National University of Singapore
Seminar of the School of Computing

Martin Henz

Search Components for State Restoration in Tree

CP2001
node

- Fixpoint computation w.r.t. filtering algorithms at each node
- Heavy reuse of data structures down the tree
- Sequential search

Constraint-based Tree Search
Issue: Efficient implementation w.r.t. time and space

• Going from B to A is hard
• Going from A to B is easy: propagation and branching
CP2001: Components for State Restoration in Tree Search

- CHIP, ...

- Used in almost every CP system (ECLiPSe, ILOG Solver)

- Ideas: trail changes and undo from below

- Inherited from Prolog

Current Approach: Trailing

extreme form: full copying

used in Oz/Mozart [Schulte 1997, Schulte 1999, Schulte 2001]

branching on the way down

Idea: copy states and recompute state from above; re-execute

born out of necessity for concurrent constraint programming

Current Approach II: Recomputation
Recomputatation needs reproducible choices

Remember where the work in trees is!

Recomputatation: bulk copying, informed guess, pessimistic

Trailing: fine-grained copying, on demand, optimistic

Both approaches copy state

Discussion
• Other ways of going from B to A?

• Trade-off between space and time

Recomputation: Involves only data structures

Trailing: Involves data structures and operations

Implementation:
Batch Recomputation

Lazy Copying

Some New Ways; Restoration Policies
Cf2001: Components for State Restoration in Tree Search

- Optimistic/Pessimistic Realistic Mentions
- Also used in OR-parallel and AND-parallel Prolog Implic-
- Widely used in operating systems
- Idea: Copy-on-write
- Lazy Copying
Relative addressing is beneficial in overall system design

Solution: relative addressing

Pointer problems in usual implementations

Engineering Issues for Lazy Copying
Requirement: monotonicity of constraints

Compress fixpoint computation into one step

the way to B

Observation: recomputation reaches many fixpoints on

Batch Recomputation
Engineering Issues for Batch Recomputation

- “Trail” the branching constraints along every path
- applicable to adaptive recomputation
Test set covers range of toy and benchmark problems

Linux

400 MHz Pentium II, 256MB main memory, running

Compare runtime and memory consumption

Based on Figaro library, a C++ library for CP(FD)

Allows comparison, "everything else being equal"

Node class (C++) decides restoration policy

Component-based implementation

Experimental Evaluation: Setup
C P200: Components for State Restoration in Tree Search
compare logic programming with constraint programming

• adaptive recomputation never breaks

• naive recomputation needs calibration

\begin{align*}
\text{Time of } & \text{TR, RE, AR vs. CP} \\
\text{Memory of } & \text{TR, RE, AR vs. CP}
\end{align*}

\begin{align*}
\text{AR/CP} & \text{ RE/CP} \\
\text{TR/CP}
\end{align*}
Lazy copying competitive with coarse-grained trailing

Logic programmin vs constrained programming

Lazy copying never worse than copying

Lazy copying/copying; lazy copying/c.g. trailing
difference smaller when using adaptive recomputation

Batch Recomputation vs Recomputation
Future work: Stateful filtering algorithms

Engineering issues

(“realistic”)

Lazy copying: Incremental copying on downward move

Lazy decisions

Batch recomputation: Speed up recomputation by trail-

Many ways to go from B to A

Conclusion