Technical Teport

Matjaž Depolli

first version 2013/11/11
additional results 2014/1/8

1 Introduction

Source code used as the method of the article *Exact Parallel Maximum Clique Algorithm for General and Protein Graph*[1] from the *Journal of Chemical Information and Modeling* (JCIM) was made available on-line[1]. A bug in the code was discovered by *Ciaran McCreesh* and reported to me on 2013/10/13. The bug was in the code for initial vertex sorting, which appeared different than the pseudo-code of the article. This was the code for the algorithm, developed by Tomita et al. and published in their article[2]. The algorithm was supposed to be used without modifications, and pseudo-code was written appropriately. Source code, however, performed an additional iteration within the sorting, which did not altered the order of several vertices after the sorting operation, and prevented the execution of the initial-clique checking after the sorting. This procedure checks for a clique within the sorting process, and stores the clique as the initial guess for the maximum clique, thus making the main search for the maximum clique a little bit faster. The source code was lacking this procedure, as the the condition for it would always fail. The correctness of the results was however not effected by the bug, since the initial order of vertices only influences search speed.

2 Solution

The bug was eliminated and the initial-clique checking procedure was added to the source. This procedure is not very compute intensive; it includes one comparison, and optional assignment of one vector of vertices (a clique). On itself, it should not influence the run times of the program in any significant way. It could, in theory, reduce the search time for maximum clique significantly when providing a good initial guess. How often a good initial guess is provided, however, is difficult if not impossible to determine analytically. Also the bug-free initial ordering should provide a boos in speed, but again, the speedup cannot be analytically estimated. Therefore a series of tests was performed to evaluate the performance difference caused by the bug. The tests were performed on a machine from the local computer cluster (CPU: Intel Xeon E5520 @ 2.27GHz).

[1] `http://www-e6.ijs.si/ matjaz/maxclique/`

3 Experiments and Results

First the source code was modified only to output whether the algorithm finds an initial guess for maximum clique or not and used to determine on which graphs the algorithm was more influenced by the bug. Then the version of algorithm with the bug and the version without were run on all the input graphs from the experiments published in the before-mentioned article. Both algorithms were executed in single-threaded mode, to isolate the effects of the bug from the effects of the parallelization. The results are summarized in Table 3. Since the computer used for the new experiments differs from the computer used in the experiments for the article, absolute times are not used, instead the speedup measured as the ratio of the number of iterations performed and the speedup of the execution time are shown, both for the new version, relative to the old version with the bug. Only one run was performed for both algorithm versions, therefore there is some variation in run times even in cases when the number of iterations performed is the same.

The bug in the source code increased the mean execution time of the algorithm on DIMACS benchmark graphs by roughly 6%. For most of the graphs, the difference is negligible and within the normal variation of run times. A few graphs stands out though, namely `gen200 p0.9 44, p hat300-2, p hat500-2, and san400 0.7 1`, where the performance is significantly improved - by more than 20%. Weather the clique is found or not in the phase of ordering vertices (preprocessing) does not seem important for DIMACS graphs. The largest speedup occurs on the Graph "san400 0.7 1", where the clique is not found in the preprocessing and the only difference is the order of vertices.

On the protein product graphs, however, the difference is far from negligible and the bug fix produces significant improvements, as shown in Table 3. The clique is found in preprocessing in 90 % cases on these graphs, and this seems to speedup the bug-free algorithm enormously.

Parallel speedups on Protein product graphs are greatly reduced, since the sequential algorithm times decreased by such large factors. The execution times are shown in Figure 3, while the renewed speedup measurements are shown in Figure 3. These measurements are done against the sequential algorithm, therefore also the speedup of the parallel algorithm on a single thread is shown, which experiences slowdown purely due to parallelization overhead. The computer used for measurements is not the same as in the JCIM article, and comprises only 4 cores with hyperthreading technology. Therefore, although plotted on the same graph, the experiments on 5 or more cores are not directly comparable, since they are running partly on virtual cores, which achieve up to 30 % of the performance of sibling physical cores.

4 Summary and Commentary

While the bug is mostly harmless in 90 % of DIMACS graphs it slows down the algorithm significantly in the other 10 % of the cases; that is by 20-60 %. The effects are very similar in proportion both on the number of performed iterations and on the total execution time. On protein product graphs, on the other hand, the effects of the algorithm are negligible in only 1 case, while they are very large in the other 9 cases. For 8 of those cases, the number of recursions the

graph name	initial clique	speedup (iterations)	speedup (time)
C125.9	x	1.052	1.024
C250.9	x	0.999	1.009
MANN a27		1.000	1.019
MANN a45		1.000	1.007
brock200 1	x	1.000	1.025
brock200 2	x	1.011	1.024
brock200 3	x	1.005	1.090
brock200 4	x	1.002	1.036
brock400 1		1.000	0.991
brock400 2	x	1.034	1.021
brock400 3		1.001	0.999
brock400 4		1.000	0.992
gen200 p0.9 44	x	1.216	1.201
gen200 p0.9 55	x	1.018	1.013
hamming10-2		1.000	0.974
hamming8-2		1.000	0.985
hamming8-4		1.000	1.009
johnson16-2-4		1.000	0.993
keller4		1.000	0.967
p hat1000-1		1.000	0.964
p hat1000-2		1.023	0.978
p hat1500-1		1.000	0.936
p hat300-2	x	1.713	1.262
p hat300-3	x	1.008	1.027
p hat500-1	x	0.998	1.008
p hat500-2	x	1.487	1.455
p hat500-3	x	1.029	1.026
p hat700-1		1.000	0.995
p hat700-2	x	1.038	1.021
p hat700-3	x	1.052	1.037
san1000		1.000	0.998
san200 0.9 1		1.038	1.045
san200 0.9 2		0.990	0.961
san200 0.9 3		1.026	1.032
san400 0.5 1		1.001	1.012
san400 0.7 1		1.690	1.690
san400 0.7 2		1.000	0.984
san400 0.7 3		1.000	0.992
san400 0.9 1		1.000	0.993
sanr200 0.7	x	1.083	1.068
sanr200 0.9	x	1.025	1.032
sanr400 0.5	x	1.001	1.021
sanr400 0.7	x	1.000	1.009

graph name	initial clique	speedup (iterations)	speedup (time)
1KZKA 3KT2A	x	142.160	10.967
1allA 3dbjC	x	82.530	18.882
1f82A 1zb7A	x	18.296	8.649
2FDVC 1PO5A	x	116.262	23.385
2UV8I 2J6IA		0.997	1.027
2W00B 3H1TA	x	21.652	12.982
2W4JA 2A2AD	x	17.060	8.265
3HRZA 2HR0A	x	304.418	186.698
3P0KA 3GWLB	x	2.382	1.577
3ZY0D 3ZY1A	x	20.400	2.206

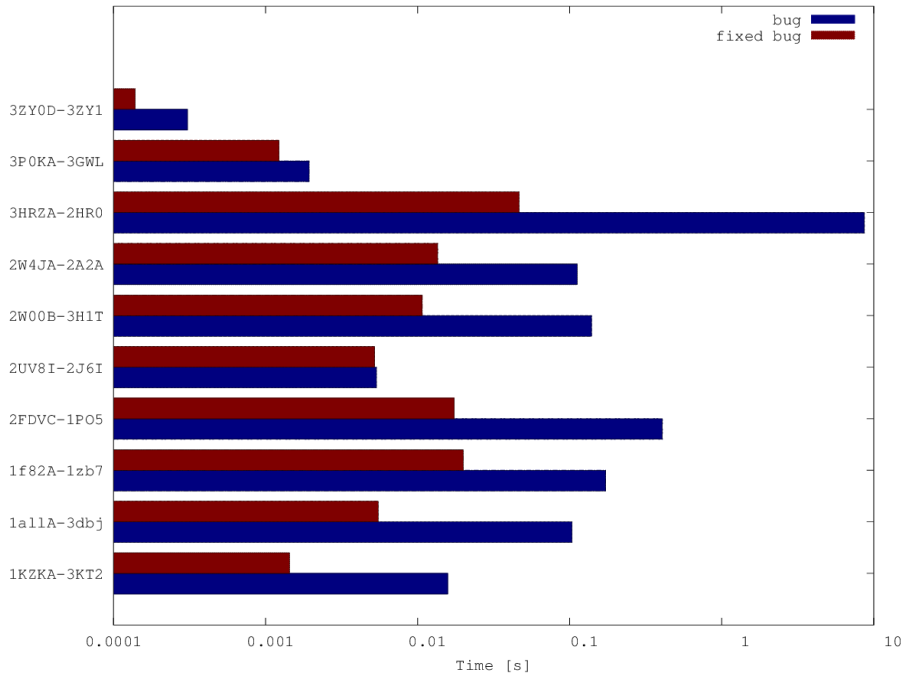

Figure 1: Execution times of the sequential and algorithms before and after the bug fix. Protein product graphs are used, that were also experimented with in the JCIM article

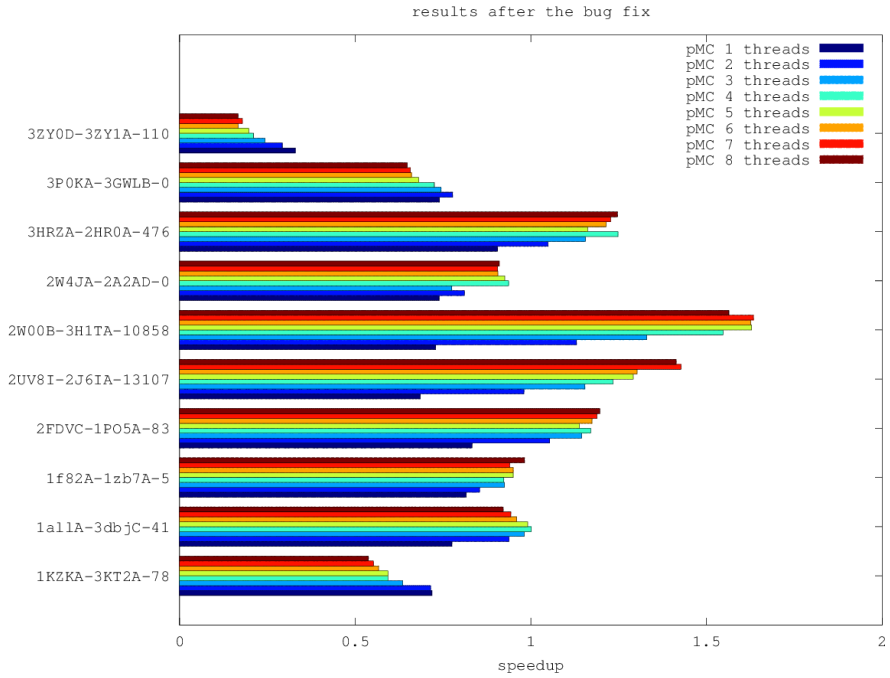

Figure 2: Parallel speedups of the parallel algorithm on protein product graphs, that were also experimented with in the JCIM article

algorithm does is reduced by a factor of 20 or more, in 3 cases even by a factor of 100 or more. These factors are smaller when execution times are considered, but they are still greater than 10 in half the cases.

Whether a clique is found within the initial sorting does not seem to play a role in bug induced slowdown in the case of DIMACS. There are cases where the bug fix increases and cases where it decreases the number of iterations of the algorithm. This holds for cases when initial sorting finds and does not find a clique. The results therefore suggest that both, the improved initial sorting and the initial clique guess improve the search speed. This hypothesis seems reasonable, since it is known that there is no known sorting that would work best in all cases, and that the initial clique guess is very likely not maximal. In contrast, the results on the protein product graphs suggest that when the initial clique guess is obtained, the number of iterations is greatly reduced. There is only one case, where no initial clique guess was made, and that example executes in larger number of iterations after than before the bug fix. It is nevertheless reasonable to believe that both, the initial clique guess and the different order of vertices play an important role in the speedup brought by the bug fix.

These experiments further support the notion that initial sorting is one of the most important factors of the maximum clique search, and that the initial sorting suggested by Tomita et al.[2] works especially great on the Protein product graphs. The number of graphs tested is too small to support this claim with great confidence and caution should be taken, but with more tests, this claim is very likely to be confirmed. Please note that the reviewed results introduce great

changes in the Protein Product Graphs subsection of the section Results of the original article. With the new results, the MaxCliqueSeq algorithm steals the lead in three additional cases (2w4jA-2a2aD, 1kzkA-3kt2A, and 3hrzA-2hr0A), making it the fastest in 9 of 10 cases. In addition, the speedups shown here make it by far the fastest of the tested algorithms in several cases.

On the other hand, the new results bring into question the usefulness of the parallelization for the use on protein product graphs. These graphs are now solvable sequentially in such short times, that the parallelization cannot reduce the execution time much further. The bar is therefore raised for the complexity of the protein product graphs on which the maximum clique algorithm will benefit from the parallelism.

References

[1] Matjaž Depolli, Janez Konc, Kati Rozman, Roman Trobec, and Dušanka Janežič. Exact parallel maximum clique algorithm for general and protein graphs. *Journal of chemical Information and Modeling*, 53(9):2217–2228, 2013.

[2] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, 37(1):95–111, 2007.