Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp807-811)

Applying AI and Incomplete Solution Principles to Solve NP-hard Problems in the Real-Time Systems

DENISS KUMLANDER
Department of Informatics
Tallinn University of Technology
Raja St.15, 12617 Tallinn
ESTONIA

Abstract: - In this paper a question of using artificial intelligence principles and an incomplete solution approach were explored for solving NP hard problems in the real-time systems using the maximum clique finding problem as an example. The incomplete solution is used to analyze different best known algorithms in the real-time environment, while artificial intelligence principles were implemented in a form of a meta-algorithm containing other problem specific algorithms. Experiments conducted in this paper have demonstrated that the meta-algorithm in a randomly generated graphs environment required up to 3 times less time to find a solution in a certain range of graphs than the best known general type algorithm, and was never slower in other ranges.

Key-Words: - NP-hard problem, maximum clique, artificial intelligence, incomplete solution

1 Introduction

There are a lot of problems that are not so easy to solve as it looks like at first. A *polynomial time algorithm* is an algorithm whose time complexity function is $O(p(n))$, where $p(n)$ is a polynomial function. Any algorithm whose time complexity function cannot be so bounded is an *exponential time algorithm*. The first most important researches in the algorithms complexity area were done by Turing in the 1940s. Turing has demonstrated that some problems are "undecidable", i.e. those problems can be solved algorithmically. Moreover his works have greatly affected complexity theory for "decidable" problems since his abstract computer model, so called Turing machine, was used for researches and definitions in this area. NP-class problems are defined as problems that can be solved on a *none-deterministic* Turing machine in a *polynomial* time [8]. There is a P class as well, which contains problems that can be solved in a polynomial time on the deterministic Turing machine, which is also called just the Turing machine. Sometimes NP-class is defined as a class of problems that cannot be solved on the Turing machine, although it is not quite correct, since P $\subseteq$ NP. So NP-P problems cannot be solved in a polynomial time on the deterministic Turing machine. NP-completeness theory foundations were laid in a Cook paper presented in 1971 [6]. He has shown that any NP problem can be converted into the satisfiability problem in a polynomial time. He also has demonstrated that there are some other problems, which have the same complexity as the satisfiability problem. Those problems are "hardest" problems or are an essence of NP-class. Later a lot of problems have been shown to be as "hard" as the satisfiability problem [10] and those problems were called NP-complete problems. The formal definition for NP-complete is the following: a problem is *NP-complete* if the problem belongs to NP-class and *any* other problem of NP-class can be polynomially transformed into this problem. Problems, which are at least the same hard, i.e there are NP-complete tasks that are Turing reducable, are called *NP-hard*.

There are a lot of algorithms targeted to solve different NP-hard problems, mainly concentrating on different aspects of a problem to be solved. In this paper we are going to highlight another important property - an environment where algorithms are applied, which is also very important to consider. A lot of applications are serving solutions for for real-time systems, which can be defined as a system where NP-hard problems should be solved during some restricted time. It means that if a sub-algorithm serving soltuion is not able to produce it during the available time, then it is stopped and the current best solution is used. Therefore there is a potential conflict between real-time systems and NP hard problems requirements (mostly time requirement) that will be reviewed in this paper using the maximum clique problem as one of the NP-hard problems.

Let $G=(V,E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Two vertices are called to be *adjacent* if they are connected by an edge. A *clique* is a complete subgraph of G, i.e. one whose vertices are pairwise adjacent. The *maximum clique problem* is a problem of finding the maximum complete subgraph of G, i.e. a set of vertices from G that are pairwise adjacent. This problem is NP-hard on general graphs [8], i.e. no polynomial time algorithms are expected to be found. This problem is choosen as an example of NP hard problems since there is a great interest in developing a fast exact algorithm for instances with a reasonable number of vertices since it can be used in several important practical applications. Examples are efficient register allocation [5], on-line bin-stretching [1], scheduling of parallel jobs [3] and a lot of others.

So far, researches where not really interested in considering the real-time environment although a lot of applications are hosted in such environments, like dispatching, airlines/trains scheduling and controlling applications [14]. The real-time systems where reviewed in one of our earlier articles [11] where different algorithms were reviewed from the real-time systems environment point of view. Here this discussion is continued by applying artificial intelligence principles to an incomplete solution finding, which makes it much more efficient. Notice that an idea of adaptive maximum clique algorithms is also sometimes discussed in conference halls, but unfortunately these discussions were not finished so far in a form of an article except some our previous works [13] that are about to be extended by this paper.

The paper is organised as follows. The paper starts in the section 2 introducing an "incomplete" solution approach that enables analysing different algorithms, which are known to be best for the maximum clique finding [4, 12, 15], from the real-time systems point of view. The next section describes ways of applying artificial intelligence principles to the described in the introduction topic of the paper. Experiments are conducted in the section 4. The last section concludes this paper.

2 Incomplete Solution

An "incomplete solution" term describing the real-time system case was introduced in one of our previous researches [11]. Normally a maximum clique algorithm finds a solution somewhere in the middle of its work. Thereafter the algorithm tries to prove that it is the maximum one by looking through all remaining vertices. Some algorithms are able to find the best solution during the first iterations, although sometimes it depends on the vertices sorting. We will call an "incomplete solution" such solution, which is already best/maximum for a particular graph, although it is not yet proved. As you see, it is rather a state of the maximum clique finding process than a type of solution since at the end of ends an incomplete solution will become a complete one. This state occurs as soon as the maximum clique is found and last up to the algorithm's work end [11].

Applying exact solutions in the real-time systems has a clear advantage over using heuristic algorithms. The real-time systems don't guarantee, but potentially could provide enough time to find the maximum clique therefore there is a certain probability that an exact solution will be able to end its work and provide the proved solution.

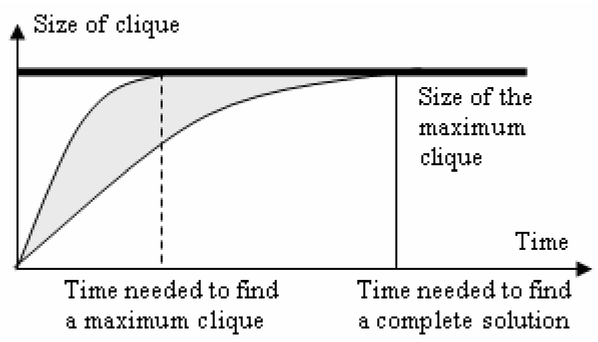
Size of clique
Size of the maximum clique
Time
Time needed to find a maximum clique
Time needed to find a complete solution

Fig 1. Maximum clique finding time scheme

Different algorithms are behaving differently in finding the incomplete solution, which is denoted on the figure 1 by a grey area moving towards the maximum clique level with different speeds from the time scale point of view. This line for each individual algorithm is greatly depends on the algorithm structure. Therefore it is a type of information that:

- Can be tuned in existing algorithms;
- Should be considered by the artificial intelligence algorithms if any is applied.

Some algorithms that are good for finding the maximum clique in standard systems are slow to find the incomplete solution and therefore could be avoided to apply in the real-time systems [11].

3 Artificial Intelligence

There is a clear need to find a way that could guarantee applying the best algorithm among available as it was shown in the previous subchapter. Some artificial intelligence principles can be used to build an algorithm that will be able to meet this requirement. A meta-algorithm idea was proposed in one of our previous works for the general problem [13]. The meta-algorithm contains all algorithms that can be useful to apply, knowledge data and a decision module, which defines the best algorithm among available basing on the graph properties and knowledge that the meta-algorithm already has.

Results of applying those ideas to the general maximum clique finding case were promising since even a simple meta-algorithm produced performance increase up to 1.9 times in certain ranges of parameters and never was slower. At the same time this effect was not studier fully since starts to appear stronger when more different graphs instances exist. Another advantage of using the meta-algorithm was its adaptivity to particular environments where it has to operate. Initial analyses on the real-time system from the incomplete solutions point of view shows that effect of applying the meta-algorithm idea should be much stronger: the real-time systems are much more critical, the incomplete solution finding process requires much less time, algorithms characteristics varies more in the incomplete solution case.

A central part of the meta-algorithm is its knowledge base and the decision module. The meta-algorithm designing stage has to define list of parameters to be used to make decisions. A density is the first and main parameter that should be included into the list, as most researches differentiate graph cases by this parameter [4, 9, 15]. A number of vertices is not so obvious since different algorithms' work-properties grow proportionally and the number of vertices doesn't affect those proportions of best-known algorithms. A type of a graph is another important parameter to use since there are certain algorithms that are either built especially for certain types or behave much better on certain types [2]. Unfortunately the type identification process is very complex (even the graph decomposition task is hard to solve) therefore other indirect properties are proposed to be used like a source of the graph. Those indirect properties could refer to special types, although it doesn't happen always.

The decision module can operate using two types of knowledge definitions: static and dynamic data. The static knowledge is knowledge obtained during previous researches and remains unchanged during the meta-algorithm work-live. Typically it is information about some special graphs that can be resolved by special algorithms. If dynamic data is missed then pervious researches about different algorithms defines fully decision rules and a decision tree. An expert type algorithm is a good example of such meta-algorithms. The dynamic data is data, which is not loaded prior the meta-algorithm goes into the operational phase or is changing during that. A learning (training) process has to be established to derive this type of knowledge.

One alternative to learn the meta-algorithm can be to train it to use available (inside its) algorithms. It is done before the meta-algorithm goes into a production phase, i.e. before it starts to work as a part of the real-time systems. It is always advisable to use for training examples that are either got from the production environment or is simulated very closely. This type of learning suits first of all to systems that are stable, i.e. graphs instances that have be solved are not changing dramatically during the system's live. In that case the training prior to real use will guarantee the correct process of applying the learned knowledge.

If the system is not stable from the graph instances point of view or new instances could appear, then it is advisable either to re-train the system periodically or apply an "online" training process, i.e. training during the operation. The real-time systems have to solve problems quickly, but sometimes do it rarely and certain resources could be available during this "free" time for other operations. Those resources can be used to run other algorithms with last graph instances to find if other algorithms can perform better than the used one. This could follow to knowledge reformulation. Different points systems, for example, can be used to learn and avoid too large affect of old data in case instances of some sources are slowly migrating to other types.

4 Experiments

The described principles were applied to the incomplete solution finding of the maximum clique problem from the randomly generated graphs. The main purpose of these tests is to obtain efficiency of applying some artificial intelligence principles for finding incomplete solutions of an NP- hard problem. Three algorithms were used as algorithms that are available to the meta-algorithm and to compare with.

The first algorithm to participate in the testing is a very simple and effective algorithm proposed by Carraghan and Pardalos [4]. This algorithm was used for benchmarking as a base algorithm in the Second DIMACS Implementation Challenge [9]. Besides, using of this algorithm as a benchmark is advised in one of the DIMACS annual reports [7]. The algorithm is a branch and bound one and uses number of remaining vertices on each depth to prune the maximum clique search tree. This algorithm doesn't spend valuable time on complex bounds checking and can be characterised as a direct one, from the solution finding point of view.

Remaining two algorithms are best known algorithms for finding the maximum clique at the moment accordingly to articles published during last 3 years. The only except among best known algorithms that is not included in tests is an algorithm, which is proposed by Östergård [16]. The algorithm considers vertices in the backward order, i.e. contains so called backtracking search logic and therefore is not efficient in the incomplete solution [11] finding. Two others algorithms, that are used in the work, are very similar since improve the first algorithm by using a heuristic vertex colouring for the maximum clique search tree pruning formula instead of number of remaining vertices. The second algorithm is proposed by Tomita and Seki [15] and re-colour vertices on each new depth of the search tree. The third one is proposed by Kumlander [12] and re-uses vertex colouring obtained before the maximum clique search is started. The last algorithm does not use its backtracking search logic, as in the original paper.

A test-program was used to test algorithms that were described above. It includes a graph generation subtask and a subtask that runs algorithms to be researched and measures the spent time. The graph density parameter was used to distinguish different tests' graph cases, since the source is the same and cannot be used as an additional parameter. 100 graphs are generated for each density and densities vary from 5% to 95% with a step that equals to 5%. Each generated graph is used as an input for each algorithm to be analysed. Experiments period was divided into the learning phase and the operational one, since the graph generation module was producing the same randomised graphs during both phases, so training during the operational phase would not add much more into the learned knowledge quality. An incomplete solution was incorporated into the test in the following way: each time a graph instance has to be solved, the Tomita and Seki algorithm was used to obtain the solution. Thereafter all algorithms were run one by one with the maximum clique size as an input parameter. As soon an algorithm reaches the maximum clique size level it stops (the incomplete solution is found).

Times for finding a solution are summed up for each density by algorithms (for all 100 instances) and an algorithm with the shortest time is defined as the best one for that density. That algorithm is to be used by the meta-algorithm during the operation phase.

Results obtained during the operational phase are presented as ratios of algorithms summed spent times on finding the maximum clique. Those are shown by densities. This presentation makes those results reproducable on any platform etc., i.e. platforms, computers and programming languages independent. The meta-algorithm is used here as a benchmarking one. Therefore ratios show nothing else than how much slower an algorithm is in compare to the meta-algorithm. The meta-algorithm selects an algorithm to apply simply by following the earlier obtained mapping of best algorithms to density parameter therefore additional time spent on following rules is nearly 0. Therefore an algorithm that was used by the meta-algorithm on each density can be seen in the table 2 also – its ratio equals to 1.00. This makes possible to omit the learned knowledge presentation table in that work since the algorithm that was used to find solutions for certain density graphs can be derived from the results table below by looking for an algorithm having 1.00 on that density. Algorithms columns headers of the table mean:

CP – time needed to find the incomplete solution (max clique) by Carraghan and Pardalos [4] algorithm divided by time needed to find the incomplete solution (max clique) using the meta-algorithm;

TS – time needed to find the incomplete solution (max clique) by Tomita and Seki [15] algorithm divide by time needed to find the incomplete solution (max clique) using the meta-algorithm;

DK – time needed to find the incomplete solution (max clique) by Kumlander's [12] algorithm divided by time needed to find the incomplete solution (max clique) using the meta-algorithm.

Table 1. Ratios of times needed to find the incomplete solution (max clique)

Density	***CP***	***TS***	***DK***
5 %	1.00	5.29	5.36
10 %	1.00	3.52	3.30
15 %	1.00	2.30	2.48
20 %	1.00	1.14	1.17
25 %	1.02	1.00	1.10
30 %	1.30	1.00	1.16
35 %	1.59	1.00	1.50
40 %	2.47	1.00	1.66
45 %	2.65	1.00	1.96
50 %	3.02	1.00	2.44
55 %	4.45	1.00	2.82
60 %	4.76	1.00	3.52
65 %	9.65	1.00	3.42
70 %	8.83	1.00	4.05
75 %	9.81	1.00	3.91
80 %	13.44	1.00	5.26
85 %	38.94	1.00	4.84
90 %	233.46	1.00	4.61
95 %	2542.26	1.00	5.10

For example 3.02 in the column *CP* and in the row with density 50 % means than Carraghan and Pardalos algorithm requires 3.02 times more time than the meta-algorithm to find the maximum clique from 100 graphs having 50% vertices' density.

The results show that in the presented environment the meta-algorithm is never slower since there is no number less than 1.00. The best algorithm to compare it with is the Tomita and Seki [15] algorithm, which was used in most cases and its average time ratio is also the lowest among others. It is clear to see, that applying only Tomita and Seki algorithm on low densities graphs (up to 20%) will be slower than the intelligent meta-algorithm approximately 3 times having no advantages on higher densities.

5 Conclusion

In this paper a question of using artificial intelligence principles and an incomplete solution approach were explored for solving NP hard problems in the real-time systems using the maximum clique finding problem as an example. The incomplete solution approach was used to fix a moment when an algorithm finds a solution to eliminate the proving part of its work. A meta-algorithm was proposed to be used, that contains algorithms to solve the problem, knowledge and a decision module defining, which algorithm to apply. The learning process greatly depends on the operational environment stability. Learning in operation or prior were proposed, using density, source of graphs etc parameters to cluster algorithms. Experiments have demonstrated that the meta-algorithm in the randomly generated graphs environment required up to 3 times less time to find a solution in a certain range of graphs than the best known general type algorithm, and was never slower in other ranges. It demonstrates that applying the artificial intelligence principles in the real-time systems can produce a sufficient performance improvement of solving NP-hard problems in such critical applications. The meta-algorithm adaptivity to environments is an extra advantage of the described approach.

A lot of described algorithms are using different heuristics to find bounds, for example the described algorithms are using the vertex-colouring subtask for that. The future research can be an expansion of artificial intelligence principles to different used heuristic as different graph types could demand using different vertex-colouring approaches. Another topic could be researching algorithms on possibility to add artificial intelligence into the core of those instead of using just outside.

References:

[1] Y. Azar, O. Regev, On-line bin stretching, *Theoretical Computer Science*, Vol. 268, 2001, pp. 17-41

[2] C. Berge, V. Chv'atal, Topics on Perfect Graphs, *Ann. Discrete Math.*, Vol. 21, North-Holland, Amsterdam, 1984

[3] S. Bischof, E.W. Mayr, On-line scheduling of parallel jobs with runtime restrictions, *Theoretical Computer Science*, Vol. 268, 2001, pp. 67-90

[4] R. Carraghan, P.M. Pardalos, An exact algorithm for the maximum clique problem, *Op. Research Letters*, Vol. 9, 1990, pp. 375-382

[5] G.J. Chaitin, M.A. Auslander, A.K. Chandra, J. Cooke, M.E. Hopkins, P. Markstein, Register allocation via coloring, *Computer Languages*, Vol. 6, 1981, pp. 47-57

[6] S.A. Cook, The complexity of theorem proving procedures, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, 1971, pp. 151-158

[7] DIMACS, *Annual Report*, Center for Discrete Mathematics and Theoretical Computer Science, 1999

[8] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, New-York, 2003

[9] D.S. Johnson, M.A. Trick, Cliques, Colouring and Satisfiability: Second DIMACS Implementation Challenge, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, Vol. 26, 1996

[10] R.M. Karp, Reducibility among combinatorial problems, *In complexity of Computer Computations*, Plenum Press, New York, 1972, pp. 85-103

[11] D. Kumlander, Incomplete solution approach for the maximum clique finding in the real-time systems, *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications*, 2006, pp. 75-79

[12] D. Kumlander, A practical algorithm for the maximum clique finding, *Proceedings of the IADIS International Conference Applied Computing*, 2006, pp. 266-272

[13] D. Kumlander, Improving the maximum clique finding applications using artificial intelligence principles, *Proceedings on the 7th WSEAS International Conference on Automation and Information*, 2006, pp. 132-137

[14] D.J. Musliner, J.A. Hendler, A.K. Agrawala, E.H. Durfee, The Challenges of Real-Time AI, *Computer*, Vol 28, No. 1, 1995, 58-66

[15] E. Tomita, T. Seki, An effcient branch-and-bound algorithm for finding a maximum clique, *Discrete Mathematics and Theoretical Computer Science, 4th International Conference, LNCS 2731 Springer*, Vol. 2003, pp. 278-289

[16] P.R.J. Östergård, A fast algorithm for the maximum clique problem, *Discrete Applied Mathematics*, Vol. 120, 2002, pp. 197-207