Optimizing Description
Logic Subsumption

Maryam Fazel-Zarandi
Department of Computer Science
University of Toronto
Outline

- Introduction
- Optimization Techniques
- Comparison with Other Systems
- Comparing Optimizations
- Discussion
Introduction

- Realistic applications typically require:
  - expressive logics
  - acceptable performance from the reasoning services

- The usefulness of Description Logics (DLs) in applications has been hindered by the basic conflict between expressiveness and tractability.

- Early experiments with DLs indicated that performance was a serious problem, even for logics with relatively limited expressive powers.
Introduction

- Terminological reasoning in a DL based Knowledge Representation System is based on determining subsumption relationships with respect to the axioms in a KB.

- Procedures for deciding subsumption (or equivalently satisfiability) in DLs have high worst-case complexities, normally exponential with respect to problem size.

- Empirical analyses of real applications have shown that the kinds of construct which lead to worst case intractability rarely occur in practice.
Syntax and Semantics:
- DLs are formalisms that support the logical description of concepts and roles.

Tableaux subsumption testing algorithm
- “Using an Expressive Description Logic: FaCT or Fiction?”
- **Problem**: The algorithm is too slow to form the basis of a useful DL system.
- **Solution**: Employ optimization techniques.
Outline

- Introduction
  - Optimization Techniques
  - Comparison with Other Systems
  - Comparing Optimizations
  - Discussion
Optimization Techniques
Different Optimization Techniques

- Preprocessing optimizations
  - Lexical Normalization and Simplification
  - Absorption

- Partial ordering optimizations

- Satisfiability optimizations
  - Semantic Branching Search
  - Local Simplification
  - Dependency Directed Backtracking
  - Heuristic Guided Search
  - Caching Satisfiability Status
Lexical Normalization & Simplification

- Concepts in negation normal form.
  - An atomic concept and its negation in the same node label → clash!
  - Not good for concept expressions, the negation is in NNF

- Normalization:
  - Transform concept expressions into a lexically normalized form
  - Identify lexically equivalent expressions

- Simplification:
  - Eliminate redundancy
  - Identify obvious satisfiability and unsatisfiability

<table>
<thead>
<tr>
<th>Concept expression</th>
<th>Normal form</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊥</td>
<td>¬T</td>
</tr>
<tr>
<td>C ∪ D</td>
<td>¬(¬C ∩ ¬D)</td>
</tr>
<tr>
<td>∃R.C</td>
<td>¬(∀R.¬C)</td>
</tr>
<tr>
<td>¬¬C</td>
<td>C</td>
</tr>
<tr>
<td>C ∩ D</td>
<td>C</td>
</tr>
<tr>
<td>∩{∩{C_1,…,C_n},…}</td>
<td>∩{C_1,…,C_n,…}</td>
</tr>
<tr>
<td>∩{C}</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Normalisation rules for FaCT and DLP

<table>
<thead>
<tr>
<th>Concept expression</th>
<th>Simplification</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀R.T</td>
<td>T</td>
</tr>
<tr>
<td>∩{T, C,…}</td>
<td>∩{C,…}</td>
</tr>
<tr>
<td>∩{¬T,…}</td>
<td>¬T</td>
</tr>
<tr>
<td>∩{C, ¬C,…}</td>
<td>¬T</td>
</tr>
</tbody>
</table>

Table 4. Lexical simplification rules for FaCT and DLP
Lexical Normalization & Simplification

- **Example:** \[ \exists R. (C \cap D) \cap \forall R. \neg C, \]
  \[ \cap \{ \neg (\forall R. \neg \cap \{C, D\}), \forall R. \neg C \}, \]

- **Advantages:**
  - Easy to implement.
  - Subsumption/satisfiability problems can often be simplified, and sometimes even completely avoided.
  - The elimination of redundancies and the sharing of syntactically equivalent structures may lead to the KB being more compactly stored.

- **Disadvantage:**
  - For very unstructured KBs there may be no benefit, and it might even slightly increase size of KB.
Absorption

- General axioms are costly to reason with due to the high degree of non-determinism that they introduce.
  - Eliminate general axioms from the KB whenever possible

- Absorption is a technique that tries to eliminate general inclusion axioms \((C \subseteq D)\) by absorbing them into primitive definition axioms.

- Example:

  \[
  \text{geometric-figure} \cap \exists \text{angles.three} \subseteq \exists \text{sides.three} \quad \Rightarrow \quad \text{geometric-figure} \subseteq \text{figure}
  \]

  \[
  \text{geometric-figure} \subseteq \exists \text{sides.three} \cup \neg \exists \text{angles.three}
  \]

  \[
  \text{geometric-figure} \subseteq \text{figure} \cap (\exists \text{sides.three} \cup \neg \exists \text{angles.three}).
  \]
Absorption

- Advantages:
  - It can lead to a dramatic improvement in performance.
  - It is logic and algorithm independent.

- Disadvantage:
  - Overhead required for the pre-processing, although this is generally small compared to classification times.
Different Optimization Techniques

- Preprocessing optimizations
  - Lexical Normalization and Simplification
  - Absorption

- Partial ordering optimizations

- Satisfiability optimizations
  - Semantic Branching Search
  - Local Simplification
  - Dependency Directed Backtracking
  - Heuristic Guided Search
  - Caching Satisfiability Status
DL systems are often used to classify a KB, that is to compute a partial ordering or *hierarchy* of named concepts in the KB based on the subsumption relationship.

Must ensure that the classification process uses the smallest possible number of subsumption tests.

Algorithms based on traversal of the concept hierarchy
- Compute a concept’s subsumers by searching down the hierarchy from the top node (the *top search* phase)
- Compute a concept’s subsumees by searching up the hierarchy from the bottom node (the *bottom search* phase).
Optimizing Classification

- Advantages:
  - It can significantly reduce the number of subsumption tests required in order to classify a KB [Baader et al., 1992a].
  - It is logic and algorithm independent.

- It is used (in some form) in most implemented DL systems.
Different Optimization Techniques

- Preprocessing optimizations
  - Lexical Normalization and Simplification
  - Absorption

- Partial ordering optimizations

- Satisfiability optimizations
  - Semantic Branching Search
  - Local Simplification
  - Dependency Directed Backtracking
  - Heuristic Guided Search
  - Caching Satisfiability Status
Semantic Branching Search

- **Syntactic branching:**
  - Choose a disjunction \((C_1 \lor \ldots \lor C_n)\)
  - Search the different models obtained by adding each of the disjuncts

- Alternative branches of the search tree are not disjoint \(\Rightarrow\) recurrence of an unsatisfiable disjunct in different branches.

- **Semantic branching:**
  - Choose a single disjunct \(D\)
  - Search the two possible search trees obtained by adding \(D\) or \(\neg D\)
Advantages:
- It is DPLL based. A great deal is known about the implementation and optimization of this algorithm.
- It can be highly effective with some problems, particularly randomly generated problems.

Disadvantages:
- It is possible that performance could be degraded by adding the negated disjunct in the second branch of the search tree:
  - Example: if the disjunct is a very large or complex concept.
- Its effectiveness is problem dependent.
Simplification

A technique used to reduce the amount of branching in the expansion of node labels:
- Deterministically expand disjunctions in $L(\chi)$ that present only one expansion possibility.
- Detect a clash when a disjunction in $L(\chi)$ has no expansion possibilities.

Also called boolean constraint propagation (BCP)
- The inference rule $\frac{-C_1, \ldots, -C_n, C_1 \cup \ldots \cup C_n \cup D}{D}$ being used to simplify expressions.
Simplification

- Example:
  - \( \{(C \sqcup (D_1 \cap D_2)), (\neg D_1 \sqcup \neg D_2), \neg C \} \subseteq L(\chi) \)
  - \( \neg C \in L(\chi) \rightarrow \) deterministically expand \( (C \sqcup (D_1 \cap D_2)) \rightarrow \) add both \( D_1 \) and \( D_2 \) to \( L(\chi) \)
  - Identify \( (\neg D_1 \sqcup \neg D_2) \) as a clash
  - No branching

- Advantages:
  - It is applicable to a wide range of logics and algorithms.
  - It can never increase the size of the search space.

- Disadvantages:
  - It may be costly to perform without using complex data structures [Freeman, 1995].
  - Its effectiveness is relatively limited and problem dependant.
    - Most effective with randomly generated problems, particularly those that are over-constrained.
Dependency Directed Backtracking

- **Trashing:**
  - Inherent unsatisfiability concealed in sub-problems can lead to large amounts of unproductive backtracking search.

- **Example:**
  \[ \mathcal{L}(x) = \{(C_1 \cup D_1), \ldots, (C_n \cup D_n), \exists R.(C \cap D), \forall R.\neg C\} \]
Dependancy Directed Backtracking

- Allows rapid recovery from bad branching choices
- Most commonly used technique is backjumping
  - Tag concepts introduced at branch points
  - Expansion rules combine and propagate tags
  - On discovering a clash, identify most recently introduced concepts involved
  - Jump back to relevant branch points without exploring alternative branches
  - Effect is to prune away part of the search space

- Highly effective — essential for usable system
  - E.g., GALEN KB, 30s (with) → months++ (without)
Dependency Directed Backtracking

- Advantages:
  - It can lead to a dramatic reduction in the size of the search tree and thus a huge performance improvement.
  - The size of the search space can never be increased.

- Disadvantage:
  - The overhead of propagating and storing the dependency sets.
Heuristic Guided Search

- Guide the search → try to minimize the size of the tree.

- MOMS heuristic:
  - Branch on the disjunct that has the maximum number of occurrences in disjunctions of minimum size → maximizes the effectiveness of BCP

- JW heuristic: (a variant of MOMS)
  - Consider all occurrences of a disjunct, weight them according to the size of the disjunction in which they occur.
  - Select the disjunct with the highest overall weighting.

- Oldest-First heuristic:
  - Use dependency sets to guide the expansion → maximizes the effectiveness of backjumping.
  - Choose a disjunction whose dependency set does not include any recent branching points.
Heuristic Guided Search

- **Example:**
  -\{(C \cup D_1), \ldots, (C \cup D_n)\} \subseteq \mathcal{L}(\chi)
  - When \(C\) is added to \(\mathcal{L}(\chi)\), all of the disjunctions are fully expanded
  - When \(\neg C\) is added to \(\mathcal{L}(\chi)\), BCP will expand all of the disjunctions

- **Advantages:**
  - They can be used to complement other optimizations.
  - They can be selected and tuned to take advantage of the kinds of problem that are to be solved (if this is known).

- **Disadvantages:**
  - They can add a significant overhead.
  - Heuristics can interact adversely with other optimizations.
  - Heuristics designed to work well with purely propositional reasoning may not be particularly effective with DLs, where much of the reasoning is modal.
Caching

- During a satisfiability test there may be many successor nodes created.
  - These nodes tend to look very similar.
  - Considerable time can be spent re-performing the computations on nodes that end up having the same label.
    - The satisfiability algorithm only cares whether a node is satisfiable or not → this time is wasted.

- Successors are only created when other possibilities at a node are exhausted → The entire set of concept expressions that come into a node label can be generated at one time.

- The satisfiability status is determined by this set of concept expressions.
  - Other nodes with the same set of initial formulae will have the same satisfiability status → saves a considerable amount of processing.
Caching

- **Advantages:**
  - It can be highly effective with some problems, particularly those with a repetitive structure.
  - It can be effective with both single satisfiability tests and across multiple tests (as in KB classification).

- **Disadvantages:**
  - Retaining node labels and their satisfiability status involves a storage overhead.
  - The adverse interaction with dependency directed backtracking
  - Its effectiveness is problem dependent.
    - Highly effective with some hand crafted problems,
    - Less effective with realistic classification problems,
    - Almost completely ineffective with randomly generated problems.
Outline

✓ Introduction
✓ Optimization Techniques
  ▪ Comparison with Other Systems
  ▪ Comparing Optimizations
  ▪ Discussion
Comparison with Other Systems
Effectiveness of the Optimizations

- Schild has shown that determining subsumption in expressive DLs is equivalent to determining satisfiability of formulae in propositional modal or dynamic logics.

- Four systems were tested:
  - Optimized DL systems:
    - FaCT ✓
    - DPL ✓
  - Unoptimized DL system:
    - KRIS ✓
    - CRACK
  - Heavily-optimized reasoner for propositional modal logics:
    - KSAT ✓

- Neither KRIS nor KSAT can be used on all tests.
  - Neither handle transitive roles.
  - KSAT cannot handle a knowledge base.
Test Suite 1 - Tableaux’98

- A propositional modal test suite.

- Consists of 9 classes of formulae, in both provable and non-provable forms, for each of \( K \), \( KT \), and \( S4 \).

- 21 examples of exponentially increasing difficulty for each class of formula
  - The increase in difficulty is achieved by increasing the modal depth.

- **Test methodology**: ascertain the number of the largest formula of each type that the system is able to solve within 100 seconds of CPU time.

- Results: FaCT and DLP outperformed the other systems, with DLP being a clear winner.
Neither KSAT nor KRIS can be used to perform $S_4$ satisfiability tests (They can't reason with transitive roles).

Table 5. Results for $K$ and $KT$

<table>
<thead>
<tr>
<th></th>
<th>FaCT</th>
<th>DLP</th>
<th>KSAT</th>
<th>Kris</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch</td>
<td>$p$</td>
<td>$n$</td>
<td>$p$</td>
<td>$n$</td>
</tr>
<tr>
<td>$cl$</td>
<td>$&gt;20$</td>
<td>$8$</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
</tr>
<tr>
<td>grz</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
<td>$19$</td>
<td>$13$</td>
</tr>
<tr>
<td>lin</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
<td>$17$</td>
<td>$&gt;20$</td>
</tr>
<tr>
<td>path</td>
<td>$7$</td>
<td>$6$</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
</tr>
<tr>
<td>ph</td>
<td>$6$</td>
<td>$7$</td>
<td>$7$</td>
<td>$9$</td>
</tr>
<tr>
<td>poly</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
<td>$13$</td>
<td>$12$</td>
</tr>
<tr>
<td>$t4p$</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
<td>$10$</td>
<td>$18$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>FaCT</th>
<th>DLP</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_4$</td>
<td>$p$</td>
<td>$n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$45$</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
</tr>
<tr>
<td>branch</td>
<td>$2$</td>
<td>$&gt;20$</td>
<td>$18$</td>
<td>$12$</td>
</tr>
<tr>
<td>grz</td>
<td>$5$</td>
<td>$4$</td>
<td>$10$</td>
<td>$&gt;20$</td>
</tr>
<tr>
<td>ipc</td>
<td>$8$</td>
<td>$4$</td>
<td>$3$</td>
<td>$&gt;20$</td>
</tr>
<tr>
<td>md</td>
<td>$2$</td>
<td>$&gt;20$</td>
<td>$15$</td>
<td>$15$</td>
</tr>
<tr>
<td>path</td>
<td>$5$</td>
<td>$4$</td>
<td>$7$</td>
<td>$&gt;20$</td>
</tr>
<tr>
<td>ph</td>
<td>$&gt;20$</td>
<td>$2$</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
</tr>
<tr>
<td>$s5$</td>
<td>$5$</td>
<td>$3$</td>
<td>$&gt;20$</td>
<td>$&gt;20$</td>
</tr>
</tbody>
</table>

Table 6. Results for $S_4$
Another propositional modal test suite

The method uses a random generator to produce formulae. Each formula is a conjunction of $L K$-clauses

- A $K$-clause is a disjunction of $K$ elements, each element being negated with a probability of 0.5.
- An element is either:
  - a modal atom of the form $\forall R.C$, where $C$ is a $K$-clause
  - a propositional variable chosen from the $N$ propositional variables that appear in the formula, at the maximum modal depth $D$.

2 sets of formulae:

- **PS12** with $N = 4$, $K = 3$, and $D = 1$
- **PS13** with $N = 6$, $K = 3$, and $D = 1$

The test sets are created by varying $L$ from $N$ to $30N$, and generating 100 formulae for each integer value of $L/N$.

- For SAT problems, when the other parameters are fixed, the value of $L/N$ determines the “hardness” of formulae.
The performance differences between FaCT, DLP, and KSAT are much less marked.

- With such small number of literals, the purely propositional problems at depth 1 can almost always be solved deterministically.
Test Suit 3 – Expressive KBs

- Take an expressive knowledge base and construct a version of it that is acceptable to FaCT, DLP, KRIS, and CRACK.

- GALEN knowledge base
  - High level ontology

- Test KB construction:
  - Translate the GRAIL syntax of the GALEN KB into the standard syntax
  - Eliminate concept inclusion axioms by using absorption
  - Discard all role axioms

- The resulting KB contains 2,719 named concepts and 413 roles.
Test Suit 3 – Expressive KBs

- Results:
  - Neither KRIS nor CRACK was able to classify the KB
  - FaCT classified the KB in 211 seconds.
  - DLP did so in 70 seconds

- Testing other KBs:
  - They are too small or too simple
Outline

✓ Introduction
✓ Optimization Techniques
✓ Comparison with Other Systems
  ▪ Comparing Optimizations
  ▪ Discussion
Comparing Optimizations
Comparing Optimizations

- The comparison with other systems does not show which of the optimizations are most effective.

- Recent versions of DLP have compile-time configuration options.

- 22 configurations – Each was ran over the 3 test suites.

Results:
- Test Suite 1:
  - Caching – Backjumping – Semantic Branching
- Test Suite 2:
  - Normalization – Semantic Branching – Backjumping – BCP
- Test Suite 3:
  - Backjumping – Caching – Semantic Branching
  - Without absorption, satisfiability could not be proved by either FaCT or DLP
Outline

✓ Introduction
✓ Optimization Techniques
✓ Comparison with Other Systems
✓ Comparing Optimizations

▪ Discussion
Discussion

- To be useful in realistic applications, DL systems need both expressive logics and fast reasoners.

- Effective optimization techniques can make a dramatic difference in the performance of knowledge representation systems based on expressive DLs.

- These techniques can operate at every level of a DL system:
  - Simplify the KB,
  - Reduce the number of subsumption tests required to classify it,
  - Substitute tableaux subsumption tests with less costly tests,
  - Reduce the size of the search space resulting from non-deterministic tableaux expansion.
The most effective of these optimizations are absorption and backjumping:
- Impose a very small additional overhead,
- Can dramatically improve typical case performance,
- Hardly ever degrade performance (to any significant extent).

Other widely applicable optimizations include normalization, semantic branching and local simplification.

Various forms of caching can also be highly effective, but they do impose a significant additional overhead in terms of memory usage, and can sometimes degrade performance.

Heuristic techniques, at least those currently available, are not particularly effective and can often degrade performance.
Discussion

- Several exciting new application areas for very expressive DLs:
  - Reasoning about DataBase schemata and queries
  - Providing reasoning support for the Semantic Web.
  - Require logics even more expressive than those implemented in existing systems.
  - The challenge is to demonstrate that highly optimized reasoners can provide acceptable performance even for these logics.

- Given the immutability of theoretical complexity, no (complete) implementation can guarantee to provide good performance in all cases.

- The objective of optimized implementations is to provide acceptable performance in typical applications.
THANK YOU!!

Any Questions ???
Propositional Modal Logic

- **Syntax**
  - Propositional logic
  - Modal operators
    - - necessarily (box)
    - - possibly (diamond)

- **K:**
  - Necessitation Rule: If A is a theorem of K, then so is □A.
  - Distribution Axiom:
    - □(A→B) → (□A→□B).