Extensions for Algebraic Dynamic Programming

Christopher Bacher

bacher@ac.tuwien.ac.at

Algorithms and Complexity Group
TU Wien
November 7^{th}, 2016

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: Edit Distance/Min-Cost Alignment Problem

Problem

Given are two strings $X = x_1 x_2 \dots x_n$, $Y = y_1 y_2 \dots y_n$, the matching operators `match` $m(x_i, y_j)$, `replace` $r(x_i, y_j)$, `delete` $d(x_i,)$ *from X,* `insert` $a(, y_j)$ *in X and a cost assignment to each operator. What is the minimum cost alignment of X and Y*

Example from [ADP Lecture, Universitt Freiburg] (Mhl et al., 2011)

Example: Edit Distance/Min-Cost Alignment Problem

Problem

Given are two strings $X = x_1 x_2 \dots x_n$, $Y = y_1 y_2 \dots y_n$, the matching operators `match` $m(x_i, y_j)$, `replace` $r(x_i, y_j)$, `delete` $d(x_i,)$ *from X,* `insert` $a(, y_j)$ *in X and a cost assignment to each operator. What is the minimum cost alignment of X and Y*

Example from [ADP Lecture, Universitt Freiburg] (Mhl et al., 2011)

Productions of the RTG for the EDP

$$M \to xMy|xD|Ay|\epsilon$$
$$D \to M|xD|Ay|\epsilon$$
$$A \to M|xD|Ay|\epsilon$$

Σ-Evaluation Algebras

- Specifies the **evaluation phase**
- Requires a candidate choice function $h(.)$

For our example (adapted notation):

Evaluation of the productions

$$M = m(x, y) + M|d(x) + D|A + a(y)|0$$
$$D = M|d(x) + D|A + a(y)|0$$
$$A = M|d(x) + D|A + a(y)|0$$

Candidate choice/objective function: $h(M) = h(D) = h(A) = \min$

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

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

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

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

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!

Example: Resource Constraint Shortest Path

Given graph $G = (V, A)$ and k resource capacities $Q^{(k)}$

$$S_{s,t} \to P_{s,V\setminus\{s,t\},t,Q^{(k)}}$$

$$P_{s,X,t,q^{(k)}} \to a_{s,x} P_{x,X-x,t,q^{(k)}-r^{(k)}_{s,x}} \quad \forall x \in X, [(s,x) \in A][\forall k : q^{(k)} - r^{(k)}_{s,x} \geqslant 0]$$

$$\mid a_{s,t} \quad [(s,t) \in A][\forall k : q^{(k)} - r^{(k)}_{s,t} \geqslant 0]$$

Example: Traveling Salesman Problem

Given a graph $G = (V, A)$ visit all vertices in V exactly once Formalization of the Bellman-Held-Karp algorithm

$$
\begin{aligned}
S &\rightarrow a_{1,i} P_{i,V\setminus\{1,i,j\},j} a_{j,1} && \forall i, j \in V, [1 \neq i \neq j][(1,i) \in A][(j,1) \in A] \\
P_{i,X,j} &\rightarrow a_{i,x} P_{x,X-x,j} && \forall x \in X, [(i,x) \in A] \\
&\mid a_{i,j} && [X = \varnothing][(i,j) \in A]
\end{aligned}
$$

Extensions to ADP

Formal Definition

- Terminals $\mathcal{A}$ . . . atomic elements of the search space
- Non-Terminals $\mathcal{N}$ . . . compositions of atomic elements
- Productions $(X_{\mathcal{I}} \rightarrow \gamma_i[p_i],\ X \in \mathcal{N},\ \gamma_i \in (\mathcal{A} \cup \mathcal{N})$

Σ-evaluation algebra

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

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

Explicit Indexed Grammars

Former frameworks ...

- Original ADP framework (Giegerich, 2002)
- Generalized ADP framework (Siederdissen, 2014/15)

hide indexing of the grammar

- Major advantage: no index errors
  - Originally possible due to restricted domain
  - Possible in GADP as decomposition semantics are specified each time anew
- Major disadvantage: Hard (if not impossible) to formulate certain problems

Whistle(r) uses explicit indexed grammars instead

Index Propagators

- Generalize the concept of index hiding/inference
- Propagators bind an index to an input
- Consume the input in a specific way

Examples for propagators

- **Sequence** . . . Consume next element from sequence or nothing
- **Cyclic Sequence** . . . Next element with arbitrary starting point
- **Resource** . . . Consume resource based on value of another index
- **Set-inclusion** . . . Move values between a **used**/**unused** set

Conditional & Constraint Productions

Conditionals

- Predicate on indices of the LHS
- RHS only applicable if predicate holds
- Can filter multiple RHS at once

Index Constraints

- Predicate on indices of the RHS
- Filter a single production
- Decomposition needs to be applied before checking

Target Value Constraints

- Predicate on indices of the RHS and the target values
- Filter a single production
- Decomposition needs to be applied before checking

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, $A \sqsubseteq B$

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

Index Relaxation & Amalgamation (I)

Remember the recursion for Shortest Path ...

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

and TSP ...

$$\begin{aligned} P_{i,X,j} &\rightarrow a_{i,x} P_{x,X-x,j} && \forall x \in X, [(i,x) \in A] \\ &\mid a_{i,j} && [X = \varnothing][(i,j) \in A] \end{aligned}$$

... both formulations describe an exponential number of **d**ifferent states.

Index Relaxation & Amalgamation (I)

Remember the recursion for Shortest Path ...

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

and TSP ...

$$\begin{aligned} P_{i,X,j} &\rightarrow a_{i,x} P_{x,X-x,j} && \forall x \in X, [(i,x) \in A] \\ &\mid a_{i,j} && [X = \varnothing][(i,j) \in A] \end{aligned}$$

... both formulations describe an exponential number of **d**ifferent states.

Polynomial time algorithms for Shortest Path exist. Why?

Index Relaxation & Amalgamation (II)

- **Relaxation** . . . Sometimes not all indices need to be stored.
- **Amalgamation** . . . Sometimes not all information carried by an index needs to be stored.

- A relaxed index is not stored but present during evaluation of RHS
- Indexed symbols distinguished only by relaxed indices map to one DP table cell

Index Relaxation & Amalgamation (II)

- **Relaxation** . . . Sometimes not all indices need to be stored.
- **Amalgamation** . . . Sometimes not all information carried by an index needs to be stored.

- A relaxed index is not stored but present during evaluation of RHS
- Indexed symbols distinguished only by relaxed indices map to one DP table cell

When can an index be relaxed/amalgamated?

Index Relaxation & Amalgamation (III)

Definition (Non-dominating Index Relaxation)

An index relaxation for $i \in \mathcal{I}_X$ of a non-terminal X is non-dominating iff ...

- any non-dominated target value in the non-relaxed program
- is not dominated by any target value in the relaxed program

Definition (Amalgable Index)

An index is amalgable if its values can be ordered by their degree of freedom.

Index Relaxation & Amalgamation (III)

Definition (Non-dominating Index Relaxation)

An index relaxation for $i \in \mathcal{I}_X$ of a non-terminal X is non-dominating iff ...

- any non-dominated target value in the non-relaxed program
- is not dominated by any target value in the relaxed program

Definition (Amalgable Index)

An index is amalgable if its values can be ordered by their degree of freedom.

A symbol needs to be computed iff no symbol has been computed ...

- with the same non-relaxed index values
- with amalgamated index value of higher degree of freedom

has been computed

Index Amalgamation

Index i of symbol X can be ordered by its degree of freedom iff ...

- Totally ordered indices (integer, float) any $u, v \in \mathcal{D}(i)$
  - either $u \leqslant v \implies X_{\ldots,u,\ldots} \sqsubseteq_S X_{\ldots,v,\ldots}$
  - or $u \geqslant v \implies X_{\ldots,u,\ldots} \sqsubseteq_S X_{\ldots,v,\ldots}$ holds
- Half-ordered indices (set-valued) any $A, B \in \mathcal{D}(i)$
  - either $A \subseteq B \implies X_{\ldots,A,\ldots} \sqsubseteq_S X_{\ldots,B,\ldots}$
  - or $A \supseteq B \implies X_{\ldots,A,\ldots} \sqsubseteq_S X_{\ldots,B,\ldots}$ holds

Index Amalgamation for Shortest Path and TSP

Shortest Path dropping the set index is ...

- **Amalgable** ... Fewer visited vertices $\rightarrow$ higher degree of freedom
- **Non-dominating** ... Visiting more vertices always has higher costs (non-negative cycles)

Index Amalgamation for Shortest Path and TSP

Shortest Path dropping the set index is ...

- **Amalgable** ... Fewer visited vertices → higher degree of freedom
- **Non-dominating** ... Visiting more vertices always has higher costs (non-negative cycles)

TSP dropping the set index is ...

- **Amalgable** ... Fewer visited vertices → higher degree of freedom
- **Non-dominating** ... Reusing a table cell may lead to unvisted vertices!

Conclusion

To conclude . . .

- ADP Provides a formal language for DP
- Explicitly Indexed Grammars can make modelling more expressive
- Index Propagators return the benefit of implicit indexing
- Set Indices and Index Relaxation allow for efficient algorithms not possible in Generalized ADP