Does your dynamic programming code output correct values?

Pseudo-Boolean Reasoning About States and Transitions to Certify Dynamic Programming and Decision Diagram Algorithms

Emir Demirović[1], Ciaran McCreesh[2], Matthew J. McIlree[2], Jakob Nordström[34], Andy Oertel[43], Konstantin Sidorov[1]

[1]Delft University of Technology
[2]University of Glasgow
[3]University of Copenhagen
[4]Lund University

Knapsack problem, Algorithms 101

Knapsack problem, dynamic programming

Input data

Item #	Weight	Profit
1	1	2
2	3	4
3	5	7
4	7	10

At most 8

Dynamic programming table

	1	2	3	4	5	6	7	8
1	2	2	2	2	2	2	2	2
2								
3								
4								

Dynamic programming recurrence: $P(k+1, w) = \max(P(k,w), v_k + P(k, w - w_k))$

Off-by-one error! Those should have been v_{k+1} and w_{k+1}

Knapsack problem, dynamic programming

Input data

Item #	Weight	Profit
1	1	2
2	3	4
3	5	7
4	7	10

At most 8

Dynamic programming table

	1	2	3	4	5	6	7	8
1	2	2	2	2	2	2	2	2
2	2	2	4	6	6	6	6	6
3	2	2	4	6	7	9	9	11
4	2	2	4	6	7	9	10	**12**

Dynamic programming recurrence: $P(k+1, w) = \max(P(k, w), v_{k+1} + P(k, w - w_{k+1}))$

And then it gets worse

DP is surprisingly error-prone:

- Implementation errors: off-by-one, dimension mix-ups, etc.
- Incorrect recurrences

Matrix chain multiplication is another example:

$$M(k,k) = 0$$
$$M(j,k) = \min_{\ell}\big(M(j,\ell) + M(\ell+1,k) + p_{j-1}p_k p_\ell\big)$$

Our contribution: embedding the DP transitions in **proof logging**

Introduce new variables representing the DP states

Justify every DP transition with a statement of the form “If the previous states were valid, so is the next one”

Extract the unconditional bound for the state encoding the input problem

Proof logging workflow

Problem instance → Solver → Optimal solution

Solver → Optimality proof

$\color{orange}{2x + 3y} \to \max$
$(1) x + 2y \leq 3$
$(2) 4x + 5y \leq 10$
x, y natural

- $\frac{1}{3} \times (1) + \frac{2}{3} \times (2)$
- $\lfloor (3) \rfloor$
- $\frac{1}{2}(1) + \frac{1}{2}(4)$

Proof logging workflow

Problem instance → Checker → **Yea**/nay

Optimality proof → Checker

$2x + 3y \to \max$

$(1) x + 2y \leq 3$

$(2) 4x + 5y \leq 10$

x, y natural

- $\frac{1}{3}(1) + \frac{2}{3} \times (2)$
- $\lfloor (3) \rfloor$
- $\frac{1}{2}(1) + \frac{1}{2}(4)$

Proof logging workflow

Problem instance → Checker → Yea/**nay**

Optimality proof → Checker

$2x + 3y \to \max$
$(1)\, x + 2y \leq 3$
$(2)\, 4x + 5y \leq 10$
x, y natural

- ~~$\frac{1}{3}(1) + \frac{2}{3} \times (2)$~~
- $\lfloor (3) \rfloor$
- $\frac{1}{2}(1) + \frac{1}{2}(4)$

$\frac{2}{3}(1) + \frac{1}{3} \times (2)$

Let's try fitting it in the DP context

We need to encode the **input problem** and the **proof for the DP table values**

The first part is easy, we re-formulate the knapsack as a **pseudo-Boolean optimization** problem:

$$2x_1 + 4x_2 + 7x_3 + 10x_4 \rightarrow \max \quad \text{Profits}$$

$$\text{Weights} \quad x_1 + 3x_2 + 5x_3 + 7x_4 \leq 8 \quad \text{Capacity}$$

$$x_j \in \{0, 1\}, 1 \leq j \leq 4$$

But how to fit the DP table on the proof framework?

Modeling DP states in the proof

Column: $x_1 + 3x_2 \leq 4$

Entry: $2x_1 + 4x_2 \leq 6$

	1	2	3	4	5	6	7	8
1	2	2	2	2	2	2	2	2
2	2	2	4	6	6	6	6	6
3	2	2	4	6	7	9	9	11
4	2	2	4	6	7	9	10	**12**

For any feasible solution, $\boldsymbol{x_1 + 3x_2 \leq 4 \Rightarrow 2x_1 + 4x_2 \leq 6}$ holds

Modeling DP states in the proof

Column: $x_1 + 3x_2 \leq 4$

Entry: $2x_1 + 4x_2 \leq 6$

	1	2	3	4	5	6	7	8
1	2	2	2	2	2	2	2	2
2	2	2	4	6	6	6	6	6
3	2	2	4	6	7	9	9	11
4	2	2	4	6	7	9	10	**12**

Weight bound **Profit bound**

For any feasible solution, $\boldsymbol{x_1 + 3x_2 \geq 5} \vee \boldsymbol{2x_1 + 4x_2 \leq 6}$ holds

Modeling DP states in the proof

Column: $x_1 + 3x_2 \leq 4$

Entry: $2x_1 + 4x_2 \leq 6$

	1	2	3	4	5	6	7	8
1	2	2	2	2	2	2	2	2
2	2	2	4	6	6	6	6	6
3	2	2	4	6	7	9	9	11
4	2	2	4	6	7	9	10	**12**

Weight bound **Profit bound**

For any feasible solution, $\boldsymbol{W_5^2} \vee \boldsymbol{P_6^2}$ holds

Interlude: VeriPB

- A proof system and a checker for pseudo-Boolean problems
- Strengthening rules for reasoning without loss of optimality:
  - Introducing new variables
  - Symmetry breaking
  - Dominance reasoning
  - …and many more!

VeriPB: The Easy Way to Make Your Combinatorial Search Algorithm Trustworthy (Gocht, McCreesh, Nordström, 2020)

Modeling DP states in the proof

Starting from the knapsack DP table, we now have new variables

$$W_w^k \leftrightarrow w_1 x_1 + \cdots + w_k x_k \geq w$$

and

$$P_p^k \leftrightarrow p_1 x_1 + \cdots + p_k x_k \leq p$$

- For each $P(w, k) = p$, we want declare $W_{w+1}^k \vee P_p^k$
- Left term is false for the whole knapsack, and P_p^k is an objective bound we are looking for

…and how do you justify $W_{w+1}^k \vee P_p^k$?

Modeling DP transitions in the proof

…and $W_5^2 \vee P_2^1$

True if $W_2^1 \vee P_2^1$…

	1	2	3	4	5	6	7	8
1	2	2	2	2	2	2	2	2
2	2	2	4	6	6	6	6	6
3	2	2	4	6	7	9	9	11
4	2	2	4	6	7	9	10	**12**

Derive an implication $(W_2^1 \vee P_2^1) \wedge (W_5^1 \vee P_2^1) \Rightarrow (W_5^2 \vee P_6^2)$

How well does this work?

- **Enabling** proof logging = writing a few lines to a file per DP entry
- **Verifying** the log is super-linear w.r.t. the number of steps
- Verifying **kernel** proofs scales linearly

Wrap-up

An approach for encoding states and justifying transitions in VeriPB

Low-maintenance technique

- Little performance overhead
- Directly maps to the DP computation
- No need for a separate proof system!

What is next?

- Engineering improvements
- Decision diagram use cases

VeriPB is general enough to capture **diverse** inference techniques