the Ramsey number $R(K_3, K_x)$. The Ramsey number is not printed out explicitly. The program converts the option into the equivalent option "local_density x 1". So what you get for each vertex number is the number of mtf–graphs whose complements do not contain the K_x. The smallest vertex number where the number of these mtf–graphs is zero is the Ramsey number $R(K_3, K_x)$. If no number of generated mtf–graphs is zero, then the Ramsey number is bigger than v (where v is the maximum number of vertices in a generated mtf–graph). So you should choose v as high as possible. There will be no loss of time because mtf–graphs are not generated if they are not needed for the computation of the Ramsey number.

ramsey_kn-e x This option works like the option "ramsey_kn", but it computes the Ramsey number $R(K_3, K_x - e)$ instead of $R(K_3, K_x)$. The program converts the option into the equivalent option "local_density x 2".

ramsey s x_1 x_2 Use this option to compute the Ramsey numbers of the test graphs which are in the file named s. The test graphs must be stored with "multi_code2_s_old". You need not provide the parameters x_1 and x_2. If you don't, all test graphs are read. If you only provide the parameter x_1, the first $x_1 - 1$ test graphs of the file s are dropped. If you provide both parameters, the program only reads the test graphs x_1 to x_2 where the first test graph in the file has number 1. The program prints out the explicit Ramsey numbers. The number of mtf–graphs which is also printed out says how many mtf–graphs had to be generated to obtain the Ramsey numbers. This time the output file(s) do(es) not get the mtf–graphs but the test graphs. For each Ramsey number an own file is created. By using the option "outputlevel" you can determine which files must be created at all. By default, only one output file is created: It contains all test graphs whose Ramsey number is **higher** than v. For these graphs the exact Ramsey number is not computed since no mtf–graphs with more than v vertices are generated.

write_ramseygraph You can use this option only simultaneously with the option "ramsey". If you use it, the program will write mtf–graphs instead of test graphs to the output file. The use of the option "outputlevel" will be ignored. Only one output file is opened. For every test graph which Ramsey number is higher than v an example mtf–graph is written which complement does not contain the test graph. An mtf–graph is not written several times if the considered test graph is a subgraph of other test graphs because then it is clear that the mtf–graph's complement does not contain the other test graphs.

If no test graph has a Ramsey number higher than v, then the number "1" is written to the output file as the only output.

write_ramseygraph_all x You can use this option only simultaneously with the option "ramsey". If you use this option, then for every test graph an own output file is opened. For every test graph, all Ramsey graphs with at least x and at most v vertices are written into its related file. If x is missing, then all Ramsey graphs with exactly v vertices are written. If $x > v$, then x is corrected to value v. If you use this option together with then option "stdout", then only one test graph is permitted.

stdout If you use this option, the output graphs which are written by default (see options "outputlevel" and "ramsey") are written on standardout. You can use this option to pipe the graph codes. You can open more output files by using the option "outputlevel", but they are not touched by the option "stdout".

reverse The use of this option reverts the order in which the graphs are generated. Normally mtf–graphs which are "almost regular" are generated last. These are the graphs which need most memory. So if you use the option, then those mtf–graphs which require much memory are generated first. For some Ramsey computations those graphs might be more important than the others. Then you can save time by using this option.

extinfo If you use this option, additional internal information is written into the logfile. This information is useful only for users who know the algorithm very well.

grpsize If you use this option, then the groupsize (the size of the automorphism group) of every generated mtf–graph is counted. A complete statistics about how many mtf–graphs have which groupsize is written into the logfile. A shortened statistics is written on the screen.

5 Example program calls

1. `mtf n 14`

   This call generates all mtf–graphs with up to 14 vertices. For each vertex number the program prints out the number of generated mtf–graphs. The mtf–graphs with 14 vertices are stored in a file named "MTF_N14.m2so". Here "m2so" stands for "multi_code2_s_old". Since this is our first program call with these parameters, the message "`Could not find dump file...`" is printed out.

2. `mtf n 14 brandt`

   This is the same as above, but this time the program generates the output file "MTF_N14.br" which contains "brandt_old"–data.

3. `mtf n 14 outputlevel 4`

   The program prints out the message: `Opened dump file...: Process has already finished!` This is the fact because the dump file created by the first program call is

used. Apart from the output level, we wanted to do exactly the same as in the first call, and the program recognized that it can use the dump file to continue the former process. Then it read the dump file, which contained the information that the process is already finished. So we change the call:

4. `mtf n 14 outputlevel 4 no_recover`

   This time the dump file is ignored. This is the same as the first example, but this time 11 output files are created. Their names are "MTF_N4.m2so" to "MTF_N14.m2so". The first one contains all mtf–graphs with 4 vertices, the last one contains all mtf–graphs with 14 vertices.

5. `mtf n 14 outputlevel 4 no_recover stdout`

   This is the same as before, but this time only 10 output files are created, namely "MTF_N4.m2so" to "MTF_N13.m2so". The mtf–graphs with 14 vertices are written on standardout.

6. `mtf n 17 file MTF_N14.m2so`

   Now we generate all mtf–graphs with up to 17 vertices. Since we have already generated all mtf–graphs up to 14 vertices, we do not need to do that again. Instead of that, the program reads the file "MTF_N14.m2so" which contains these graphs and generates their successors. Of course, in this example the option "file" is not so powerful since it takes only 5 seconds to generate the mtf–graphs with 14 vertices, but 5 minutes to generate those with 17 vertices.

   Have a look at the statistics the program prints out: Below 14 vertices, no mtf–graphs have been generated. This is because the generation begins with graphs which have 14 vertices.

7. `mtf n 17 local_density 10 10`

   Now we generate all mtf–graphs with up to 17 vertices which have local density 10 10. Those with 17 vertices are written into the file "MTF_N17-_LOCDEN_10_10.m2so".

8. `mtf n 25 local_density 10 10 file MTF_N17_LOCDEN_10_10.m2so`

   In the last call it took us some time to compute the results, and we saw that on vertex level 17 there were only a few graphs left which had local density 10 10. So we want to know which is the smallest vertex number where no mtf–graph has local density 10 10. We read in the file which was created in the last example because only the successors of these graphs are interesting: If an mtf–graph has no local density 10 10, then its successors have neither. Here the option "file" is really powerful, since we obtain the final result in a few seconds instead of doing the whole computation again with a higher vertex number.

   We see that 18 is the smallest vertex number where no mtf–graph has local density 10 10. Of course, no mtf–graph with less than 17 vertices has been generated.

9. `mtf n 25 ramsey_kn 6 outputlevel 26`

   We compute the ramsey number $R(K_3, K_6)$. This takes about 5 seconds. We see that 18 is the smallest vertex number where no mtf–graph is left. So $R(K_3, K_6) = 18$. Let's assume that we are not interested in the mtf– graphs whose complements do not contain the K_6. So we use the option "outputlevel" to prevent that an output file is opened.

10. `mtf n 24 ramsey file-which-contains-the-K6`

    Let's assume that the file named "file-which-contains-the-K6" contains the K_6 in "multi_-code2_s_old". Then this program call does the same as the last one, but there are three differences:

- It takes more time to compute the result.
- If output is produced, then the K_6 is written and not the mtf–graphs (in this example no output is produced, since the Ramsey number of the K_6 is not 24).
- The statistics for the mtf–graphs differs: Here the number of mtf–graphs which were actually generated is printed out. In the last example the number of mtf–graphs which were generated **and good** were printed out (where "good" means that their complements did not contain the K_6).

11. `mtf n 15 ramsey 6er.m2so 3 112 outputlevel 10`

    Let's assume that the file named "6er.m2so" contains all connected graphs with 6 vertices. There are 112 of these graphs. The program takes the graphs with numbers 3 to 112 and computes their Ramsey numbers. In this example, we could as well forget the parameter "112" since this is the last test graph in the file. The program opens 7 output files named "MTF_N10_RAMSEY_6er.m2so_F3_L112.m2so" to "MTF_N15_RAMSEY_6er.m2so_F3_L112.m2so" and "MTF_N+16_RAMSEY_6er.m2so_F3_L112.m2so". The first output file contains the test graphs with Ramsey number 10 (in this example there are no test graphs with this Ramsey number, so the file remains empty), the second file contains the test graphs with Ramsey number 11 and so on. The last file contains the test graphs with Ramsey number 16 or higher. This cannot be determined exactly because the program only generates mtf–graphs with up to 15 vertices. So we make another attempt to determine these Ramsey numbers:

12. `mtf n 24 ramsey MTF_N+16_RAMSEY_6er.m2so_F3_L112.m2so`

    This time the limit for the vertex number is high enough. We could as well use the input file "6er.m2so" again, but then we would compute results that we already know.

13. `mtf n 18 mod 5 class 0 level 15 outputlevel 19`

    It takes quite much time to generate all mtf–graphs with 18 vertices. Since they are too many to save them we prevent the creation of an output file by using the option "outputlevel". We split the process into five smaller processes using the above call and four other calls with the options "class 1", "class 2", "class 3" and "class 4" instead of "class 0". Up to vertex level 15, every process does the same. If we have a look at the statistics, we see that the number of generated mtf–graphs is the same in each process. From level 16 to level 18, we must add the single numbers of generated mtf–graphs to obtain the absolute number of mtf–graphs.

    Normally, one of the classes is much bigger than the other ones. To equalize the classes, you should choose the level as high as possible.

6 Output files

In this section the naming of the output files is discussed. The naming of the files is important in two cases:

- When a process is split into many small processes, the output files of these processes must have different names.
- When a process is recovered, the program must find the old output files to append the new data and it must find the dump file. The program must not mix output files or dump files of different program calls.

For these reasons the names of almost every option are considered in the file name to distinguish between different program calls. The only exceptions are the following:

outputlevel Since this option has influence on the creation of output files at all, it does not need to be considered in the name.

no_save

no_recover These two options have no influence on what is generated.

extinfo This option has only influence on the contents of the logfile.

The logfile always has the suffix "`.log`". The dump file always has the suffix "`.dump`". The output files have either the suffix "`.m2so`" or "`.br`". This depends on the code. The code is also considered in the names of the dump file and the logfile. The string "_BR" or "_M2SO" is used. Other strings which are used are "_LOCDEN_x_y", "_RAMSEY_s_Fx_Ly" (where "F" stands for "first" and "L" for "last"), "_Nx", "_RAMSEY_Kx", "_RAMSEY_Kx-e", "_IFILE_s" (where s is the name of the input file), "_WRG" (for "write_ramseygraph"), "_WRGAx" (for "write_ramseygraph_all" where x is the optional parameter), "_Tx" (if the file contains Ramsey graphs related to test graph number x), "_MODx.y_Lz" (where x is the class, y is the mod and z is the level of the split), "_REV" and "_GS". Every file name begins with "MTF".

7 Input files

Input files always must contain "multi_code2_s_old"–data. The result of the computation depends on the input file. For example, if you want to use an input file to generate all mtf–Graphs with 17 vertices, you must provide an input file which leads to a complete and disjunct result. "Complete" means that you will obtain every mtf–graph with 17 vertices. "Disjunct" means that no mtf–graph with 17 vertices will be generated twice. An input file which fulfills these conditions is the file "MTF_N14.m2so" (see section "Example calls"). Since you start with every mtf–graph of order 14, you obtain every mtf–graph of order 17 because every mtf–graph with 17 vertices is a successor of an mtf–graph with 14 vertices. Since the graphs in the file are distinct, the result is disjunct because the successors of two distinct mtf–graphs of the same order are also distinct.

Normally all mtf–graphs in an input file have the same number of vertices. But you can as well provide an input file with mtf–graphs which have several different orders, but the input file is nevertheless complete and disjunct. To be disjunct, no mtf–graph in the file is allowed to be the successor of another. Furthermore, no two graphs in the file must be isomorphic. If these two conditions are fulfilled, you only have to make sure that the input file is complete.

Sometimes it is even useful to use an input file which is not complete. For example, if you split up the file "MTF_N14.m2so" into several smaller files, you can avoid the options "mod" and "class". You start several processes, and every process gets a different part of the input file. However, splitting up an input file is not supported by *MTF*. You can split up a file e. g. by using the program *gconv*.

8 Recovering

Unless you use the option "no_recover", the program considers the options you have chosen, builds the appropriate dump file name and looks for a file with that name. If it does not find a file with that name or if you use the option "no_recover", the program starts the process right from the beginning. It opens a new logfile and new output files. If the program finds the old dump file, it reads the contents. There are two possibilities:

- The dump file contains the information that the process is not yet over. Then the program reads the whole dump file which contains the information about where to continue. The

old logfile and the old output files are opened, but they are not overwritten. Additional data will be appended to the old files. Besides, the output level is also stored in the dump file. So if you recover a process and provide a new output level, this new output level will be ignored. Otherwise you would get output files which are not complete because they were only opened before or after the recovering.

- The dump file contains the information that the process is already over. This kind of dump file is always written at the end of a process (unless you use the option "no_save") to protect the created output files from additional data. If you try to recover such a process, the program prints a warning and stops immediately.

If you recover a process, be sure that all the old output files and the old log file are still in the same place as they were before, because they will be updated!

The dump file is endian–independent. So you can theoretically stop a process on one computer, then copy the data on another computer and recover the process there. This is useful if you have split a process and the slowest computer has got the hardest part of the process. However, you must copy all the old output files on the new computer or concatenate them later with the new files.

If you interrupt a process and recover it, then the output files become a little bit larger than if the process was executed in one go. This is because the program does not remember the last output before the shutdown and so it cannot compress the first graph after the recovering. However, the additional amount is only some bytes.

Everytime the program writes a graph into an output file, it must also write the dump file. If it does not, the program will generate the same graph again after the recovering. So it must write the new dump file which contains the information that the graph has already been generated. To decrease the number of times that the dump file is written, the program collects some graphs before it writes them all at once. If you are very unlucky, then the process is interrupted while writing the graphs to the output file(s). Then you will get the same graphs again after the recovering because the dump file is written after the graphs (so the process has been interrupted before the dump file could be written). Maybe then the output files are even unreadable. But you have to be really unlucky if that should happen because in the worst case the output procedure takes only 3% of the whole process time. In many cases there is no output at all.

9 Using options simultaneously

You cannot use the following options simultaneously:

- "ramsey_kn" and "ramsey_kn-e"
- "ramsey_kn" and "ramsey"
- "ramsey_kn-e" and "ramsey"
- "ramsey_kn-e" and "local_density"
- "ramsey_kn" and "local_density"
- "write_ramseygraph" and "write_ramseygraph_all"

You can use the options "ramsey" and "local_density" simultaneously, if there is a need for that. Then only those mtf–graphs which have the local density are tested whether their complements contain the test graphs.

You can use the options "level x" and "file" simultaneously, even if the input file contains as well graphs with more as with less than x vertices. Let's assume that you split the process

into m parts. For each input graph the program distinguishes whether the graph has more than x vertices or not: If it has more than x vertices, then the program takes only every m–th of those graphs. If it has at most x vertices, then the program generates its successors, and as soon as it generates a graph with more than x vertices, it takes only every m–th of those. These two countings are independent of each other. If an input graph has more vertices than the desired maximum number of vertices, then the graph is dropped.

If you use the options "mod" and "ramsey" simultaneously, then the statistics is useless. The Ramsey numbers are mostly wrong because in one process not all mtf–graphs have been tested whether their complements contain the test graphs. So the Ramsey numbers are normally too low. To split up a Ramsey process, it is better to split the input file so that each process gets only a few test graphs.

10 Internal constants

The following internal constants may be changed by the user. They can be found at the beginning of the source code (after the introducing comment).

MAXN2 This is the maximum number of vertices up to which mtf–graphs can be generated. Default is 25, the upper limit is 32. The higher the value, the more memory is needed to store the mtf–graphs.

TESTGRAPHENLISTENGROESSE This is the maximum number of test graphs to be read when using the option "ramsey". Default is 65535. This number cannot be increased because all count variables related to test graphs are of type "unsigned short". However, if not every graph is read, then the file which includes the test graphs can contain more than 65535 graphs.

MAXFULLTEST This is the maximum number of test graphs to be read when using the option "write_ramseygraph_all". Default is 100. However, if not every graph is read, then the file which includes the test graphs can contain more than 100 graphs.

MEMBLOCKSIZE The program has its own memory management. For that it uses memory blocks. The size of each block is "MEMBLOCKSIZE" bytes. The smaller the value, the more blocks will be allocated and so the more memory and the more time will be needed. You should not change the value. Default ist 65536.

filenamenlaenge If a file name occurs which is longer than 255 characters, then you must increase this value. But this is only possible if the file name contains names of other files together with long paths. It is then useful to copy these other files into the current directory so that the paths vanish.

Sicherungsintervall Unless the option "no_save" is used, a dump file is created after at most this amount of seconds. Default is 900. The smaller the value, the more time is needed by the program.

11 Hints

Ramsey If you use the option "ramsey" and you know that the Ramsey number of each test graph is at most x, use the option "n" with parameter $x - 1$. This may save a lot of time during the process, since for each test graph the computation stops as soon as the program recognizes that the Ramsey number must be higher than $x - 1$. If you use the option "n" with parameter x, then the program tries to find out if the Ramsey number is maybe higher than x.

12 Results

The following results were computed on a DEC alpha (21064A, 233 MHz).

#vertices	#mtf–graphs	computation time
4	2	0.0s
5	3	0.0s
6	4	0.0s
7	6	0.0s
8	10	0.0s
9	16	0.0s
10	31	0.0s
11	61	0.1s
12	147	0.1s
13	392	0.4s
14	1274	1.2s
15	5036	4.2s
16	25617	17.5s
17	164796	97.2s