Search Strategies: Lookback

Summary

- Lookback schemes
- Gaschnig's Backjumping
- Graph Based Backjumping
- No-Good Recording

Lookback

Basic Ideas

- lookahead schemes
  - foresee and avoid dead-ends in the future
  - operates during the forward phase
- lookback schemes: avoid repeating the same error in the search
  - avoid repeating same errors when a dea-dend is found
  - operates during the backward search

Lookback Schemes

Main Approaches

- Backjumping
  - improves on backtracking one step backwards
  - analysing the reason for the dea-dend we can avoid irrelevant backtrack points
  - jump to the source of failure (culprit)
- Constraint Recording (no-good learning)
  - record the reason for dead-end as a new constraint
  - avoid finding the same inconsistencies again

Example

Example (Graph Coloring)

- Variables: $x_1, x_2, x_3, x_4, x_5, x_6, x_7,$
- Domains: $D_{x_1} = \{B, R, G\}, D_{x_2} = D_{x_5} = \{B, G\}, D_{x_3} = D_{x_4} = D_{x_7} = \{R, B\}, D_{x_6} = \{R, G, Y\}$
- Constraints: $x_1 != x_2, x_1 != x_3, x_1 != x_4, x_1 != x_7, x_2 != x_6, x_3 != x_7, x_4 != x_5, x_4 != x_7, x_5 != x_6, x_5 != x_7$
- Backtracking with $d = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$
- Consider assignment $\{R, B, B, B, G, R\}$
- Dead-end at x_7 does not depend on x_6

Example

Example (Graph Coloring)

- (R, B, B, B, G, R) a conflict set for x_7
- $(R, -, B, B, G, -)$ another conflict set for x_7
- $(R, -, B, -, -, -)$ minimal conflict set for x_7
- (R, B, B, B, G, R) leaf dead end
- every conflict set is a no-good

Dead end

Definition (Dead End State at Level i)

- assignment $\bar{a}_i = \{a_1, \cdots, a_i\}$
- $\forall a_{i+1} \in D_{x_{i+1}}$
- $\{a_1, \cdots, a_i, a_{i+1}\}$ is inconsistent
- x_{i+1} dead end variable

Solving Dead Ends

- there might be sub-tuples of $\bar{a}_i$ that conflict with x_i
- therefore backtracking to x_i might be useless
- we should jump to x_b such that $\bar{a}_{b-1}$ contains no conflict sets for x_{i+1}

Conflict Set

Definition (Conflict Set)

- $\bar{a} = \{a_{i_1}, \cdots, a_{i_k}\}$ consistent instantiation of arbitrary subset
- x not instantiated variable
- if $\nexists b \in D_x | < \bar{a}, x = b >$ is consistent
- $\bar{a}$ is a conflict set for x
- if $\bar{a}$ does not contain a subtuple that conflicts with x, $\bar{a}$ is a minimal conflict set of x

Definition (Leaf Dead-End)

- partial solution $\bar{a}_i$ that conflicts with x_{i+1}
- x_{i+1} is the leaf dead-end variable

No-Good

Definition (No good)

- Given a network $\mathcal{R} = \{X, D, C\}$
- any partial instantiation $\bar{a}$ that does not appear in any solution of R is a no-good

No good

- any conflict set for any variable is a no-good
- there are no-good that are not conflict sets for any single variable
- For previous GC example $x_1 = R, x_2 = G$ is a no-good but is not a conflict set for any variable

Jumping back

Main idea of backjumping

- When dead-end, jump back as far as possible
- without loosing possible solutions
- maximal jump: jump as back as possible
- safe jump: do not skip any solutions

Safe Jump

Definition (Safe Jump)

- $\bar{a}_i = \{a_1, \cdots, a_i\}$ leaf dead end state
- x_j $j \leq i$ is safe (relative to $\bar{a}_i$) if
- $\bar{a}_j$ is a no-good

Note

- safety of jump is algorithm independent
- maximality is algorithm dependent

BackJumping Styles

Styles

Backtracking that backs up several layers when a not extensible assignment is found

- Gaschnig's
- Graph based
- Conflicts-directed

Gaschnig's Backjumping

When and where to jump

- When: Backtrack when leaf dead ends are found
- Where: Backtrack to the culprit variable

Culprit Variable

Definition (culprit variable)

- $\bar{a}_i = a_1, \cdots, a_i$ leaf dead end
- culprit index $\rightarrow culp(a) = min_{1 \leq j \leq i}\{a_1, \cdots, a_j\}$ conflicts with x_{i+1}

Gaschnig's Backjumping: Example

Example (GC Example)

- Consider previous GC example with same order
- Consider the following trace for backtrack: R, B, B, B, G, R
- This is a leaf dead end and Gaschnig Backjump will backtrack to x_3
- It saves searching the branch where $x_6 = Y$

Gaschnig's Backjumping: Properties

Properties

- Historically first back jump algorithm
- Performs only safe jump
- It performs maximal jumps
  - jumping further back we could loose solutions
- Can be implemented efficiently
  - Culprit variables could be marked in the forward phase

Gaschnig's Backjumping: Limitations

Limitations

- Not very powerful
  - Do not cut significant parts of the search space
  - Expands strictly more space than forward checking
- It backtrack only at leaf dead ends
  - too late, most of the work already done.
- Graph-Based Backjumping improves on this

Graph-Based Backjumping

Main Ideas

- jumps back at leaf dead ends and at internal dead ends
- uses only information derived from the graph structure to jump
  - do not use information on variables' domain or on nature of constraints

Definition (internal dead ends)

A no-good that is defined on the first i variables and is not a leaf dead end.

Jumps and internal dead ends

Example (Internal dead ends)

- Consider the GC example
- x_3 is an internal dead ends

internal dead ends

In general, x_i is an internal dead end if a backjumping algorithm jumps there from a leaf dead end, and there are no more candidate values to instantiate.

Graph-Based Backjumping: Where to Jump

Using Graph-based info

- use preceding adjacent variables as an approximation of a minimal conflict set
  - adjacent: connected on the primal graph
- assume all possible conflicts do appear
  - this really depends on the particular assignment
- Latest preceding adjacent variable is the culprit variable

Useful definitions

Definition (Ancestor, parent)

- Given constraint graph, node ordering and a variable x
- The ancestor set of x $anc(x)$ set of variable that precede and are connected to x
- The parent of x $p(x)$ is the maximal (or latest) variable in $anc(x)$

Safe Jump for GB-Backjumping

Theorem (safe jump)

- *Given $\bar{a}_i$ leaf dead end and x_{i+1} the leaf dead end variable*
- *Then $p(x_{i+1})$ is a safe jump*

Proof.

- $\bar{a}_i$ consistent but any extension to x_{i+1} inconsistent
- then $\bar{a}_i$ is a conflict set for x_{i+1}
- then also $\bar{a}_{p(x_{i+1})}$ is a conflict set for x_{i+1}
- because there are no variables x_j with $p(x_{i+1}) < j < i + 1$ that are adjacent to x_{i+1}
- since any conflict set is a no-good then the jump is safe

$\square$

Comparing to Gaschnig

comparison

- Jumping back to the parent can not be better than jumping back to culprit variable, in general it can be worse
- But we can jump from internal leaf dead end as well
- simple idea: jump back to the parent of the internal dead end variable
- this is wrong: we could loose solutions

Unsafe Jumps

Example (Unsafe Jump)

- Consider the GC example
- Suppose x_5 is leaf dead end, if $x_4 = p(x_5)$ is an internal dead end it is safe to jump back to $x_1 = p(x_4)$
- Suppose x_7 is leaf dead end, we jump to $x_5 = p(x_7)$, if x_5 is an internal dead end it is safe to jump to $x_4 = p(x_5)$, however it is not safe to jump to $x_1 = p(x_4)$
- $x_1 < x_3$ and $x_3 \in anc(x_7)$
- Lesson: to decide where to jump from an internal dead-end it does matter which variables initiated the backtrack

The concept of session

Definition (Invisit)

The backtrack algorithm invisits x_i when it attempts to extend the assignment $\bar{a}_{i-1}$ to x_i

Definition (Session)

A session for x_i starts when x_i is invisited by the backtrack algorithm and ends when the algorithm backtracks from x_i (i.e. all possible values of x_i have been tried). It contains x_i and all the later variables visited during the session.

Relevant dead ends

Definition (Relevant dead ends)

- $r(x_i) = \{x_i\}$ when x_i is invisited
- $r(x_i) = r(x_i) \cup r(x_j)$ with $j > i$ when the algorithm backtrack to x_i from x_j (dead-end)

GB-Backjump Algorithm

Algorithm

- When at a (leaf or internal) dead end $\bar{a}_i$
- Jump back to the latest ancestor of any variable in $r(x_i)$ that is before x_i (culprit for GB backjumping)

Theorem (soundness)

Graph based backjump only performs safe, maximal jumps

Conflict Directed Backjumping

Combining the approaches

- Gaschnig: exploits information about minimal prefix conflicts to jump further from leaf dead ends
- Graph-Based: exploits connectivity information to jump also at internal dead ends
- Ideas can be combined: Conflict Directed Backjumping
- Better: avoids more state than either of the two previous approaches

No-Good Learning

Avoiding thrashing

- Backjumping can avoid many irrelevant choices thus reducing the search space
- Thrashing can still happen: same no-good rediscovered over and over again
- No-good learning or constraint recording can avoid this

Adding No-Goods

Basic Idea

- Very simple concept
- When backtrack (or jump back)
- Determine a conflict set
- Add a constraint to the network that avoids that conflict set

Many possibilities

No-Good learning variations

- Shallow Learning: determine a no-good that is easy to generate but not minimal
- Deep Learning: Determine a no-good which is minimal and even derive all minimal ones from this no-good
- Bounded Learning: Store only constraint with a bounded arity (smaller than a predetermined parameter)
- Relevance bounded learning: do not record the no-good if the current state differs from the no-good in more than c variables

No-Good Learning: trade-offs

Finding a balance

- Pruning power: more no-good $\rightarrow$ less visited states
- Computation: more no-good $\rightarrow$ more effort in checking consistency
- Computation Overhead: processing no-goods to find minimal conflicts sets has a cost
- Space Overhead: storing no-good

Graph based learning

GB Learning

- In some cases finding relevant no-good is easy
  - When jumping back from a leaf dead end $\bar{a}_i$ with dead end variable x_i, $Rel(x_i) = anc(x_i)$
  - When jumping back from an internal dead end x_i, $Rel(x_i)$ the set of ancestors of all variables in $r(x_i)$ that precede x_i
  - Add the no-good $\{< x, v > | x \in Rel(x_i) \wedge v = \bar{a}_i(x)\}$