DM826 – Spring 2014

Modeling and Solving Constrained Optimization Problems

Lecture 10
Search

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Search

- Complete
  - backtracking
  - dynamic programming
- Incomplete
  - local search

Outline

1. Complete Search

2. Incomplete Search

Backtracking: Terminology

- backtracking: depth first search of a search tree
- branching strategy: method to extend a node in the tree
- node visited if generated by the algorithm
- constraint propagation prunes subtrees
- deadend: if the node does not lead to a solution
- thrashing repeated exploration of failing subtree differing only in assignments to variables irrelevant to the failure of the subtree.

Simple Backtracking

- at level j: instantiation $I = \{x_1 = a_1, \dots, x_j = a_j\}$
- branches: different choices for an unassigned variable: $I \cup \{x = a\}$
- branching constraints $\mathcal{P} = \{b_1, \dots, b_j\}$, $b_i, 1 \leq i \leq j$
- $\mathcal{P} \cup \{b^1_{j+1}\}, \dots, \mathcal{P} \cup \{b^k_{j+1}\}$ extension of a node by mutually exclusive branching constraints

(In this view, easy implementation of propagation: the branching constraints are simply scheduled for propagation)

Branching strategies

Assume a variable order and a value order (e.g., lexicographic):

A. Generic branching with unary constraints:

1. Enumeration, d-way

$$x = 1 \quad | \quad x = 2 \quad | \ldots$$

2. Binary choice points, 2-way

$$x = 1 \quad | \quad x \neq 1$$

3. Domain splitting

$$x \leq 3 \quad | \quad x > 3$$

⇝ d-way can be simulated by 2-way with no loss of efficiency. While d-way with optimla ordering of variable and values can be exponentially worse than a 2-way

⇝ 2-way seem more efficient than d-way on the same models

Branching strategies

B. Problem specific:

- Disjunctive scheduling (job-shop scheduling)
  x_i, x_j starting times of activities, d_i their duration
  on a shared resource: $x_i + d_j \leq x_j$ or $x_j + d_j \leq x_i$
  equivalent to introducing binary variables for order.

- Zykov's branching rule for graph coloring

Constraint propagation

- constraint propagation performed at each node: mechanism to avoid thrashing
- typically best to enforce domain consistency but with some exceptions (e.g., forward checking is best in SAT)
- nogood constraints added after deadend is encountered
  similar to caching or memoization techniques: record solution to subproblems and reuse them instead of recomputing them.
  Corresponds to values ruled out by higher order consistency which would be too costly to check

Nogood constraints

Definition (Nogood)

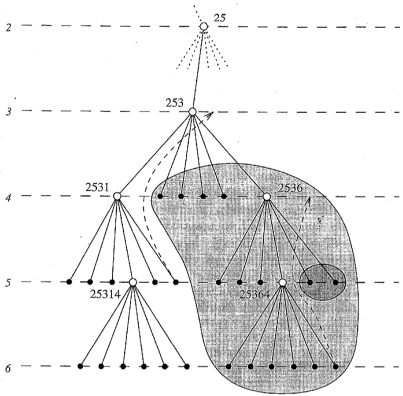
A nogood constraint is a set of assignemnts and branching constraints that is not consistent with any solution.

Implicit constraints, their addition does not remove solutions. Goal: reduce thrashing.

- Rule out inconsistencies before they are encountered during search:
  - Add implied constraints by hand during modelling
  - Automatically add them by applying constraint propagation algorithms

⇝ Rule out inconsistencies after they have been encountered
late for this node, since it has been already refuted, but it may contribute to pruning in the futre.

E.g.: On 6-queens problem:



white nodes: all constraints with some instantiated variables are satisfied
black nodes: one or more constraint checks fail
shaded area explained later

– $\{x_1 = 2, x_2 = 5, x_3 = 3\}$ is a no good: post $\neg\{x_1 = 2 \wedge x_2 = 5 \wedge x_3 = 3\}$
– Applying symmetry mapping (mirroring over x-axis): also $\{x_1 = 5, x_2 = 2, x_3 = 4\}$ is a nogood
– $(x_2 = 5) \implies (x_6 \neq 1)$

Discovering nogoods

- Let $\mathcal{P} = \{b_1 \ldots, p_j\}$ be a deadended node (b_i, $1 \leq i \leq j$, is the branching constraint posted at level i in the search tree).

- $J(\mathcal{P})$ jumpback nogood for $\mathcal{P}$ is defined recursively:
  - $\mathcal{P}$ is a leaf node. Let C be a constraint that is not consistent with p:

    $$J(\mathcal{P}) = \{b_i | vars(b_i) \cap vars(C) \neq \emptyset, 1 \leq i \leq j\}$$

  - $\mathcal{P}$ is not a leaf node. Let $\{b^1_{j+1} \ldots, p^k_{j+1}\}$ be all possible extensions of $\mathcal{P}$ attempted by the branching strategy, each of which has failed:

    $$J(\mathcal{P}) = \bigcup_{i=1}^{k} (J(\mathcal{P} \cup \{b^i_{j+1}\}) - \{b^i_{j+1}\})$$

Ex: $\mathcal{P} = \{x_1 = 2, x_2 = 5, x_3 = 3, x_4 = 1, x_5 = 4\}$, all extensions of x_6 to $\mathcal{P}$ fail:

$$\begin{aligned} J(\mathcal{P}) &= (J(\mathcal{P} \cup \{x_6 = 1\}) - \{x_6 = 1\}) \cup \ldots \cup (J(p \cup \{x_6 = 6\}) - \{x_6 = 6\}) \\ &= \{x_2 = 5\} \cup \ldots \cup \{x_3 = 3\} \\ &= \{x_1 = 2, x_2 = 5, x_3 = 3, x_5 = 4\} \end{aligned}$$

Backjumping

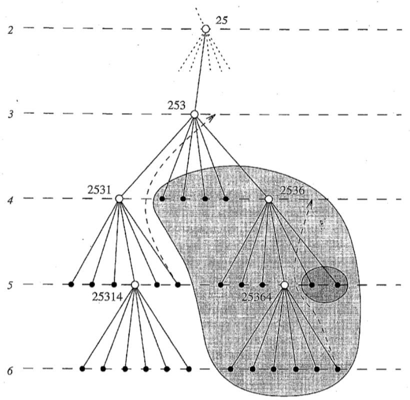
- standard backtracking: chronological backtracking: backjump to the most recently instantiated variable

- non-chronological backtracking $\equiv$ backjumping or intelligent backtracking:
  retracts the closest branching constraint that bears responsibility.

  Eg: jump back to the most recent variable that shares a constraint with deadend variable.

  Eg: $\mathcal{P} = \{b_1, \ldots, b_j\}$ non-leaf deadend
  $J(\mathcal{P}) \subseteq \mathcal{P}$ jumpback nogood for $\mathcal{P}$
  largest i, $1 \leq i \leq j : b_i \in J(\mathcal{P})$
  jumpback and retracts b_i and all those posted after b_i
  and delete nogoods recorded after b_i

E.g.: On 6-queens problem:



- deadend after failing to extend 25314. Backjump associated is $\{x_1 = 2, x_2 = 5, x_3 = 3, x_5 = 4\}$
- Backjump to $i = 5$, retracts $x_5 = 4$ (here like chronological backtr.)
- deadend discovered for 2531. Backjump nogood is $\{x_1 = 2, x_2 = 5, x_3 = 3\}$
- backjump to $i = 3$, retracts $x_3 = 3 \rightsquigarrow$ skipp all the shaded tree
- (nogood used only to backjump not for propagation, less memory usage)

Restoration Service

What do we have at the nodes of the search tree?
A computational space:

1. Partial assignments of values to variables
2. Unassigned variables
3. Suspended propagators

How to restore when backtracking?

- Trailing Changes to nodes are recorded such that they can be undone later
- Copying A copy of a node is created before the node is changed
- Recomputation If needed, a node is recomputed from scratch

Heuristics for Backtracking

Decisions must be made on Variable-Value ordering:
optimal strategy if it visits the fewest number of nodes in the search tree.
Finding optimal ordering is hard

Possible goals

- Minimize the underlying search space
- Minimize expected depth of any branch
- Minimize expected number of branches
- Minimize size of search space explored by backtracking algorithm
  (intractable to find "best" variable)

dynamic vs static strategy
In Gecode: Variable-Value Branching ch. 8 +
http://www.gecode.org/doc-latest/reference/group__TaskModelIntBranch.html

Variable ordering

dynamic heuristics:

- dom: choose x that minimizes $\text{rem}(x|\mathcal{P})$ the domain size remaining after propagation of branching constraints $\mathcal{P}$.
- dom + deg (# constraints that involve a variable still unassigned)
- $\frac{\text{dom}}{\text{wdeg}}$ weight incremented when a constraint is responsible for a deadend
- min regret
  difference between smallest and second smallest value still in the domain
- structure guided var ordering:
  instantiate first variables that decompose the constraint graph
  graph separators: subset of vertices or edges that when removed separates the graph into disjoint subcomponents

Value ordering

- estimate number of solutions:
  counting solutions to a problem with tree structure can be done in polytime
  reduce the graph to a tree by dropping constraints
- if optimization constraints: reduced cost to rank values

Variants to best search

- Limited Discrepancy search

  Discrepancy: when the search does not follow the value ordering heuristic and does not take the left most branch out of a node.

  explored tree by iteratively increasing number of discrepancies, preferring discrepancies near the root
  (thus easier to recover from early mistakes)

  Ex: ith iteration: visit all leaf nodes up to i discrepancies
  $i = 0, 1, \ldots, k$ (if $k \geq n$ depth then alg is complete)

- Interleaved depth first search
  each subtree rooted at a branch is searched for a given time-slice using depth-first.
  If no solution found, search suspended, next branch active.
  Upon suspending in the last the first again becomes active.
  Similar idea in credit based.

Randomization in Search Tree

- Dynamical selection of solution components in construction or choice points in backtracking.
- Randomization of construction method or selection of choice points in backtracking while still maintaining the method complete $\rightsquigarrow$ *randomized systematic search*.
- do backtracking until distance from a deadend has exceeded a fixed cutoff number, restart by reordering the variables
- Randomization can also be used in incomplete search

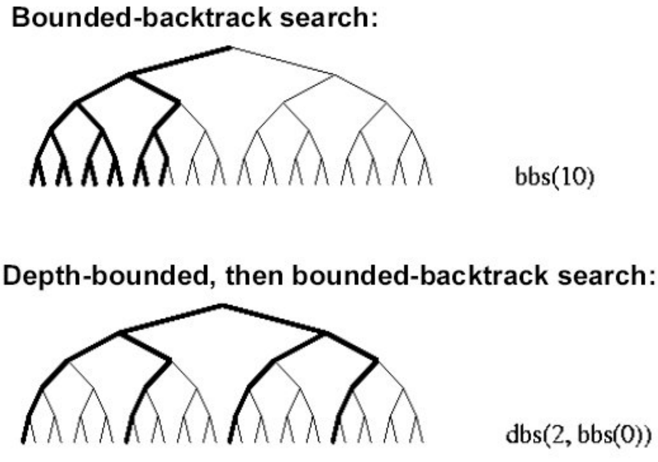
(more next time)

Outline

1. Complete Search

2. Incomplete Search

Incomplete Search

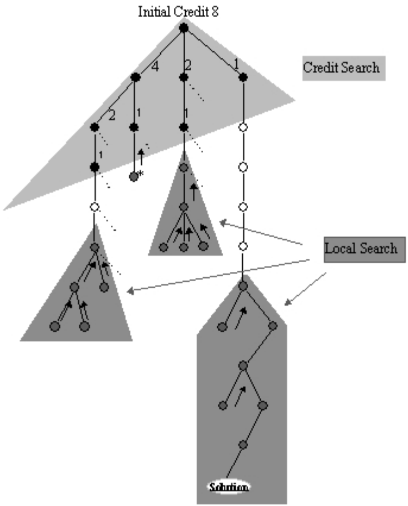


http://4c.ucc.ie/~hsimonis/visualization/techniques/partial_search/main.htm

Incomplete Search

Credit-based search

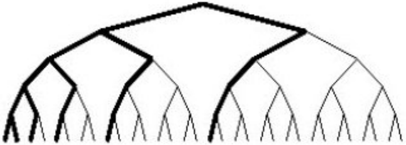
- Key idea: important decisions are at the top of the tree
- Credit = backtracking steps
- Credit distribution: one half at the best child the other divided among the other children.
- When credits run out follow deterministic best-search
- In addition: allow limited backtracking steps (eg, 5) at the bottom
- Control parameters: initial credit, distribution of credit among the children, amount of local backtracking at bottom.



Incomplete Search

Limited Discrepancy Search (LDS)

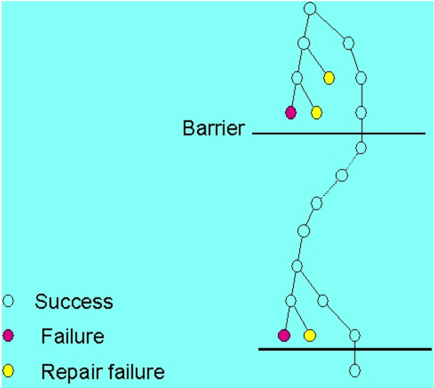
- Key observation that often the heuristic used in the search is nearly always correct with just a few exceptions.
- Explore the tree in increasing number of discrepancies, modifications from the heuristic choice.
- Eg: count one discrepancy if second best is chosen
  count two discrepancies either if third best is chosen or twice the second best is chosen
- Control parameter: the number of discrepancies



Incomplete Search

Barrier Search

- Extension of LDS
- Key idea: we may encounter several, independent problems in our heuristic choice. Each of these problems can be overcome locally with a limited amount of backtracking.
- At each barrier start LDS-based backtracking



Local Search for CSP [Hoos and Tsang, 2006]

- Uses a complete-state formulation
- Initial state: a value assigned to each variable (randomly)
- Changes the value of one variable at a time
- Evaluation of a state:
  number of constraints violated or variables to change (see soft constraints)
- Min-conflict heuristic [Minton et al., 1992]:
  - pick one variable involved in a constraint violation at random
  - assign to it the best value
- Run-time independent from problem size

References

Hoos H.H. and Tsang E. (2006). **Local Search Methods**, chap. 5. Elsevier.

Minton S., Johnston M., Philips A., and Laird P. (1992). **Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems**. *Artificial Intelligence*, 58(1-3), pp. 161–205.

Rossi F., van Beek P., and Walsh T. (eds.) (2006). **Handbook of Constraint Programming**. Elsevier.

Schulte C. and Carlsson M. (2006). **Finite domain constraint programming systems**. In Rossi et al. [2006].