Deriving Dynamic Programming via Thinning and Incrementalization

Akimasa Morihata (Univ. Tokyo)
Masato Koishi (Toshiba Solutions Corp.)
Atsushi Ohori (RIEC, Tohoku Univ.)

FLOPS 2014, Kanazawa, Japan
0-1 Knapsack Problem

Given: budget $B$ and items $i_1, \ldots, i_n$

Objective: most valuable itemset that can buy

$B = ¥300$

- 0.5MJ, ¥120
- 0.2MJ, ¥60
- 1.3MJ, ¥90
- 1.6MJ, ¥150
- 1.9MJ, ¥140
1. **Recurrence equation** *(principle of optimality):*

\[
\text{opt}(S \uplus \{i\}, c) = \max \{\text{opt}(S, c), \text{value}(i) + \text{opt}(S, c - \text{cost}(i))\}
\]

2. **Table-filling implementation:**

<table>
<thead>
<tr>
<th></th>
<th>(c = 0)</th>
<th>(c = 1)</th>
<th>...</th>
<th>(c = \text{cost}(i_1) + 1)</th>
<th>...</th>
<th>(c = B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>({i_0})</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>({i_0, i_1})</td>
<td></td>
<td></td>
<td>+ \text{value}(i_1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Textbook Approach to DP

- Problem Description
- Recurrence Equation
- Thinning
- Incrementalization
- Table-filling Implementation
Our Contribution

New method of systematically developing DP by combining known methods:

– Thinning (Bird & de Moor ’96; Morihata ’11) to expose a recurrence equation

– Incrementalization (Liu+ ’03; ’05) to develop efficient table-filling computation

This is (to our knowledge) a first study on their cooperation
Problem Description

\[ \text{knapsack} = \max_{\leq \text{value}} \circ \text{feasible} \circ \text{subsets} \]

\[ \text{subsets} \ [\] = \{[\] \} \]

\[ \text{subsets} (i : \text{is}) = \text{let } r = \text{subsets is} \ \text{in} \ r \cup \text{map } (i:) \ r \]

\[ \text{feasible} = \text{filter } (\lambda \text{is. cost is} \leq B) \]

\[ \text{value} = \text{sum} \circ \text{map valueOfItem} \]

\[ \text{cost} = \text{sum} \circ \text{map costOfItem} \]
Thinning by Shortcut Fusion

Thinning (Bird & de Moor ’97): avoid exhaustive enumeration by fusing maximizer/filter/enumerator

**Theorem.** (refinement of Morihata’11)

\[
\max_\leq (\text{filter } p \ (\text{gen } (\cup) \ (\lambda a. \text{map } (a:)) \ ([[]]))) \\
= \max_\leq (\text{gen } (\lambda x y. \ max_\leq_\vartriangleup (x \cup y)) \\
\ (\lambda a. \ max_\leq_\vartriangleup \circ \text{filter } p \circ \text{map } (a:)) \\
\ (\text{filter } p \ ([[]])))
\]

if \( p x = h x \ll c, \)
\( \leq \) and \( \vartriangleright \) are monotone on \( (:) \), \( \vartriangleright \) is increasing on \( (:) \),
and \( \text{gen} \)'s type is \( \forall \beta. \ (\beta \rightarrow \beta \rightarrow \beta) \rightarrow (A \rightarrow \beta \rightarrow \beta) \rightarrow \beta \rightarrow \beta. \)
Applying Thinning to \textit{knapsack}

\[
\text{knapsack} = \max_{\leq \text{value}} \circ \text{feasible} \circ \text{subsets}
\]

\[
\text{subsets} \ [i] = \{[i]\}
\]

\[
\text{subsets} \ (i : \text{is}) = \text{let } r = \text{subsets} \ is \ \text{in } r \cup \text{map} \ (i: : ) \ r
\]

\[
\text{Thinning}
\]

\[
\text{knapsack} = \max_{\leq \text{value}} \circ \text{subsets'}
\]

\[
\text{subsets'} \ [i] = \{[i]\}
\]

\[
\text{subsets'} \ (i : \text{is}) = \text{let } r = \text{subsets'} \ is \ \text{in } \max_{\leq \text{value} \cap \geq \text{cost}} (r \cup \text{feasible} \ (\text{map} \ (i: : ) \ r))
\]
Note: Recurrence Equation Exposed

$max_{\leq \text{value} \geq \text{cost}}$ removes an element if a more valuable and cheaper one exists, namely, ⋆

$max_{\leq \text{value} \geq \text{cost}} S$

$= \{max_{\leq \text{value}} \{is' | is' \in S, is' \leq_{\text{cost}} is\} | is \in S\}$

This essentially leads to the recurrence equation:

$opt(S \cup \{i\})$

$= \{max_{\leq \text{value}} \{is | is' \in opt(S), is \in \{is', i : is'\},$

$\quad cost(is) \leq c\} | 0 \leq c \leq B\}$

⋆assume no two candidates have the same value
Drawback of Thinning

Obtained program is not very efficient

\[ \text{knapsack} = \max_{\leq \text{value} \circ \text{subsets'}} \]

\[ \text{subsets'}[\{} = \{ \} \]  

\[ \text{subsets'}(i : \text{is}) = \text{let } r = \text{subsets'}(\text{is}) \]  

\[ \text{in } \max_{\leq \text{value} \cap \geq \text{cost}} (r \cup \text{feasible } (\text{map } (i:) r)) \]  

\[ O(B) \text{ candidates} \]

\[ O(B^2) \text{ time?} \]
Incrementalization

Incrementalization (Liu+ ’03; ’05): improving efficiency by reusing previous results

**Theorem.** (well-known)

If \( s_1 \subseteq \cdots \subseteq s_m \), for associative and commutative \( \oplus \),

\[
\left[ \oplus s_m, \ldots, \oplus s_1 \right] = \text{foldr } f \left[ \oplus s_1 \right] \left[ s_m \setminus s_{m-1}, \ldots, s_2 \setminus s_1 \right]
\]

where \( f s (r:rs) = (\oplus s) \oplus r : r : rs \)
Incrementalize \( \text{max} \)

\[
\max_{\leq \text{value} \land \geq \text{cost}} S = \{ \max_{\leq \text{value}} \{ \text{is}’ \mid \text{is}’ \in S, \text{is}’ \leq_{\text{cost}} \text{is} \} \mid \text{is} \in S \}
\]

\[
\text{let } S = \{ \text{is}_1, \ldots, \text{is}_m \} \text{ s.t. } \text{is}_1 \leq_{\text{cost}} \ldots \leq_{\text{cost}} \text{is}_m
\]

\[
\max_{\leq \text{value} \land \geq \text{cost}} S = \{ \max_{\leq \text{value}} \{ \text{is}_1, \ldots, \text{is}_k \} \mid 1 \leq k \leq m \}
\]

Incrementalization

\[
\max_{\leq \text{value} \land \geq \text{cost}} S = \text{foldr } f [\text{is}_1] [\text{is}_m, \ldots, \text{is}_2]
\]

where \( f \) is \((\text{r:rs}) = \max_{\leq \text{value}} \{ \text{is}, \text{r} \} : \text{r} : \text{rs} \)
Obtained Program

\[ knapsack = \max_{\leq \text{value}} \circ \text{subsets'} \]

\[ \text{subsets'} \[\] = \{[\] \} \]

\[ \text{subsets'} (i : \text{is}) = \text{let } r = \text{subsets'} \text{ is} \]
\[ \quad \text{in } \max_{\leq \text{value} \cong \text{cost}} (r \cup \text{feasible} (\text{map} (i::) r)) \]

- Time complexity: \(O(nB)\)
- It is essentially the one presented by de Moor (1995)
What Written in the Paper

• Our approach scales to other examples (derivations are more complicated, though)
  – longest common sequence problem
  – association-rule mining on two-dimensional numeric data (Fukuda+’99; 01)

• Discussion on possibility of automation
  – Difficulty:
    bridging the gap between two methods
Related Work (1)

• Thinning (Bird & de Moor’ 97)
  – useful to deriving DP from naive specification
  – drawback: results may be too abstract, far from efficient implementation
• Incrementalization (Liu+ ’03; ’05)
  – useful for deriving efficient table-filling implementation from recurrence equations

Our observation: they are orthogonal
Related Work (2)

• de Moor (1995):
  – proposed a law to derive DP for a class of optimization problems
  – Result for the knapsack problem is almost the same
  – We modularized his law into two components: thinning and incrementalization
    ➔ more problems can be solved
Combination of thinning and incrementalization is useful for developing DP!

Future work:

• Automation
  – Domain-specific? Solver-based?
• Combination of other techniques