Tom Waszkowycz
King's College
tw397

Part II Project Proposal

Solving NP-Complete Problems via Satisfiability

23rd October 2014

Introduction and Description of the Work

Many problems in fields such as AI, circuit design and automatic theorem proving are NP-Complete. We want to solve them efficiently, but no polynomial time algorithm has been discovered for any NP-Complete problem, despite the fact that no super-polynomial lower bounds have been found either.

One problem in NP for which many of the instances that occur in practice can be solved relatively efficiently is the Boolean satisfiability problem (SAT). This is the problem of determining if there exists an assignment of the variables of a formula in Boolean logic that satisfies the formula i.e. can evaluate it to true. Over the past decade there have been dramatic advances in heuristic based SAT-solvers, which now perform very well for many practical applications, and many of which are readily available as free or open source software.

The Cook-Levin theorem states that SAT is NP-Complete. This means that any problem in NP can be reduced in polynomial time to an instance of the Boolean satisfiability problem. I plan to use this result to take instances of various NP-Complete problems, reduce them into equivalent formulae of Boolean logic, and then use an efficient SAT solver to compute a solution to the original problem. This can be done with any problem in NP, and hence any NP-Complete problem.

Starting Point

I have knowledge of various systems of logic from attending courses in Discrete Mathematics in Part IA of the Computer Science Tripos and Logic and Proof in Part IB, and knowledge of computational complexity theory from attending the Complexity Theory course in Part IB.

Substance and Structure of the Project

I aim to implement a system that can take an instance of a problem in the complexity class NP, reduce it to an equivalent Boolean satisfiability problem, and then attempt to solve it using a SAT-solver.

This will be done by first constructing a specification language to describe problems in NP, on which work has previously been done. Ronald Fagin proved in 1974 the class of problems NP is exactly the set of all formulae expressible in existential second-order logic ($\exists$SO) (Fagins Theorem). This means all problems that I would wish to input into my system can be expressed as such, and so the front end of the system would consist of a parser which accepts such languages.

As the input problem will be in NP, by the Cook-Levin theorem we can reduce this in polynomial time to an instance of SAT. This is done by substituting values and expanding all existential and universal quantifiers over the (finite) values of the domain given by the supplied instance. This will then produce a Boolean logic formula, which can be considered as an instance of SAT equivalent to the original problem instance. It would be useful to produce this in a standardised format (such as DIMACS-CNF).

This formula will then be fed into the back end of the system, consisting of a 3rd party SAT solver. This will then attempt to solve the formula, and hence the original problem instance that was input, outputting the result (if any). It is worth noting that all current SAT-solvers still have exponential worst case time complexity, and so it is not uncommon for them to fail to find a solution within their given time limit.

The first problem I plan to focus on first is graph three colourability (3COL), as it is relatively easy to specify in $\exists$SO, and instances of the problem can be described simply by a graph consisting of a set of vertexes and a set of the edges between them. To evaluate it I will test it against a number of 3COL instances, which can be found in online benchmark collections. I will evaluate its performance by comparing the size of problems that it is able to solve against the size of problems required in practical applications and against other similar systems.

Once I have this working for 3COL I plan to expand it to work with other NP-Complete problems, and test them similarly. A possible extension, time permitting, would be to incorporate problems that take both an integer parameter as well as a graph (e.g. Clique, Independent Set, Vertex Cover), which are more difficult to specify in $\exists$SO. I would consider SAT solvers which incorporate ways of expressing integer constraints, and extend my specification language to allow for these.

Possible further evaluation methods could involve contrasting performance of my system on different NP problems, or comparing performance with a range of different SAT-solvers. To improve my project I could look into extending the specification language to allow for easier descriptions of more complex NP languages, and any improvements that I could make to my system to improve performance.

Special Resources Required

I will require use of my own laptop for development, one or more SAT-solvers (many are free or open source), and testing benchmarks of instances of NP-Complete problems (also freely available).

Success Criterion

My system should be able to take a description of an NP problem (e.g. 3COL) in my specification language, along with an instance of this problem (i.e. a graph), convert this into an equivalent SAT instance, and feed this into a SAT-solver to determine the solution to the original problem instance.

Timetable and Milestones

1. **27th Oct - 9th Nov** Initial research into similar work done
   Deliverable: More in depth knowledge and notes on existing research, and how this relates to my project

2. **10th Nov 23rd Nov** Identification of appropriate SAT solver/s, testing benchmarks and specification language
   Deliverable: Be able to specify the above with reasons

3. **24th Nov 7th Dec** Planning and initial coding of core project

4. **8th Dec - 21st Dec** Continue coding of core project

5. **22nd Dec - 4th Jan** Complete and test core project
   Deliverable: System tested and working for 3COL instances

6. **5th Jan - 18th Jan** Begin evaluation of system, and extend core project

7. **19th Jan - 1st Feb** Preparation of progress report
   Deliverable: Fully prepared document and presentation

8. **2nd Feb - 15th Feb** Complete evaluation and any further extensions
   Deliverable: System evaluated as described above

9. **16th Feb - 1st Mar** Begin writing dissertation
   Deliverable: Dissertation plan

10. **2nd Mar - 15th Mar** Continue dissertation writing

11. **16th Mar - 29th Mar** Complete draft of dissertation and submit to supervisor
    Deliverable: All parts of dissertation complete

12. **30th Mar - 12th Apr** Revise dissertation based on supervisor feedback

13. **13th Apr - 26th Apr** Any last minute revisions and preparation for submission
    Deliverable: Dissertation ready for submission