CP2001

Components for State Restoration in Tree Search

Martin Henz

Seminar of the School of Computing

National University of Singapore

Constraint-based Tree Search

- sequential search
- heavy reuse of data structures down the tree
- fixpoint computation w.r.t filtering algorithms at each node

Problem: How to go from B to A

- going from A to B is easy: propagation and branching
- going from B to A is hard
- issue: efficient implementation w.r.t. time and space

Current Approach I: Trailing

- Inherited from Prolog
- Idea: trail changes and undo from below
- Used in almost every CP system (ECLIPSE, ILOG Solver, CHIP, ...)

Current Approach II: Recomputation

- Born out of necessity for concurrent constraint programming
- Idea: copy states and recompute state from above; re-execute branching on the way down
- Used in Oz/Mozart [Schulte 1997, Schulte 1999, Schulte 2001]
- extreme form: full copying
- smart form: adaptive recomputation [Schulte 1999, Schulte 2001]

Discussion

- Both approaches copy state
- Trailing: fine-grained copying, on demand, optimistic
- Recomputation: bulk copying, informed guess, pessimistic
- Remember where the work in trees is!
- Recomputation needs reproducible choices

Issues

- Implementation:
  - Trailing: involves data structures and operations
  - Recomputation: involves only data structures
- Trade-off between space and time
- Other ways of going from B to A?

Some New Ways; Restoration Policies

- Lazy Copying
- Batch Recomputation

Lazy Copying

- Idea: Copy-on-write
- widely used in operating systems
- also used in Or-parallel and And-parallel Prolog implementations
- optimistic/pessimistic? Realistic!

Engineering Issues for Lazy Copying

- Pointer problems in usual implementations
- Solution: relative addressing
- Relative addressing is beneficial in overall system design

Batch Recomputation

- Observation: recomputation reaches many fixpoints on the way to B
- Compress fixpoint computation into one step
- Requirement: monotonicity of constraints

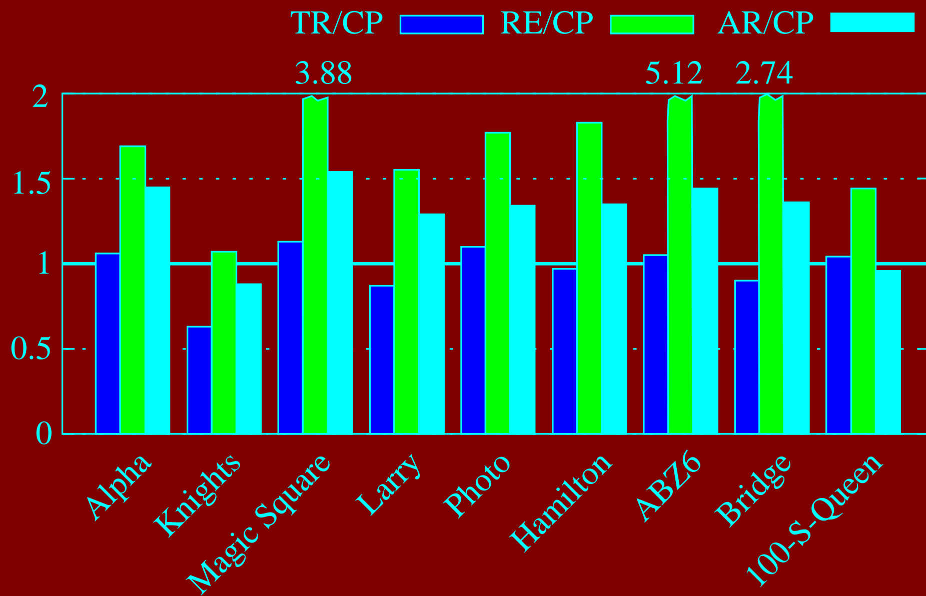
Engineering Issues for Batch Recomputation

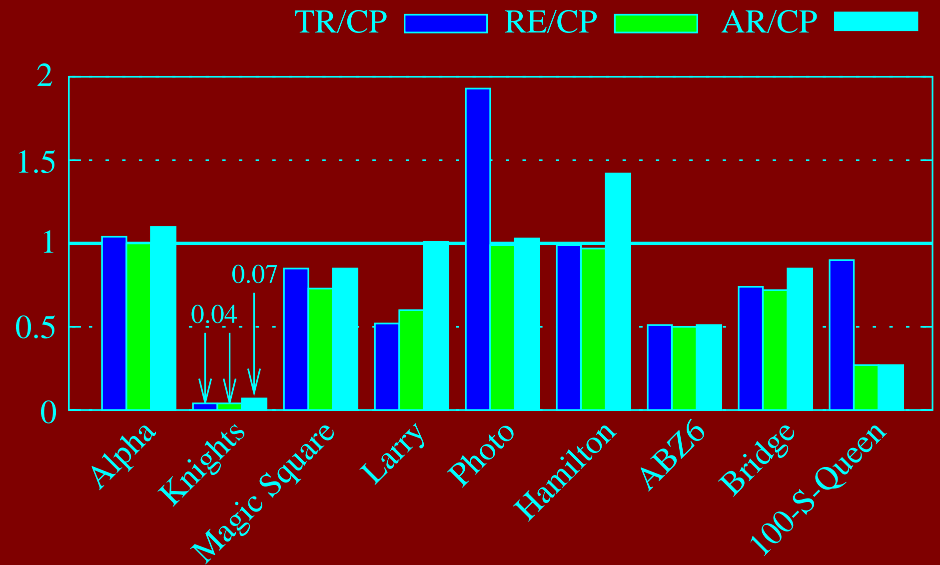
- "Trail" the branching constraints along every path
- applicable to adaptive recomputation

Experimental Evaluation: Setup

- Component-based implementation
- Node class (C++) decides restoration policy
- Allows comparison "everything else being equal"
- Based on Figaro library, a C++ library for CP(FD)
- Compare runtime and memory consumption
- 400 Mhz Pentium II, 256MB main memory, running Linux
- Test set covers range of toy and benchmark problems

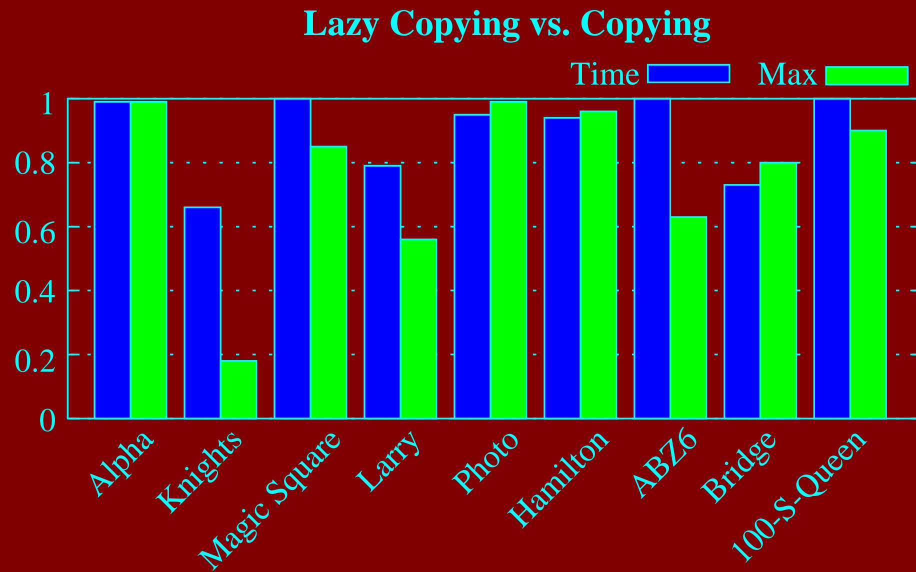
x time/copying time; x memory/copying memory

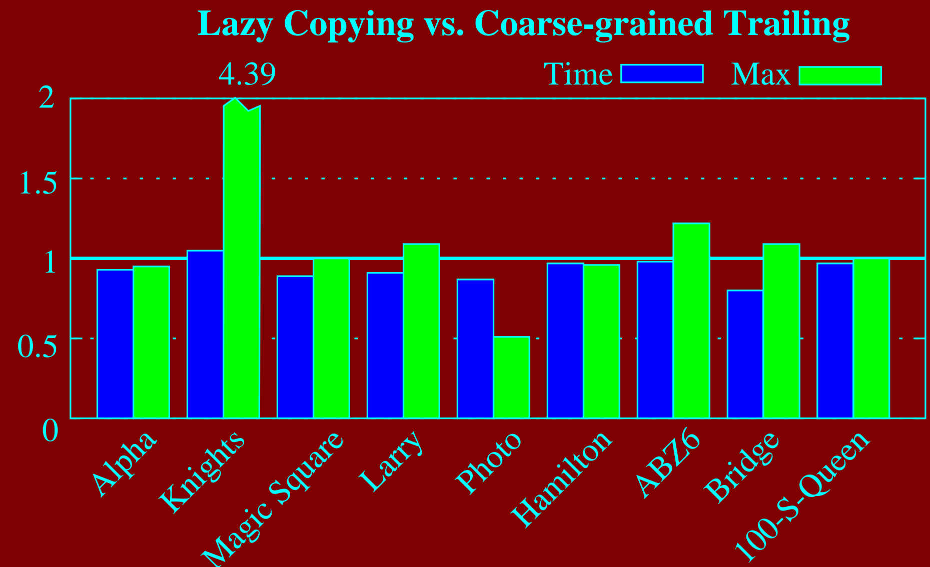




- naive recomputation needs calibration
- adaptive recomputation never breaks
- compare logic programming with constraint programming

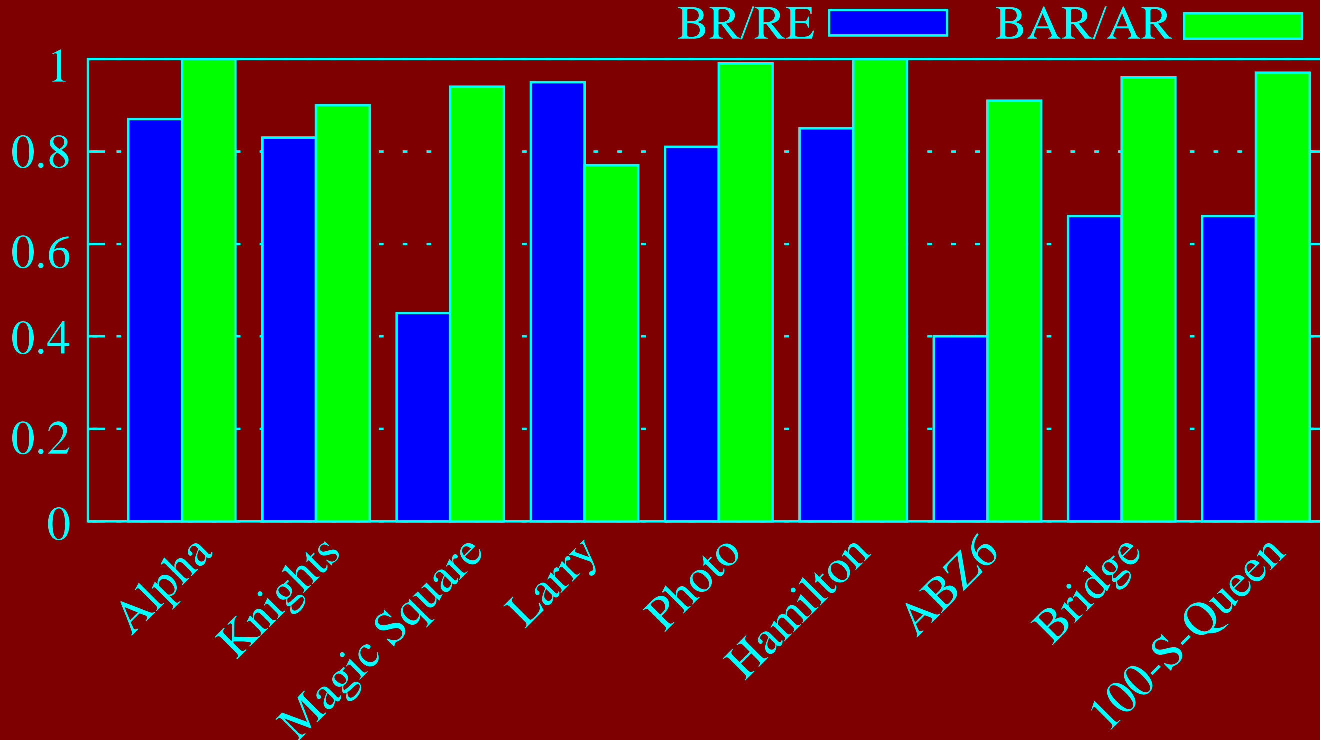
Lazy copying/copying; lazy copying/c.g. trailing





- lazy copying never worse than copying
- logic programming vs constraint programming
- lazy copying competitive with coarse grained trailing

Batch Recomputation vs Recomputation



- batch recomputation always better than recomputation
- difference smaller when using adaptive recomputation

Conclusion

- Many ways to go from B to A
- Batch recomputation: speed up recomputation by trailing decisions
- Lazy copying: incremental copying on downward move ("realistic")
- Engineering issues
- Future work: Stateful filtering algorithms