An Experimental Evaluation of Global Caching for $\mathcal{ALC}$ (System Description)

Rajeev Goré and Linda Postniece

The Australian National University

Introduction

- $\mathcal{ALC}$ is the basis of modern description logics
- $\mathcal{ALC}$ is a multi-modal version of basic modal K
- EXPTIME-complete problem:
  is φ satisfiable w.r.t. TBox axioms (global assumptions) Γ?
- Optimisations are crucial
- Caching reuses results of subcomputations
- Global caching is a new method (Goré and Nguyen 2007)

How does global caching compare to other caching methods?

Tableau Terminology

- Closed = `unsat`
- Open = possibly `sat` (expanded and not closed)
- DFS: Depth First Search

Unsophisticated Caching Methods

NC: No caching

- DFS with ancestor equality blocking

UC: Unsat caching

- DFS with ancestor equality blocking
- All closed (`unsat`) nodes are cached globally

Sophisticated Caching Methods

MC: Mixed caching (Donini and Massacci 2000)

- Search consists of a sequence of runs
- Each run builds an And-tree (model)
- Or-choices give different runs (models)
- DFS with ancestor equality blocking
- All closed (`unsat`) nodes are cached globally
- Open (possibly `sat`) nodes within the current And-tree are cached
- Globally caching open nodes can lead to unsoundness

GC: Global caching (Goré and Nguyen 2007)

- Search builds an And-Or graph
- Or-choices give Or-nodes in graph
- All nodes are cached, regardless of satisfiability status
- Every node only expanded once
- `sat`/`unsat` status propagated through And-Or graph
- Any search strategy suffices

Implementation and Data Structures

- All methods implemented using same framework / optimisations
- Prototype implemented using C++ and STL
- Cache is a `std` :: `map` (red-black tree)
- Nodes are sets of formulae, stored as bitstrings
- Each formula has a unique index
- *i*-th bit true iff formula with index *i* is in the node
- Edges are pointers between nodes
- No labels or individual names, hence cannot handle ABoxes
- Expansion strategies:
  - pure-DFS for NC, UC, MC, GC-DFS
  - heuristic-DFS for GC-Custom

Key Optimisations

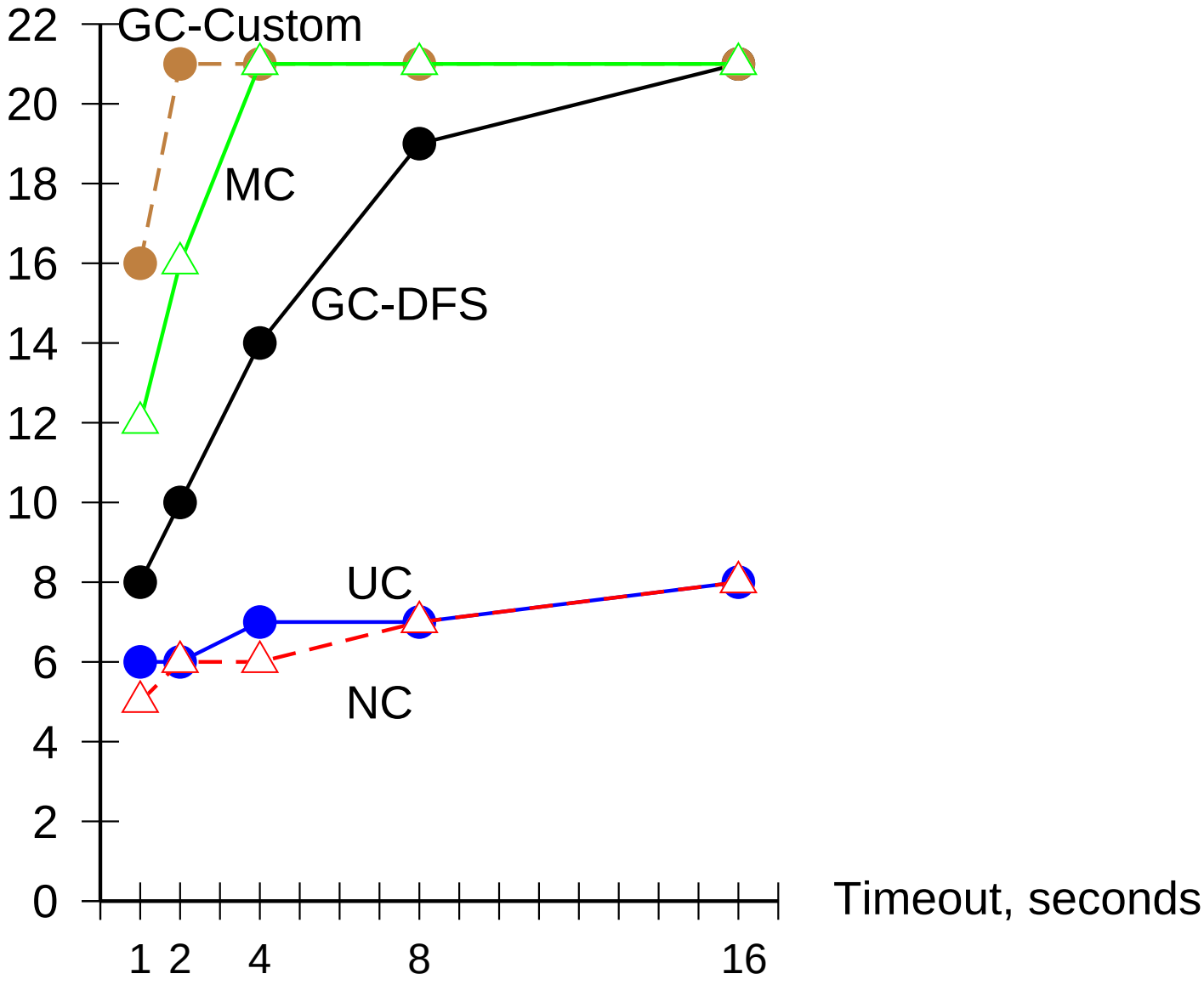
- Negation normal form
- Semantic branching for atoms
- Node normalisation (MP, subsumption, implicit And rule)
- Backjumping
- Lazy unfolding of TBox axioms

Test Data

- LWB: test formulae for modal logics K, S4, KT
  - 54 formula sets (problems)
  - Monomodal $\mathcal{ALC}$, no TBox axioms (essentially PSPACE)
- DL98: K Tbox
  - 18 formula sets (problems)
  - Monomodal $\mathcal{ALC}$ with TBox axioms (EXPTIME)
- Hence 72 formula sets (problems)
- Each set contains formulae 1..21 of increasing complexity
- Each test is a formula with a timeout
- Tests 1..21 continue until some formula exceeds timeout
- Result for a set and timeout is highest formula number solved

Representative Result for Problem k_d4_n (PSPACE)

Highest formula number solved

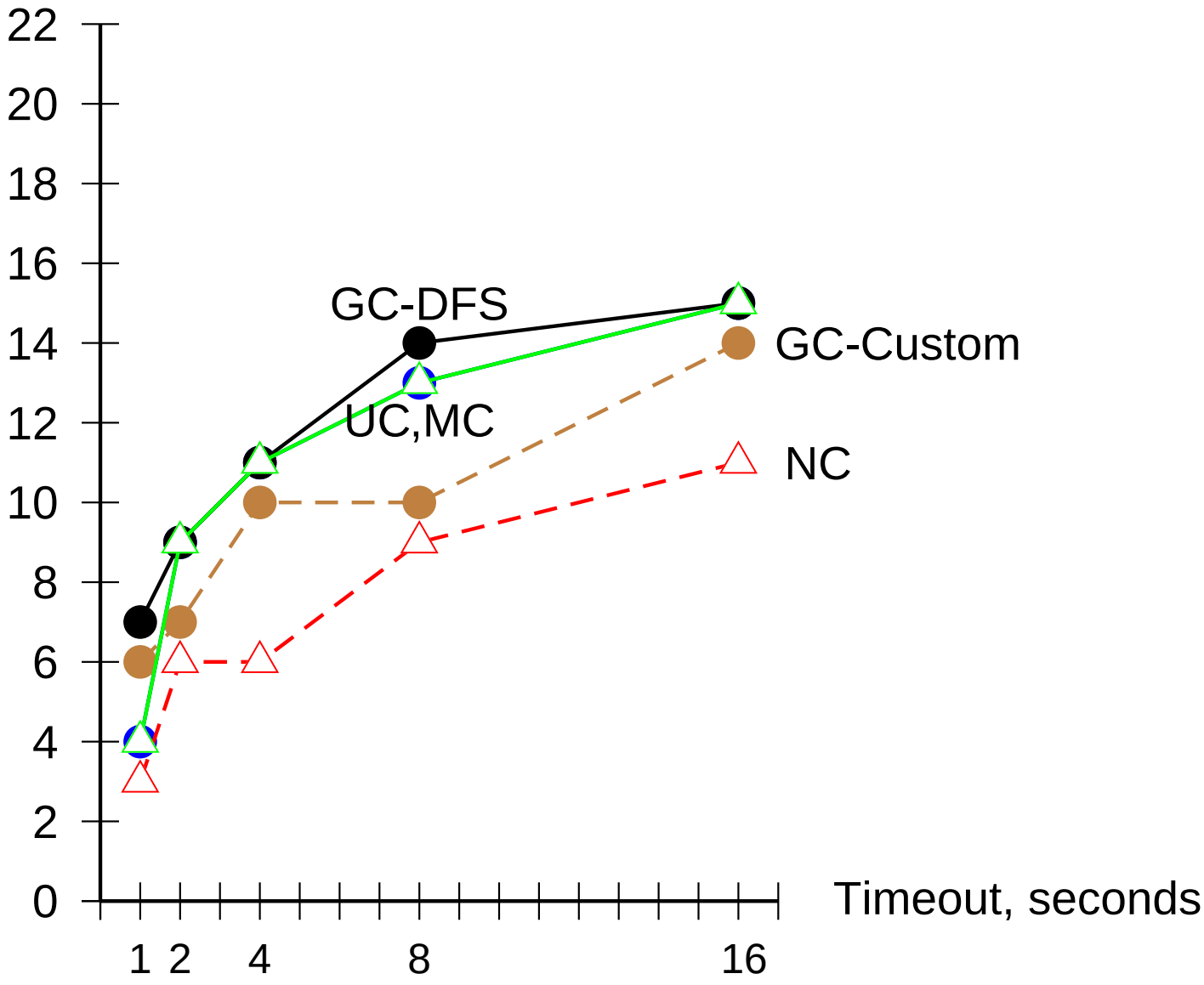


Results - K (PSPACE)

- Sophisticated better than Unsophisticated (as expected)
- The best of GC-DFS and GC-Custom usually better than MC
- No clear winner between GC-DFS and GC-Custom

Representative Result for Problem k_grz_p (EXPTIME)

Highest formula number solved



Results - K TBox (EXPTIME)

- Often the difference between Sophisticated and Unsophisticated is not as marked
- The best of GC-DFS and GC-Custom is on par or better than MC

Conclusions and Further Work

- The best of the two types of global caching is on par or better than other caching methods on all problems except two out of 72
- Vital to investigate good heuristics for global caching, since it is not tied to the DFS framework
- Source code and test data available at `http://users.rsise.anu.edu.au/~linda/CWB.html`