Backtracking Search Algorithms

Sakari Ellonen

General Idea

- Backtracking Search Algorithms are complete search methods: given sufficient time and space, they are guaranteed to find a solution if one exists.
- The search proceeds by building up a solution little by little, and when a deadend is discovered, the algorithm *backtracks* to a point where the deadend could have been avoided.

Overview of the Presentation

- Branching strategies: what the branches of the search are like.
- Constraint Propagation: how to reduce the search space *as the search progresses*.
- Nogoods: how to characterize deadends.
- Backjumping: how to backtrack to a point where all was not lost.

Preliminaries

- A constraint satisfaction problem (CSP) consists of a set of variables $X = \{x_1, \ldots, x_n\}$; a set of values $D = \{a_1, \ldots, a_d\}$, where each variable $x_i \in X$ has an associated finite domain $dom(x_i) \subseteq D$ of possible values; and a collection of constraints.
- A constraint C is a relation — a set of tuples — over some set of variables, denoted $vars(C)$.
- A solution to a CSP is an assignment of a value to each variable that satisfies all the constraints.

6-queens running example

- A variable for each column of the board $\{x_1, \ldots, x_6\}$, each with domain $dom(x_i) = \{1, \ldots, 6\}$.
- Assigning a value j to a variable x_i means placing a queen in row j, column i.
- For $1 \leq i < j \leq 6$ there is a constraint C_{ij}, given by $(x_i \neq x_j) \wedge (|i - j| \neq |x_i - x_j|)$.

Search Trees

- A (chronological) backtracking search resembles depth-first traversal of a search tree, which is generated as the search progresses.
- The method of extending a node of the tree is called *branching strategy*.
- A node in the search tree is a *deadend* if it does not lead to a solution.
- (whiteboard example of the naive backtracking algorithm for the 6-queens problem)

Branching Strategies

- In the naive BT, a node $p = \{x_1 = a_1, \ldots, x_j = a_j\}$ was extended by selecting an uninstantiated variable x and adding a branch to a new node $p \cup \{x = a\}$ for each $a \in dom(x)$.
- More generally, a node $p = \{b_1, \ldots, b_j\}$ is a set of *branching constraints*, where $b_i, i \leq j$ is the branching constraint *posted* at level i. A node p is extended by adding the branches to nodes $p \cup \{b_{j+1}^1\}, \ldots, p \cup \{b_{j+1}^k\}$. The branching constraints have to be mutually exclusive and exhaustive.
- This presentation considers only unary branching constraints.

Branching Strategies (continued)

The three most common branching strategies are:

- *Enumeration* or *d-way branching*: A branch is generated for each value in the domain of an uninstantiated variable x. (see the naive BT)

Branching Strategies (continued)

The three most common branching strategies are:

- *Enumeration* or *d-way branching*: A branch is generated for each value in the domain of an uninstantiated variable x. (see the naive BT)
- *Binary Choice Points* or *2-way branching*: Two branches are generated. Given an uninstantiated x and $a \in dom(x)$, constraints $x = a$ and $x \neq a$ are posted along the branches.

Branching Strategies (continued)

The three most common branching strategies are:

- *Enumeration* or *d-way branching*: A branch is generated for each value in the domain of an uninstantiated variable x. (see the naive BT)
- *Binary Choice Points* or *2-way branching*: Two branches are generated. Given an uninstantiated x and $a \in dom(x)$, constraints $x = a$ and $x \neq a$ are posted along the branches.
- *Domain splitting*: Given an uninstantiated x, the domain of x is split into sub-domains. For example, if $dom(x) = \{1, 2, 3, 4\}$, constraints $x \leq 2$ and $x > 2$ could be posted.

Constraint Propagation in BT

Motivation and intuition:

- As we have learnt in the previous presentations, CP can dramatically reduce the search space of a CSP.
- In the worst case (when no solution exists), backtracking algorithms have to go through the entire search space, so such reduction would be welcome.
- Each time a branching constraint is posted, applying constraint propagation may reduce the domains of still uninstantiated variables, which simplifies the subproblem. Furthermore, if some domain becomes empty, we have come to a deadend and may backtrack.

Maintaining Arc Consistency

- Arc Consistency: Given a constraint C, a value $a \in dom(x)$ for a variable $x \in vars(C)$ is said to have a *support* in C if there exists a tuple $t \in C$ such that $a = t[x]$ and $t[y] \in dom(y)$ for every $y \in vars(C)$. A constraint C is said to be *arc consistent* if for each $x \in vars(C)$, each value $a \in dom(x)$ has a support in C.

Maintaining Arc Consistency

- Arc Consistency: Given a constraint C, a value $a \in dom(x)$ for a variable $x \in vars(C)$ is said to have a *support* in C if there exists a tuple $t \in C$ such that $a = t[x]$ and $t[y] \in dom(y)$ for every $y \in vars(C)$. A constraint C is said to be *arc consistent* if for each $x \in vars(C)$, each value $a \in dom(x)$ has a support in C.
- MAC algorithm maintains arc consistency on constraints with at least one uninstantiated variable.

Maintaining Arc Consistency

- Arc Consistency: Given a constraint C, a value $a \in dom(x)$ for a variable $x \in vars(C)$ is said to have a *support* in C if there exists a tuple $t \in C$ such that $a = t[x]$ and $t[y] \in dom(y)$ for every $y \in vars(C)$. A constraint C is said to be *arc consistent* if for each $x \in vars(C)$, each value $a \in dom(x)$ has a support in C.
- MAC algorithm maintains arc consistency on constraints with at least one uninstantiated variable.
- (whiteboard example for the 6-queens problem)

The Cost of Constraint Propagation

- While CP may simplify the subproblem, it adds the extra cost of running the CP algorithm.
- One way to reduce the extra cost is to only apply CP to certain constraints
  - Forward Checking (FC) maintains arc consistency only on constraints with one uninstantiated variable (whiteboard example).

Nogoods

- A Nogood is a set of branching constraints that is not consistent with any solution.
- For example, in the 6-queens problem, $\{x_1 = 2, x_2 = 5\}$ is a nogood, because it is not contained in any solution.
  - After this has been discovered somehow, a new constraint $(x_1 \neq 2 \vee x_2 \neq 5)$ can be added.
  - $(x_1 \neq 2 \vee x_2 \neq 5)$ is called an *implied constraint*, because it does not affect the set of solutions to the CSP.

Discovering Nogoods

- A nogood is only useful if it simplifies the search in the future: thus recording nogoods that correspond to nodes already fully expanded, like in the previous example, makes little sense.

Discovering Nogoods

- A nogood is only useful if it simplifies the search in the future: thus recording nogoods that correspond to nodes already fully expanded, like in the previous example, makes little sense.
- *Jumpback nogood*: Let $p = \{b_1, \ldots, b_j\}$ be a deadend node in the search tree, where b_i is the branching constraint posted at level i in the search tree. In the case where no constraint propagation is performed, jumpback nogood for p, $J(p)$, is defined recursively as follows:

Jumpback definition

1. p is a leaf node. Let C be a constraint that is not consistent with p:

$$J(p) = \{b_i \mid vars(b_i) \cap vars(C) \neq \emptyset, 1 \leq i \leq j\}.$$

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

$$J(p) = \bigcup_{i=1}^{k} (J(p \cup \{b^i_{j+1}\}) \setminus \{b^i_{j+1}\}).$$

Jumpback example

Consider node $p = \{x_1 = 2, x_2 = 5, x_3 = 3, x_4 = 1, x_5 = 4\}$ in the 6-queens search tree. All attempts to expand the node on x_6 will fail, so we get the following:

$$
\begin{aligned}
J(p) &= \bigcup_{k=1}^{6} (J(p \cup \{x_6 = k\}) \setminus \{x_6 = k\}) \\
&= \{x_2 = 5\} \cup \cdots \cup \{x_3 = 3\} \\
&= \{x_1 = 2, x_2 = 5, x_3 = 3, x_5 = 4\}
\end{aligned}
$$

Notice that it matters which constraint is chosen at the leaf nodes! Also, there is no x_4 in $J(p)$!

Eliminating explanation

- An eliminating explanation is a contraint that is sufficient to account for the removal of a value from a domain.
- *Eliminating explanation*: Let $p = \{b_1, \ldots, b_j\}$ be a node in the search tree and $a \in dom(x)$ be a value that is removed from the domain of a variale x by constraint propagation at node p. Eliminating explanation expl$(x \neq a)$ is a subset of p such that expl$(x \neq a) \cup \{x = a\}$ is a nogood.
- In the 6-queens, when $p = \{x_1 = 2, x_2 = 5\}$, expl$(x_6 \neq 1) = \{x_2 = 5\}$.

Jumpbacks with CP

Eliminating explanations allow us to generalize the definition of jumpback nogoods to deadends found by constraint propagation:

1. p is a leaf node.

$$J(p) = \bigcup_{a \in dom(x)} \text{expl}(x \neq a)$$

2. p is not a leaf node. Same as before.

Non-Chronological Backtracking

- Upon discovering a deadend, the standard backtracking algorithms retract the most recently posted constraint.
- If the most recent constraint does not bear responsibility for the deadend, it is more efficient to retract further back in history.
- Let $J(p) \subseteq p$ be a nogood at p. Backjump point is the largest i such that $b_i \in J(p)$. (whiteboard example of $p = \{x_1 = 2, x_2 = 5, x_3 = 3, x_4 = 1\}$)

Partial Order Backtracking

- Partial order backtracking is an even more “relaxed” way of backtracking: it does not care about the order in which branching constraints have been posted, but creates an order of its own based on discovered nogoods.
- Notice that a nogood $((x_1 = a_1), \ldots, (x_j = a_j))$ can be written as
  $((x_1 = a_1) \wedge \cdots \wedge (x_{j-1} = a_{j-1})) \Rightarrow (x_j \neq a_j)$.
- Now such a nogood induces order
  $(x_j = a_j) \geq (x_1 = a_1), \ldots, (x_j = a_j) \geq (x_{j-1} = a_{j-1})$.

Partial Order Continued

- When backtracking, a maximal branching constraint should be chosen. If not possible and a non-maximal $(x = a)$ has to be chosen, then the ordering relation has to be relaxed wherever $(x = a)$ appears.
- It is not clear to me how to choose a branching constraint of a nogood to be the right side of the equation. I guess it is simply arbitrary.
- If time is not up, an example of a method to compute eliminating explanations.