Common Subexpression Elimination and SAT Problem Solving

Yuanlin Zhang
with Chris Gutierrez and Jeriah Jn-Charles
Texas Tech University
Outline

• Introduction of common subexpression elimination (CSE) and SAT solving
• Common subexpression elimination in SAT
• CSE algorithms
• Empirical work on impact of CSE on SAT solving
• Related work
• Conclusion
Introduction

• Common subexpression elimination
  – Two subexpressions are *common* if they that are syntactically or semantically equivalent
    e.g., \( a+b \) in expressions \( a+b \) and \( a+b+c \)
    \( a+b \) in expressions \( a+b \) and \( b+a+c \)
  – Replace the common subexpressions by a new variable to improve efficiency
    \( (a+b) / (a+b+c) \Rightarrow d/(d+c) \) with \( d=a+b \)
• SAT problem
  – Given a propositional formula, find an assignment of the involved variables such that the formula is satisfiable
  – SAT problem has many applications
  – Efficient SAT solvers are the key behind the successful application of SAT to many important applications including verification, planning and logic programming.
• CSE and SAT solving
  – Given the NP-completeness of SAT problems and its importance, continuous efforts have been made to improve the efficiency of SAT solving
  – CSE may be useful for SAT solving
    • Example: \{\{a, b, c, d\}, \{c, d, e\}\}
    • After CSE: \{\{a, b, X\}, \{X, e\}, \{X \leftrightarrow c, d\}\}. If e is false, we have X being true and thus the first clause is satisfied – causing more propagation.
CSE in SAT

• SAT formula is a set of clauses which are a set of literals. A literal is either a propositional variable or its negation.
• Any subset of a clause is called a subexpression.
• A subexpression is common w.r.t. clauses $C_1$, ..., $C_m$ if for all $i$, it is a subset of $C_i$. 
• CSE Problem in SAT
  – There are many ways to formulate the problem
  – As an informal example, given a formula, CSE is to eliminate the common subexpressions in such a way that the resulting problem is minimal in terms of its size.
CSE algorithms

• Frequency based algorithm
• Compression algorithm based approach
Frequency based CSE

• Eliminate the most frequent common subexpressions of all clauses first,
• Repeat this process until there is no common subexpressions with the frequency higher than a preset threshold.
• Example
  – Clauses: \{a, b\}, \{a, b, c\}, \{a, b, c, d\}
  – Let the preset threshold be 2
  – The most frequent is \{a, b\}. Replace it by e.
    • \{e\}, \{e, c\}, \{e, c, d\}, \{e \leftrightarrow a, b\}
  – The new most frequent is \{e, c\}, Replace it by f
    • \{e\}, \{f\}, \{f, d\}, \{e \leftrightarrow a, b\}, \{f \leftrightarrow e, c\}
• Background knowledge
  – Data item set mining
    • Given a collection of sets, a data item set mining is able to list all subsets of these sets such that each of them has an occurrence frequency higher than a given threshold
    • Example: \{a, b\}, \{a, b, c\}, \{a, b, c, d\}. Results of frequent (> 1) items are: \{a\}, \{b\}, \{a, b\}: 3, \{a, b, c\}, \{b, c\}, \{a, c\}: 2
• An approximate frequency based algorithm
  – Using *data item set* mining to build a table of subclauses and their frequency
  – Select the most frequent from the table and eliminate it
  – Update the frequency if subclasses in the table due to the elimination of the clause
LZW based approach

- LZW is a well-known compression algorithm. E.g., it is used in gzip
- Key idea of LZW:
  - To compress a file, it will scan a file and put any new string (it didn’t see before) into a dictionary and give the new string a code. When see the same string next time, it use the code of the string in the compressed file
  - Clearly, LZW is doing some kind of subcommon expression elimination
• LZW based CSE
  – If we take a formulas as a string, we can apply (and modify) the LZW algorithm to do CSE
  – Pros: fast
  – Cons:
    • CSE is done is a very greedy way
    • Missed some common subexpression, e.g., it misses \( \{a, b\} \) in processing a formula \( \{a, b\}, \{a, c, b\} \) because it can only processing substrings which are consecutive.
Empirical results

- Benchmarks: from SAT competition
- Purpose of experiments: how CSE impact SAT solving
- Measurement: the clock time used to solve a SAT problem and one that is after CSE.
- Note: we didn’t count the elimination time.
Empirical results

- LZW based algorithm
• Frequency based CSE
• Frequency = 3; size = 2
• Frequency = 2; size = 3;
Related work

• Compression based work
  – LZW based algorithm
  – Smallest grammar problem
• Common subexpression elimination in compiler
• CSE in declarative programming
Conclusion

• When using a proper CSE algorithm, significant improvement of problem solving efficiency can be observed in experiments on problems from SAT competition.

• However, it is also observed that for some problems, the efficiency is adversely affected.

• Future work: a more controlled excrement can be used:
  – modify a SAT solver such that variable ordering is not affected (suggested by Michael Gelfond)
  – How effective is CSE in ASP grounding?