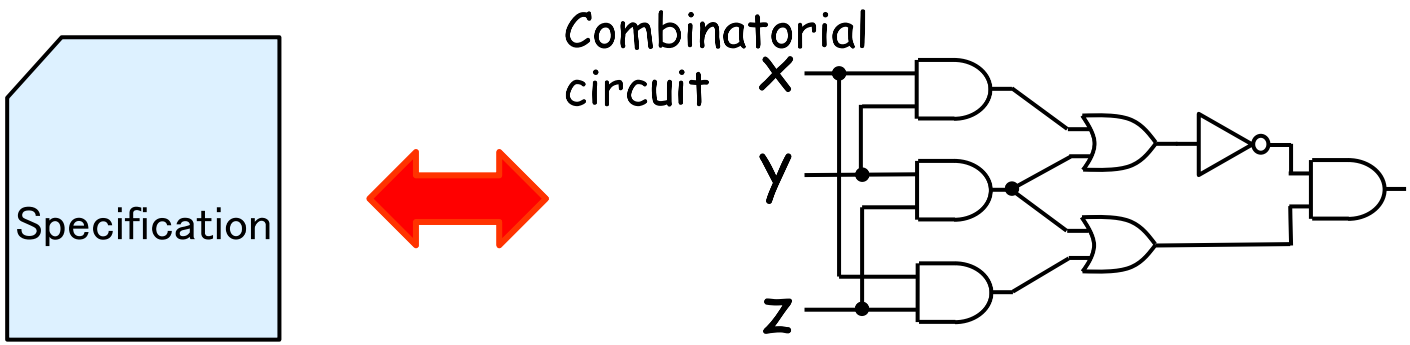
Large-Scale Knowledge Processing #14
Using Binary Decision Diagrams (Part 2)

Faculty of Information Science and Technology, Hokkaido Univ.

Takashi Horiyama

Application of BDDs: Design and verification of logic circuits

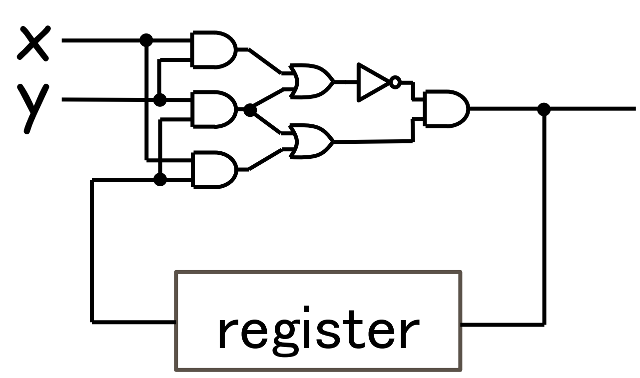
- Verification: Equivalence of **combinatorial circuits**
  - **Description of the specification** (What are outputted from the inputs) and **designed circuit** (In many case, it is optimized by a design compiler)
  - Reference circuit and designed circuit



- Exhaustively check whether the outputs of the above two ("spec. and circuit" etc.) are the same for all inputs (If we use truth tables, we need to check all 2^n cases ...)
- Represent both Boolean functions of the spec. and the designed circuit by **BDDs**
  - **Same Boolean functions** have the **same root node** (Uniqueness)

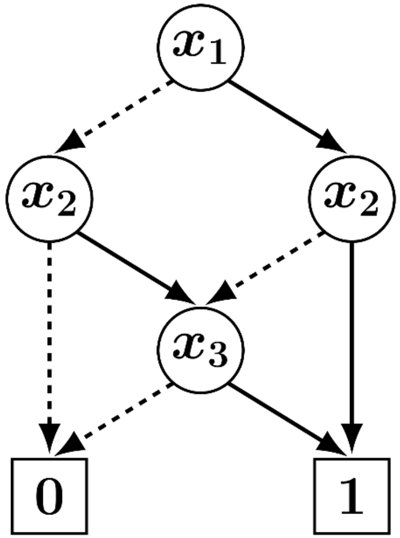
Application of BDDs: Design and verification of logic circuits

- Sequential circuits (circuits that have states, where the states changes according to the inputs)
- Check whether all sequences of the outputs are the same for all sequences of the inputs (This check is much more difficult than that for combinatorial circuits)
- States are stored in registers (memory elements)
- Exhaustively check state transitions on all inputs



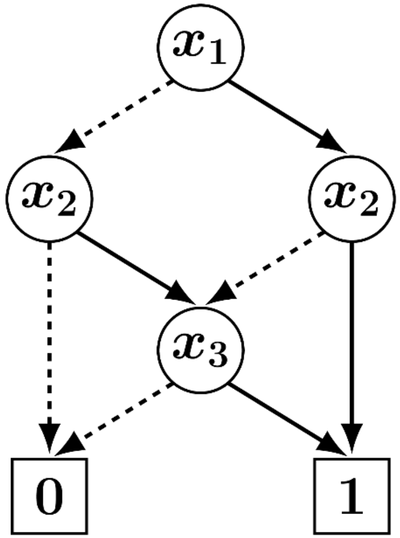
Application of BDDs: Combinatorial problems and optimization

- Represent a combinatorial problem by Boolean variables
- More precisely, represent **constraints** (conditions to be satisfied) of a combinatorial problem by Boolean functions
- Construct a **BDD** representing the AND of the Boolean functions representing the constraints
  - We have **all feasible solutions** !
- If we have BDDs, it is easy to evaluate the linear sum of the costs
  - We can easily obtain the **optimal solutions** from BDDs !



Using BDDs (1)

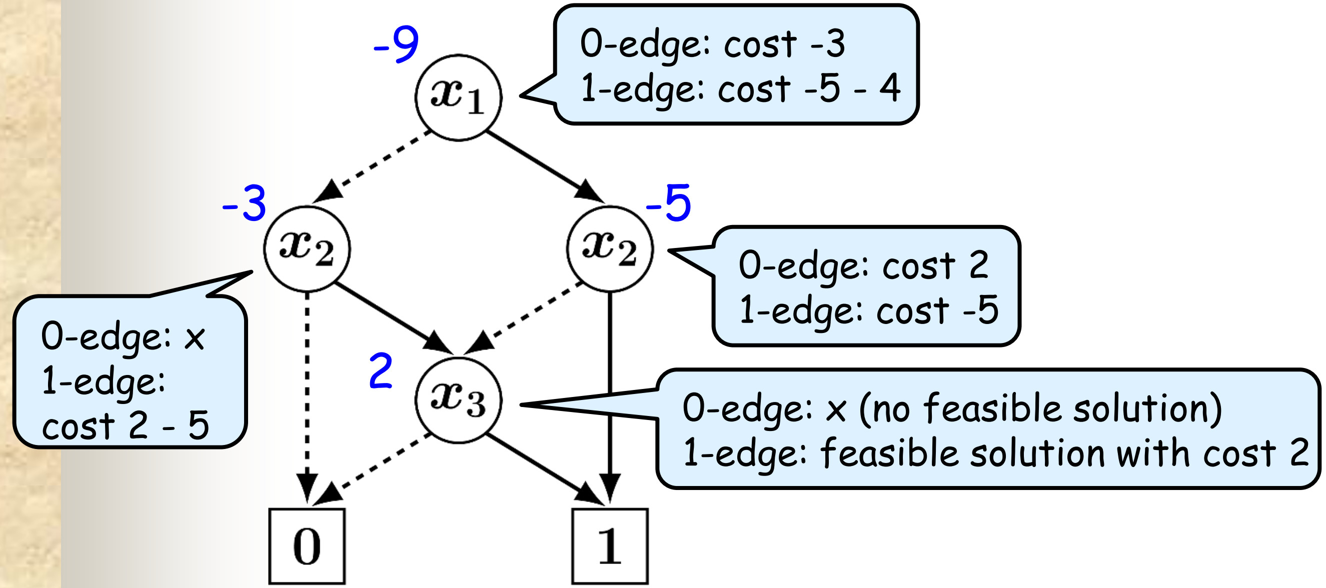
- Determine whether a given combination is a feasible solution or not
  - Traverse a BDD from the root node
  - Ex.) $\{ x_1, x_3 \}$? ⋯ Yes since $(x_1, x_2, x_3) = (1, 0, 1)$ gives $\boxed{1}$
    $\{ x_2 \}$? ⋯ No since $(x_1, x_2, x_3) = (0, 1, 0)$ gives $\boxed{0}$
- Enumerate all solutions from a BDD
  - **Depth first search**
  - When we come to the 1-node, we have a feasible solution
  - Ex.) $\{x_2, x_3\}, \{x_1, x_3\}, \{x_1, x_2\}, \{x_1, x_2, x_3\}$



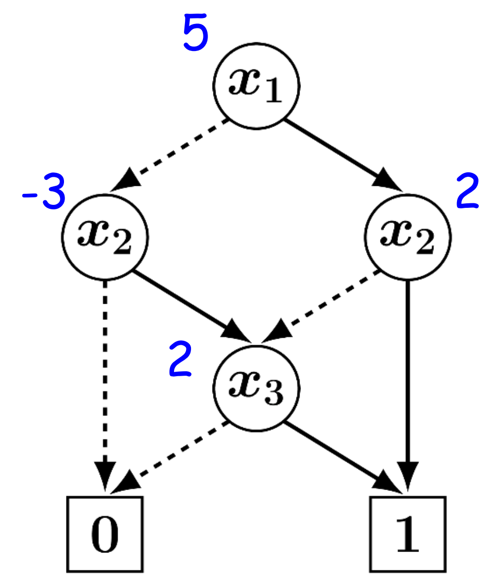
Using BDDs (2)

- Evaluation of the linear sum of the cost
  - At each node, the costs for its 0-edge and 1-edge are compared, and the better edge is adopted
  - The cost of the root node is the optimal value

Ex.) minimize $-4x_1 - 5x_2 + 2x_3$



maximize $3x_1 - 5x_2 + 2x_3$



Short break

- Take a deep breath, and relax yourself

Combinatorial optimization: Knapsack problem

Price	60 yen	30 yen	40 yen	50 yen	40 yen	30 yen
Utility	36	27	12	50	28	24

- We can buy snacks **within 100 yen** (i.e., the **budget is 100 yen**)
  - We can adopt (i.e., take) at most 1 piece for each item
  - We cannot divide an item
- We'd like to **maximize** the **sum of the utilities**

The images of the snacks are from the web sites of Meiji Co. Ltd., and Morinaga & Co., Ltd.
https://www.meiji.co.jp https://www.morinaga.co.jp

Knapsack problem : Variable x_i

Item	Item_1	Item_2	Item_3	Item_4	Item_5	Item_6
Price (yen)	60 yen	30 yen	40 yen	50 yen	40 yen	30 yen
Utility	36	27	12	50	28	24

- $$x_i = \begin{cases} 1 & \cdots \text{ We take item}_i \\ 0 & \cdots \text{ Do not take item}_i \end{cases}$$
- We can buy snacks **within 100 yen** (i.e., the **budget is 100 yen**)
  - $60\,x_1 + 30\,x_2 + 40\,x_3 + 50\,x_4 + 40\,x_5 + 30\,x_6 \leqq 100$

Knapsack problem : Formulation

Item	$Item_1$	$Item_2$	$Item_3$	$Item_4$	$Item_5$	$Item_6$
Price (yen)	p_1	p_2	p_3	p_4	p_5	p_6
Utility	u_1	u_2	u_3	u_4	u_5	u_6

- **Objective function**
  - Maximize : $\sum_{i=1}^{6} u_i x_i$
- **Constraints**
  - $\sum_{i=1}^{6} p_i x_i \leqq 100$
  - $x_i \in \{0, 1\} \quad (i = 1, 2, \ldots, 6)$

0-1 integer programming problem

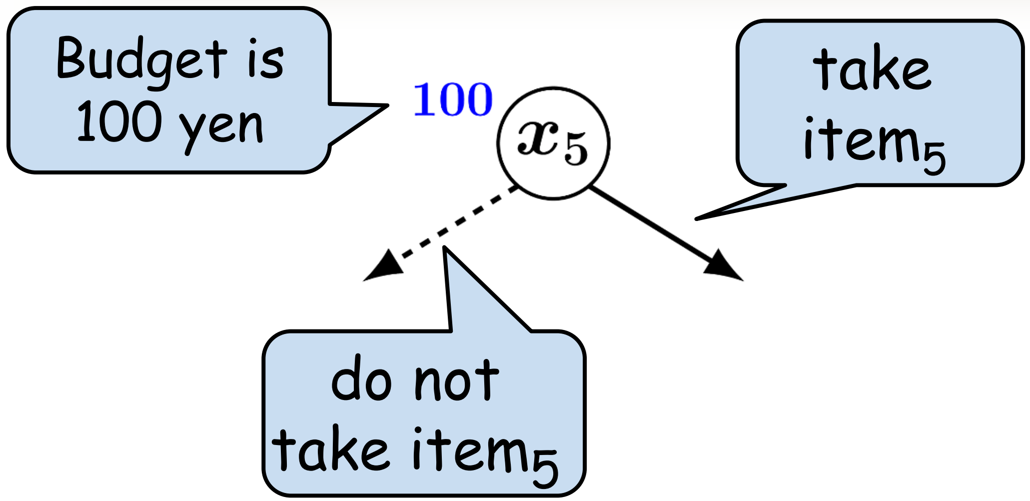
Combinatorial optimization: Knapsack problem

- Represent constraint $\Sigma_i\, p_i\, x_i \leqq c$ by a BDD

  (How ? → see the next slide)

- Maximize the objective function $\Sigma_i\, u_i\, x_i$
  (see p. 6)

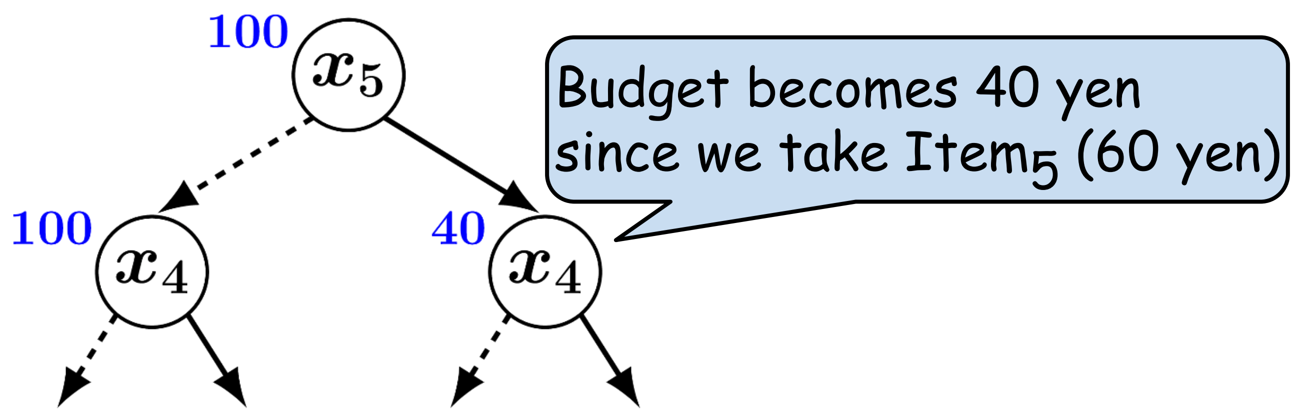
Represent the constraint of the knapsack problem by a BDD



Budget 100 yen

Item	Item_5	Item_4	Item_3	Item_2	Item_1
Price (yen)	60 yen	30 yen	30 yen	60 yen	30 yen
Utility	36	27	12	50	28

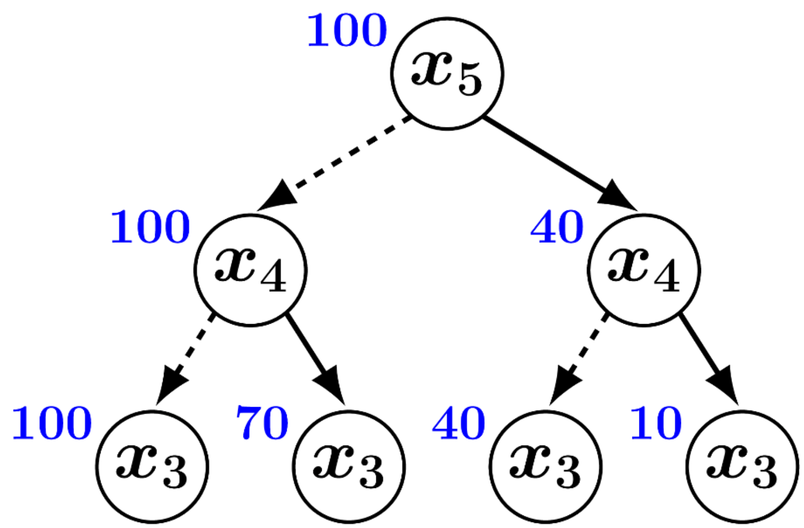
Represent the constraint of the knapsack problem by a BDD



Budget
100 yen

Item	Item$_5$	Item$_4$	Item$_3$	Item$_2$	Item$_1$
Price (yen)	60 yen	30 yen	30 yen	60 yen	30 yen
Utility	36	27	12	50	28

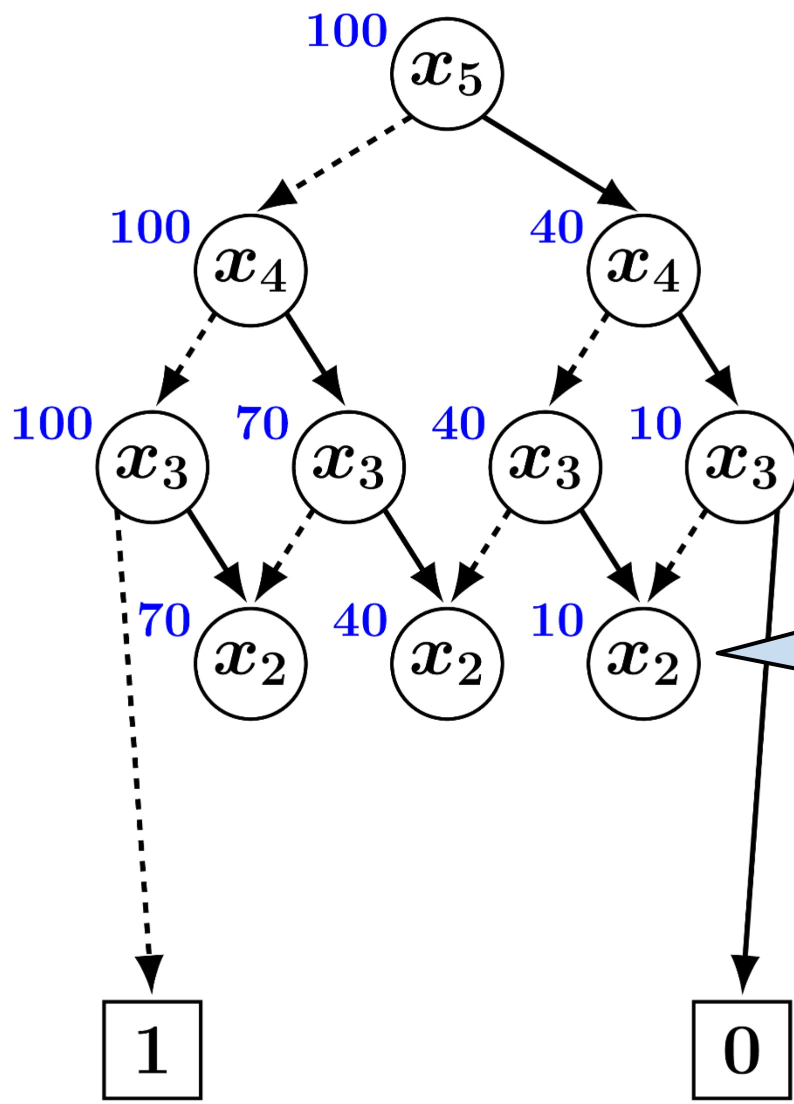
Represent the constraint of the knapsack problem by a BDD



Budget
100 yen

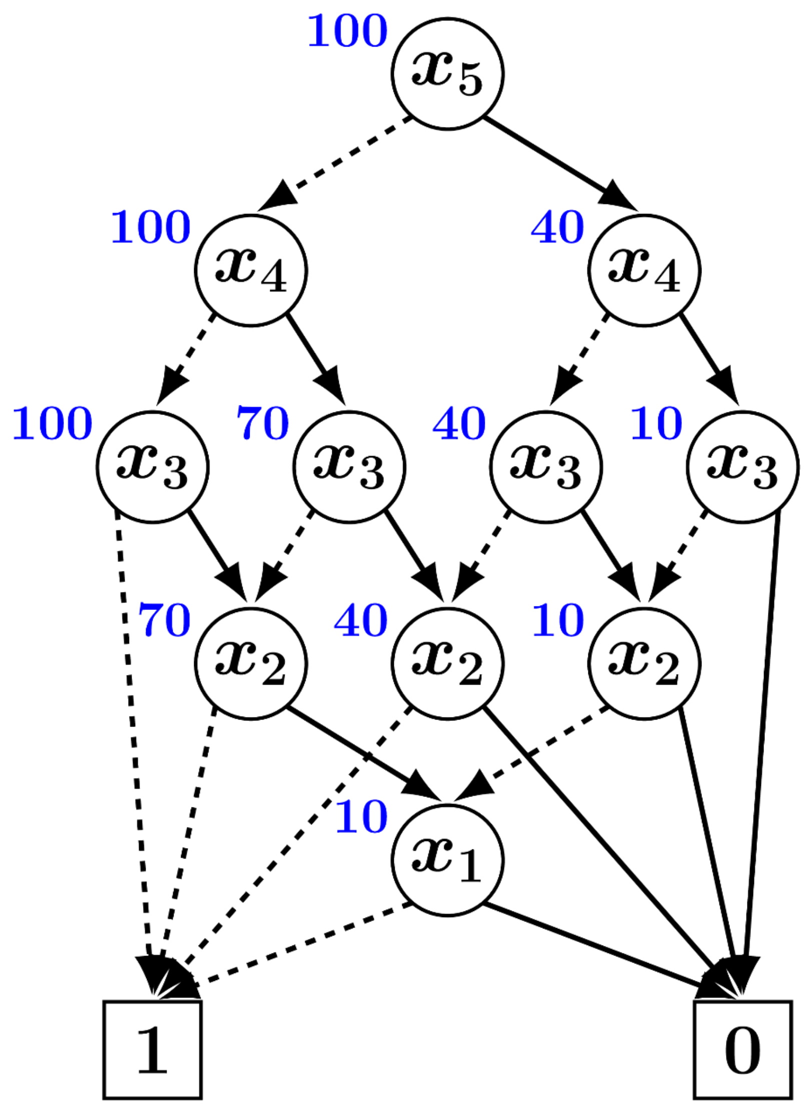
Item	Item$_5$	Item$_4$	Item$_3$	Item$_2$	Item$_1$
Price (yen)	60 yen	30 yen	30 yen	60 yen	30 yen
Utility	36	27	12	50	28

Represent the constraint of the knapsack problem by a BDD



Equivalent nodes are shared
(Budget for this subproblem is 10)

Represent the constraint of the knapsack problem by a BDD



Represent the constraint of the knapsack problem by a BDD

- Once the BDD representing the constraints are obtained, it is easy to minimize/maximize the linear sum of the costs
- Note that, however, if our task is just an optimization, it is not recommended to construct BDDs (since it requires much computation time and huge memory consumption)
- Benefits of constructing BDDs
  - Optimization with changing the objective function (We can ovoid searching the solution space twice or more for different objective function)
  - We can combine the BDDs with other constraints (Apply operation is easy)

Summary

- Design and verification of logic circuits
- Combinatorial problems and optimization
  - Determine whether a given combination is a feasible solution or not
  - Evaluate the linear sum of the costs
  - Ex.) Knapsack problem