Search for Indexed Algebraic Dynamic Programmning

Christopher Bacher

bacher@ac.tuwien.ac.at

Algorithms and Complexity Group
TU Wien
March 27^{th}, 2017

Algebraic Dynamic Programming

Algebraic Dynamic Programming (ADP)

Alternative view on Dynamic Programmning (Giegerich et al., 2002)

- Formal grammar defines the search space by **decomposition**
- Separates evaluation from search space declaration
- Separates search from search space declaration
- Works for sequence data (strings)—originally intended for bioinformatics
- Extension for set/general data structures available

Pros/Cons of ADP

Benefits

- Formal framework for expressing (some) DP algorithms
- Specification of modelling languages
- Solver software

Drawbacks

- Frameworks quite restricted
- Tailored for specific domains
- No support for real-valued indices

Example: Knapsack

Given set of items $i \in \mathcal{I}$ and knapsack of max. weight Q

- S . . . Optimally packed knapsack
- $B_{i,q}$ . . . Knapsack of weight at most q with item i considered last
  - i . . . integer
  - q . . . real-valued

$$
\begin{aligned}
S &\to \pi_i B_{i,Q-w_i} && \forall i \in \mathcal{I}, [Q - w_j \geqslant 0] \\
B_{i,q} &\to \pi_j B_{j,q-w_j} && \forall j \in \mathcal{I}, [i < j][q - w_j \geqslant 0] \\
&\mid \epsilon
\end{aligned}
$$

Example: Knapsack

Given set of items $i \in \mathcal{I}$ and knapsack of max. weight Q

- S ... Optimally packed knapsack
- $B_{i,q}$ ... Knapsack of weight at most q with item i considered last
  - i ... integer
  - q ... real-valued

$$S \to \pi_i B_{i,Q-w_i} \qquad \forall i \in \mathcal{I}, [Q - w_j \geqslant 0]$$
$$B_{i,q} \to \pi_j B_{j,q-w_j} \qquad \forall j \in \mathcal{I}, [i < j][q - w_j \geqslant 0]$$
$$\mid \epsilon$$

Explicit compatibilities

$$B_{i,q} \to B_{i,q'} \qquad \forall q' \in \mathbb{R} : w_i \leqslant q' < q$$

Example: Shortest Path

Given a graph $G = (V, A)$

- $S_{s,t}$ ... Shortest path from s to t
- $P_{s,X,t}$ ... Path from s to t with unvisited nodes X
  - s, t ... integer
  - X ... set

$$
\begin{aligned}
S_{s,t} &\rightarrow P_{s,V\setminus\{s,t\},t} \\
P_{s,X,t} &\rightarrow a_{s,x} P_{x,X-x,t} && \forall x \in X, [(s,x) \in A] \\
&\mid a_{s,t} && [(s,t) \in A]
\end{aligned}
$$

Example: Shortest Path

Given a graph $G = (V, A)$

- $S_{s,t}$ ... Shortest path from s to t
- $P_{s,X,t}$ ... Path from s to t with unvisited nodes X
  - s, t ... integer
  - X ... set

$$
\begin{aligned}
S_{s,t} &\rightarrow P_{s,V\setminus\{s,t\},t} \\
P_{s,X,t} &\rightarrow a_{s,x} P_{x,X-x,t} && \forall x \in X, [(s,x) \in A] \\
&\mid a_{s,t} && [(s,t) \in A]
\end{aligned}
$$

Shortest path is not expressibly without set semantics!

Indexed Algebraic Dynamic Programming

Example: Knapsack

Given set of items $i \in \mathcal{I}$ and knapsack of max. weight Q

- S . . . Optimally packed knapsack
- $B_{i,q}$ . . . Knapsack of weight at most q with item i considered last
  - i . . . integer
  - q . . . real-valued

$$
\begin{aligned}
S &\rightarrow \pi_i B_{i,Q-w_i} && \forall i \in \mathcal{I}, [Q - w_j \geqslant 0] \\
B_{i,q} &\rightarrow \pi_j B_{j,q-w_j} && \forall j \in \mathcal{I}, [i < j][q - w_j \geqslant 0] \\
&\mid \epsilon
\end{aligned}
$$

Example: Knapsack

Given set of items $i \in \mathcal{I}$ and knapsack of max. weight Q

- S . . . Optimally packed knapsack
- $B_{i,q}$ . . . Knapsack of weight at most q with item i considered last
  - i . . . integer
  - q . . . real-valued

$$S \to \pi_i B_{i,Q-w_i} \qquad \forall i \in \mathcal{I}, [Q - w_j \geqslant 0]$$
$$B_{i,q} \to \pi_j B_{j,q-w_j} \qquad \forall j \in \mathcal{I}, [i < j][q - w_j \geqslant 0]$$
$$\mid \epsilon$$

Explicit compatibilities

$$B_{i,q} \to B_{i,q'} \qquad \forall q' \in \mathbb{R} : w_i \leqslant q' < q$$

Example: Shortest Path

Given a graph $G = (V, A)$

- $S_{s,t}$ ... Shortest path from s to t
- $P_{s,X,t}$ ... Path from s to t with unvisited nodes X
  - s, t ... integer
  - X ... set

$$
\begin{aligned}
S_{s,t} &\rightarrow P_{s,V\setminus\{s,t\},t} \\
P_{s,X,t} &\rightarrow a_{s,x} P_{x,X-x,t} && \forall x \in X, [(s,x) \in A] \\
&\mid a_{s,t} && [(s,t) \in A]
\end{aligned}
$$

Example: Shortest Path

Given a graph $G = (V, A)$

- $S_{s,t}$ ... Shortest path from s to t
- $P_{s,X,t}$ ... Path from s to t with unvisited nodes X
  - s, t ... integer
  - X ... set

$$
\begin{aligned}
S_{s,t} &\rightarrow P_{s,V\setminus\{s,t\},t} \\
P_{s,X,t} &\rightarrow a_{s,x} P_{x,X-x,t} && \forall x \in X, [(s,x) \in A] \\
&\mid a_{s,t} && [(s,t) \in A]
\end{aligned}
$$

Shortest path is not expressibly without set semantics!

Formal Definition

- Terminals $\mathcal{A}$ . . . atomic elements of the search space
- Non-Terminals $\mathcal{N}$ . . . compositions of atomic elements
- Top-Down Productions **(Decomposition)**

$$(X_{\mathcal{I}}\ [p_j^{(lhs)}] \rightarrow \gamma_i\ [p_i^{(rhs)}], X \in \mathcal{N}, \gamma_i \in (\mathcal{A} \cup \mathcal{N})^*$$

- Bottom-Up Productions **(Composition)**

$$\{\alpha_1, \ldots, \alpha_k, \ldots, \alpha_K\} \dashrightarrow \hat{X}_{\mathcal{I}_X} \quad [p_i]\ \langle c_i \rangle, \alpha_k \in (\mathcal{A} \cup \mathcal{N})$$

  - p_i . . . filter predicate
  - c_i . . . propagating constraint

$$x_{jk} = v(\mathcal{I}_1, \ldots, \mathcal{I}_j \backslash x_{jk}, \ldots, \mathcal{I}_K)$$

Search Space, Derivation Trees, and Candidates

Evaluation Algebras

- Used to compute objectives for solution candidates
- Assign values to complete subtrees
- Can only be evaluated bottom-up
- Only evaluated once and stored in the according DP table cell

$$\sigma(a_{\mathcal{I}}) = f_a(\mathcal{I}), \qquad a_{\mathcal{I}} \in \hat{\mathcal{A}}$$

$$\sigma(A_{\mathcal{I}}) = \bigcup_{(A^{(i)}_{\mathcal{I}[p_i]} \to \gamma_i) \in \mathcal{P}(A_{\mathcal{I}})} f_{A^{(i)}}\Big[\bigtimes_{X^{(j)}_{\mathcal{V}_j} \in \gamma_i} \sigma\left(X^{(j)}_{\mathcal{V}_j(\mathcal{I})}\right)\Big], \qquad A \in \mathcal{N}$$

Cost Algebras

- Assign costs to deriving a symbol from the axiom S

State Compatibility & Dominance

- Original ADP uses choice functions for state selection e.g. min, max

Instead define . . .

- States: Each indexed symbol represents a state
- Compatibility: Some states represent compatible objects
- Dominance: If state A dominates a compatible state B then A is always preferred to B

State Compatibility

State A is compatible with state B, $A \sqsubseteq B$, given an evaluation algebra σ iff . . .

- it exists a (virtual) production $\hat{B}^{(\text{ex.})}_{\mathcal{I}_B} \rightarrow \gamma_1 \hat{A}_{\mathcal{I}_A} \gamma_2$
- and an extension σ_e to σ
- s.t. $\sigma_e(\hat{B}^{(\text{ex.})}_{\mathcal{I}_B}) = f_{B^{(\text{ex.})}}[\sigma_e(\gamma_1), \sigma_e(\hat{A}_{\mathcal{I}_A}), \sigma_e(\gamma_2)] \subseteq \sigma(\hat{B}_{\mathcal{I}_B})$

State Compatibility

State A is compatible with state B, $A \sqsubseteq B$, given an evaluation algebra σ iff ...

- it exists a (virtual) production $\hat{B}^{(\text{ex.})}_{\mathcal{I}_B} \rightarrow \gamma_1 \hat{A}_{\mathcal{I}_A} \gamma_2$
- and an extension σ_e to σ
- s.t. $\sigma_e(\hat{B}^{(\text{ex.})}_{\mathcal{I}_B}) = f_{B^{(\text{ex.})}}[\sigma_e(\gamma_1), \sigma_e(\hat{A}_{\mathcal{I}_A}), \sigma_e(\gamma_2)] \subseteq \sigma(\hat{B}_{\mathcal{I}_B})$

Properties

- **Reflexivity** ... $A \sqsubseteq A$
- **Transitivity** ... $A \sqsubseteq B$ and $B \sqsubseteq C$ implies $A \sqsubseteq C$
- **Simple** ... $A \sqsubseteq_S B$ iff $\hat{B}^{(\text{ex.})}_{\mathcal{I}_B} \rightarrow \hat{A}_{\mathcal{I}_A}$
- **Implicit** ... productions present in the grammar
- **Explicit** ... productions not present in the grammar

State Dominance

Given two states A and B, $A \sqsubseteq B$

$$p(\mathcal{V}_A(\mathcal{I}_A), \mathcal{V}_B(\mathcal{I}_B), \sigma(\hat{A}_{\mathcal{I}_A}), \sigma(\hat{B}_{\mathcal{I}_B}), \mathcal{C}) = \top$$

Evaluating Indexed Grammars

Top-Down Evaluation

Two datastructures ...

- $\mathcal{Q}$ ... *Pending* queue of uncomputed non-terminals
- $\mathcal{H}$ ... *Hold* set of non-terminals with open dependencies

1. $\mathcal{Q} = \{\text{axiom } S\}$; $\mathcal{H} = \emptyset$
2. Until Q is empty
   1. Remove symbol A from Q
   2. If A is not computed and has not failed
      For all productions $A \rightarrow \gamma$
      1. Check constraints $[p^{(}lhs)]$ and $[p^{(}rhs)]$
      2. Generate all $X \in \gamma \cap \mathcal{N}$ s.t. X is uncomputed
      3. Add generated X to Q
      4. Mark all rules continaing $X \in \gamma$ s.t. X is marked as failed, as failed
      5. If all rules failed $\Rightarrow$ mark X as failed; recursive cleanup of Q and H
      6. If some X is added to Q add A to A
      7. If no X is added to $Q \Rightarrow$ evaluate X; update table cell
         and add all $Y \in H$ to Q where Y depends on X and all dependencies are satisfied
3. If $\mathcal{H}$ is not empty $\Rightarrow$ fail due to cycle in grammar

Bottom-Up Evaluation

Two datastructures . . .

- $\mathcal{Q}$ . . . *Trigger* set of computed symbols

1. Initialize $\mathcal{Q}$ with all valid terminals t_i
2. Until Q is empty
   1. Remove symbol A from Q
   2. For all productions $\{\gamma\} \dashrightarrow X$, $A \in gamma$
      1. Propagate constraints $\langle c \rangle$
      2. For each assignment in the cross-product of the filtered index domains
         1. If for some element in the indexed γ no value is known continue
         2. Check constraints $[p]$
         3. Evaluate the rule
         4. Update the table cell
         5. If the cell content changed add X to $\mathcal{Q}$

Bi-Directional Evaluation

Simultaneously use ...

- Top-Down Evaluation
- Bottom-Up Evaluation
- May avoid unnecessary productions

Generates two fronts, where each front ...

- is a set of unexpanded symbols $A \in \mathcal{F}$
- is derived from a common origin $\mathcal{O}$
- forms a sequence $\mathcal{F}^{(0)} = \mathcal{O}, \mathcal{F}^{(1)}, \ldots, \mathcal{F}^{(k)}$

Tries to join the fronts ...

- Each element in one front needs at least one *join partner* in the other
- Two elements are join partners iff:
  - There exists a sequence of productions deriving/constructing one from the other
  - Search engine needs hints to decide the existance in form of virtual productions
  - Exact conditions not 100% formalized

Thanks for your attention