

Exploiting Symmetry in Multiple Knapsack Problems

Alex Fukunaga

Global Edge Institute

Tokyo Institute of Technology

fukunaga@is.titech.ac.jp



Multiple Knapsack Problem (MKP)



- **Generalization of Knapsack to multiple containers**
 - n items with weights w_1, \dots, w_N , profits p_1, \dots, p_N
 - m bins with capacities c_1, \dots, c_M
 - Each item assigned to at most one bin (container)
 - Decision variable $x_{ij} = 1$ iff item j is assigned to bin i , otherwise $x_{ij} = 0$

- **Maximize**
$$\sum_{i=1}^m \sum_{j=1}^n p_j x_{ij}$$
 maximize sum of profits

- **Subject to**
$$\sum_{j=1}^n w_j x_{ij} \leq c_i, i = 1, \dots, m$$
 capacity constraints

$$\sum_{i=1}^m x_{ij} \leq 1, j = 1, \dots, n$$

assign each item to at most one bin

$$x_{ij} \in \{0,1\} \forall i, j$$

- Complexity: **Strongly-NP-complete**

Applications of the MKP

- **Continuous, double-call auctions**
- **Multiprocessor scheduling**
- **Task assignment in multi-agent systems (staff scheduling, autonomous robots)**

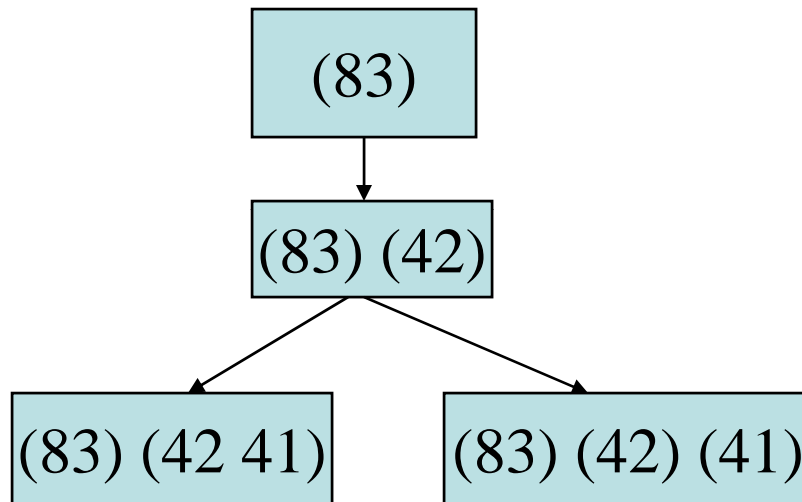
A (Slightly) Simpler Problem: Multiple-Subset-Sum Problem (MSSP) Bin Packing

- **Given:**
 - (one-dimensional) Items with weights: 9,8,4,3,1
 - 2 Bins with capacities 10, 10 (assume all bins have same capacity)
- **Find assignment of items to bins which minimizes sum of wasted space in bins**
 - Equivalent to MKP where $p_i = w_i$ for all i
- **Solution: {9,1} and {8} , wasted space = 2**
- **Complexity: Strongly NP-complete**
- **We'll use MSSP to explain key ideas because it is less complicated than the MKP**

Natural Problem Space – Item Oriented

- “Item-oriented” Branch-and-bound
- Each node corresponds to an item
- Each branch assigns an item to a different bin (possibly assigns it to no bin).
- Previous research focused on efficient upper and lower-bounding techniques.
- Most of the previous branch-and-bound algorithms for multiple-container problems use this problem space.
 - Martello & Toth, 1990 (bin packing)
 - Pisinger, 1999 (multiple knapsack)
 - Labbe, Laporte, Toth, 1995 (bin covering)

Item-Oriented Problem Space for Instance {5,12,40,41,42,83}, bin capacity=100

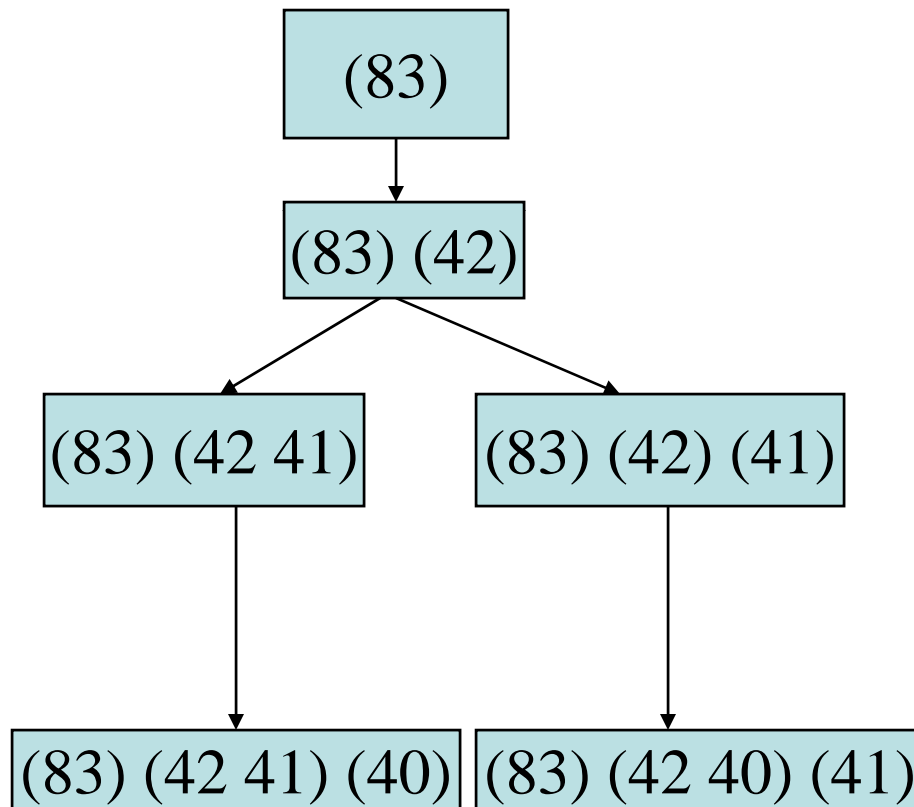


Assign 83 to a bin

Assign 42 to a bin

Assign 41

Item-Oriented Problem Space for Instance {5,12,40,41,42,83}, bin capacity=100



Assign 83 to a bin

Assign 42 to a bin

Assign 41

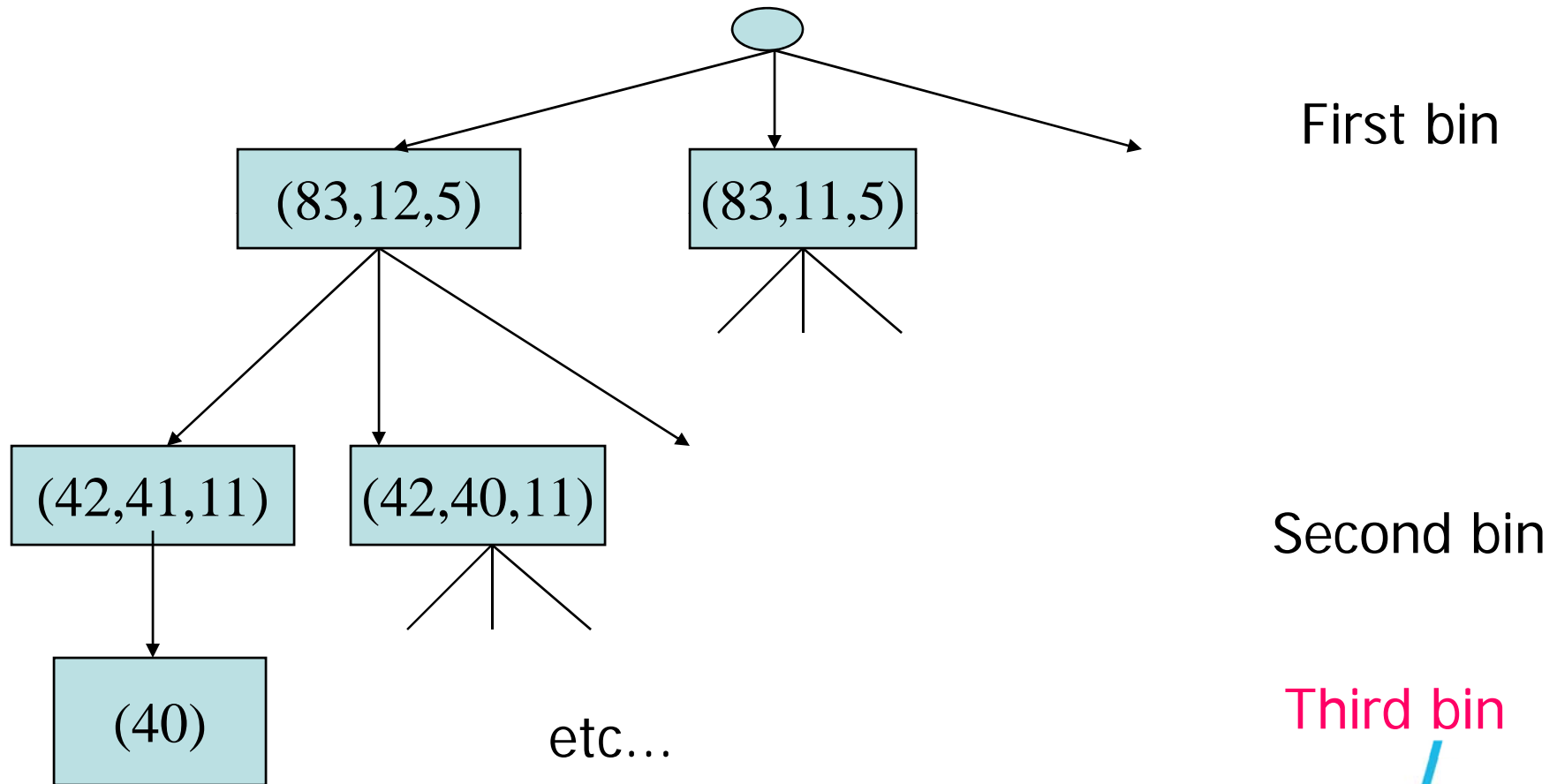
Assign 40

... and so on...

An alternate search space: “Bin-oriented” Branch-and-bound

- Each node corresponds to a bin
- Instead of considering one item at a time
 - Each branch represents a different set of numbers to assign to the bin.
- Feasible set: capacity constraint satisfied
- Maximal set: A Feasible set S where there does not exist any item I s.t. $S \cup I$ is feasible.
- e.g., Bin capacity 10, Items 5,4,3,2
 - {5,4} is maximal
 - {5,3} is feasible but not maximal
 - {5,4,3} is not feasible
- **Assign Maximal, Feasible set of items to bins**

Bin-Oriented Problem Space for Items={5,11,12,40,41,42,83}, all bins with capacity=100

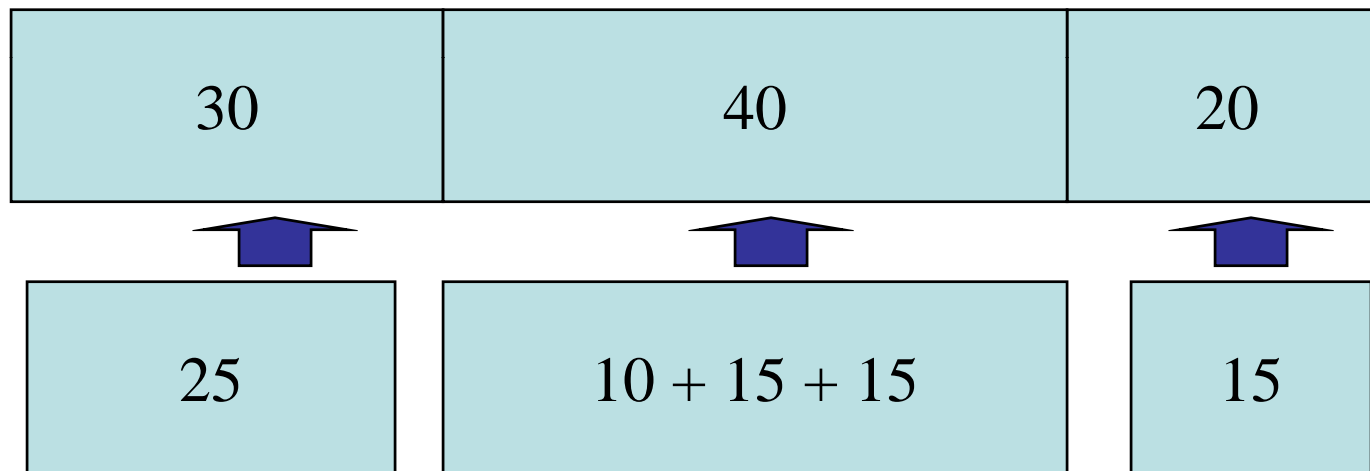


Dominance - Intuition

- Items = {96,80,15,12,4,3}, bins with capacity = 100, 100
- {96,3} and {96,4} are maximal, feasible assignments
- **Case A: assign {96,4} to a bin**
 - wasted space=0, remaining subproblem is {80,15,12,3}
- **Case B: assign {96,3} to a bin**
 - wasted space=1, remaining subproblem is {80,15,12,4}
- **The total wasted space in the optimal solution in Case A must be less than or equal to the total wasted space in the optimal solution in Case B.**
 - Optimal for case A: {80,15,3}, waste=2, total waste=2
 - Optimal for case B: {80,15,4}, waste=1, total waste=2
- **Intuition: Don't waste space early!**
- **Thus, {96,4} dominates {96,3}.**
 - The optimal solution in the subtree under {96,4} is *at least as good* as the optimal solution under {96,3}, so prune {96,3}

General Dominance Criterion (based on Martello-Toth, 1990)

- A dominates B if we can pack subsets of B into items of A
- (30,40,20) dominates (25,15,15,15,10)

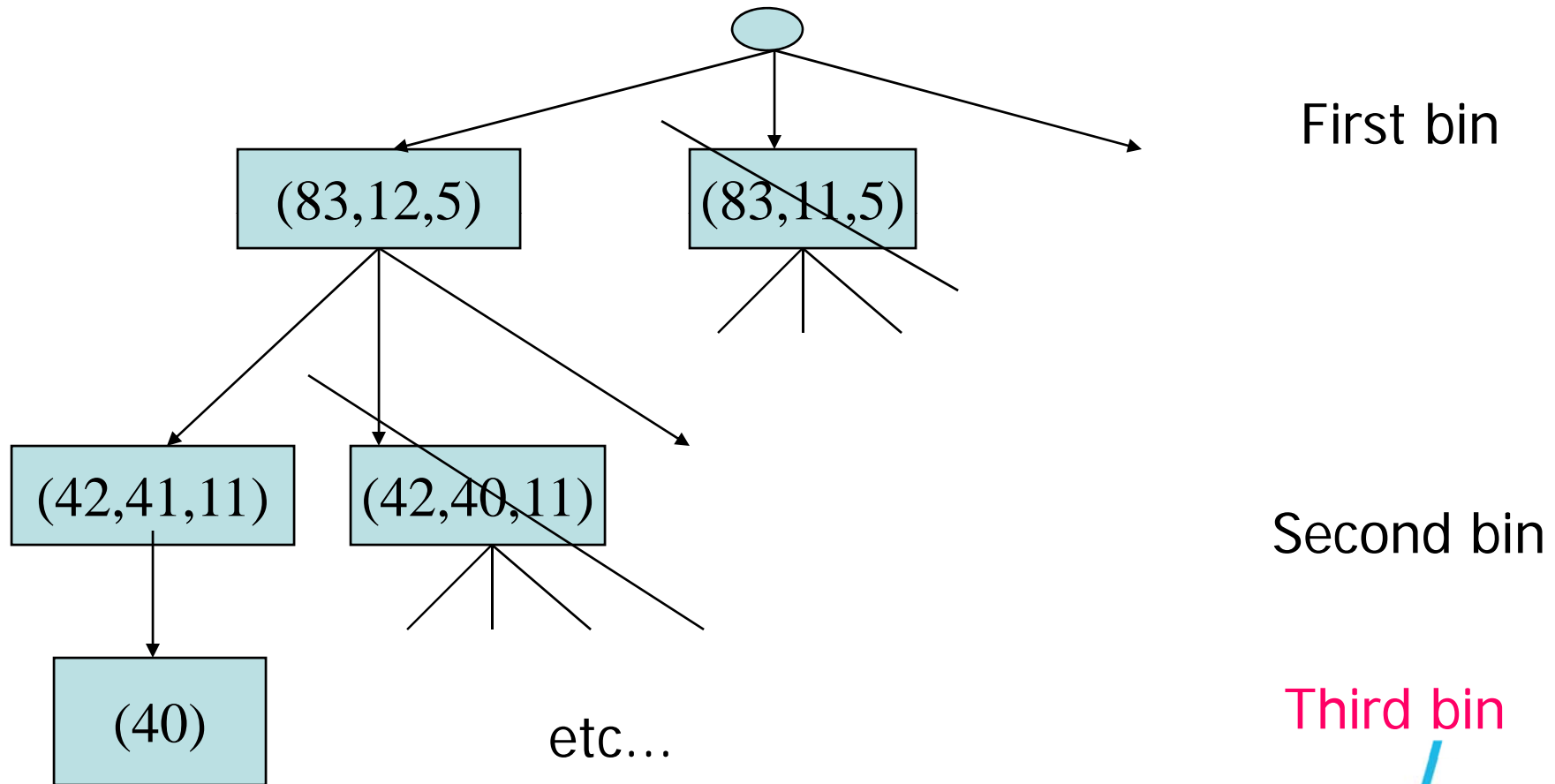


- In general, a dominance check is a smaller, NP-complete bin packing problem!.

Bin Completion Algorithm

- Depth-first branch-and-bound in bin-oriented problem space
 - **Each node corresponds to a maximal, feasible set.**
- Prune search using a dominance criterion among bins
- For each bin, consider only the undominated bin assignments
 - **Nontrivial to do enumerate only undominated bin assignments**
 - **Linear-space algorithm for enumerating undominated assignments (linear in # of remaining items)**
 - (Fukunaga and Korf, JAIR, 2007)

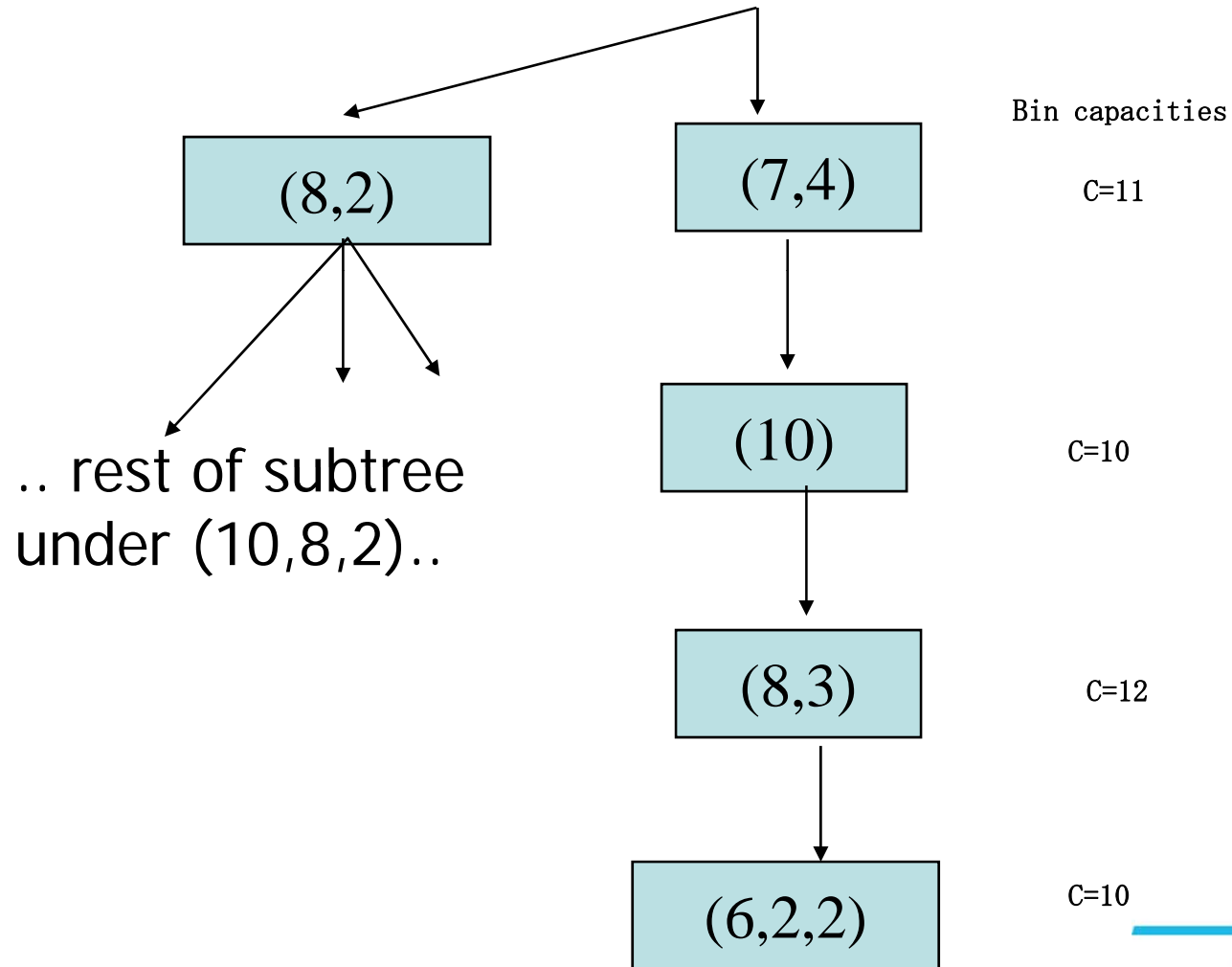
Bin-Oriented Problem Space for Instance {5,11,12,40,41,42,83}, all bins have capacity=100



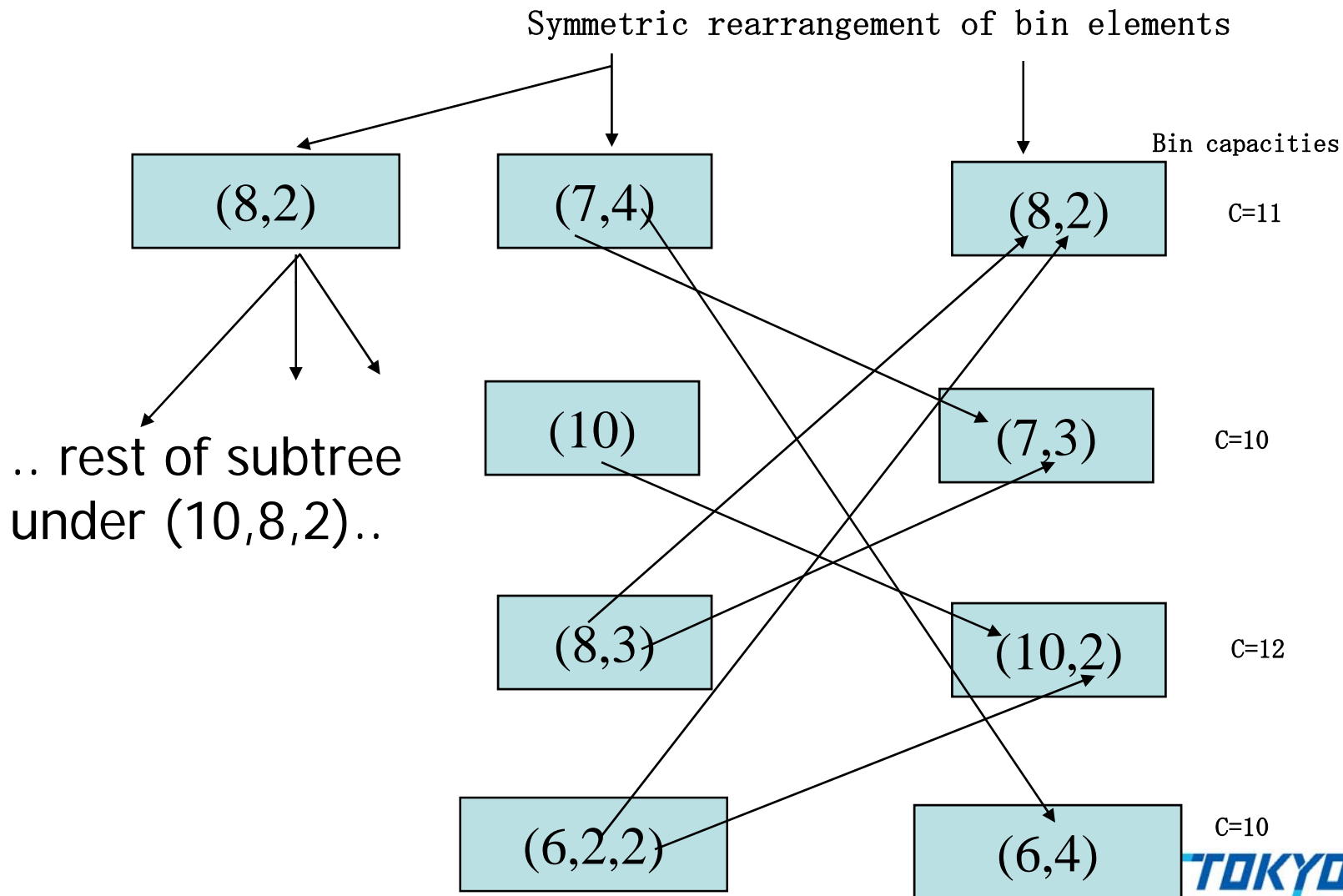
Exploiting Symmetry in the Bin Completion Search Tree

- **Most nodes are redundant due to symmetry**
- **Symmetries become evident in bin-oriented search space.**
- **Detect symmetries in search tree**
 - Path Symmetry
- **Exploit dominance condition deep in tree**
 - Path Dominance
- **General approach: Symmetry Breaking via Dominance Detection (SBDD) (Fahle, Schamberger, Sellmann, 2001, Focacci and Milano 2001).**

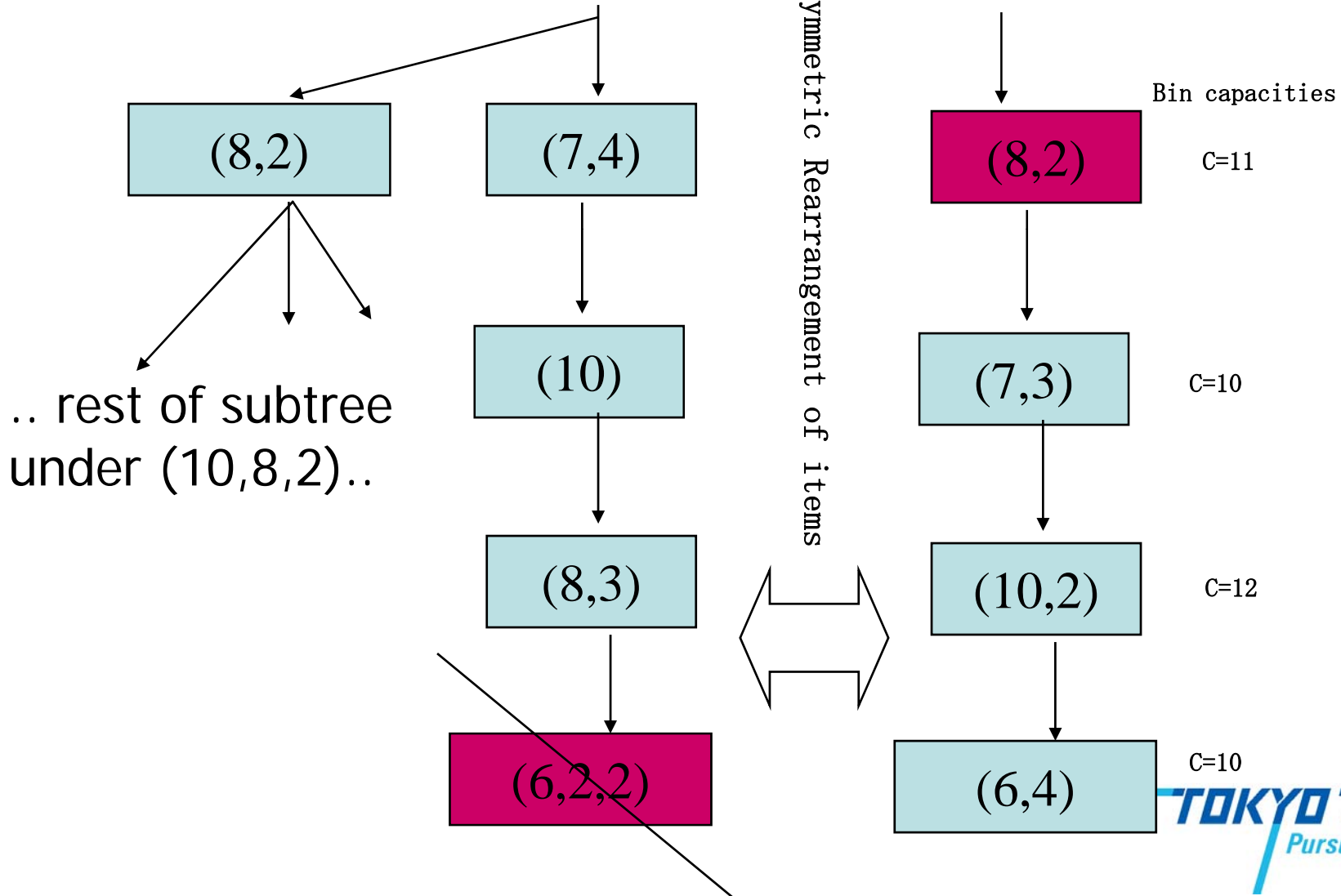
Path Symmetry



Path Symmetry

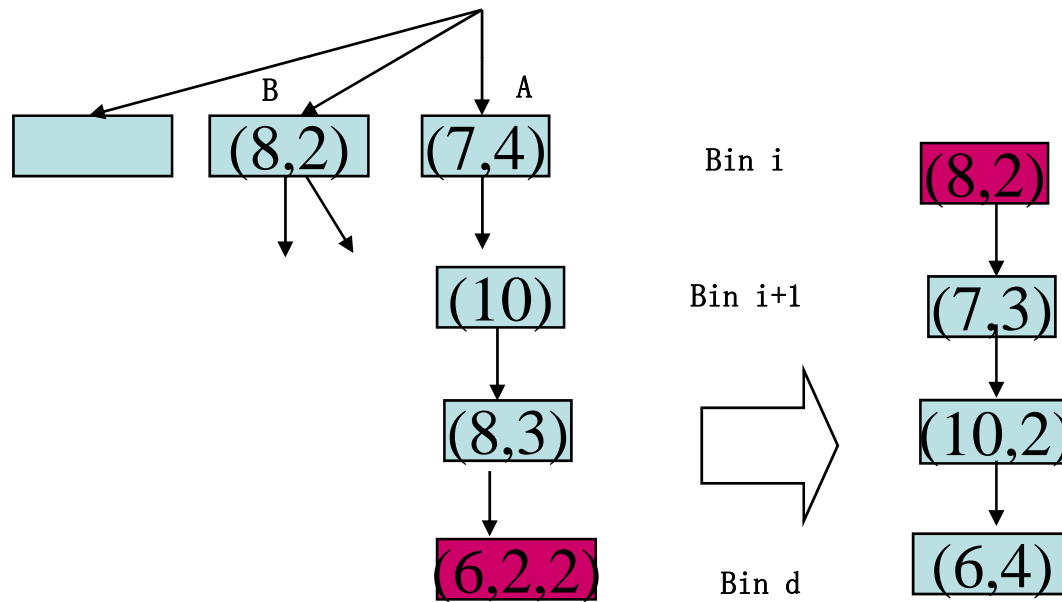


Path Symmetry



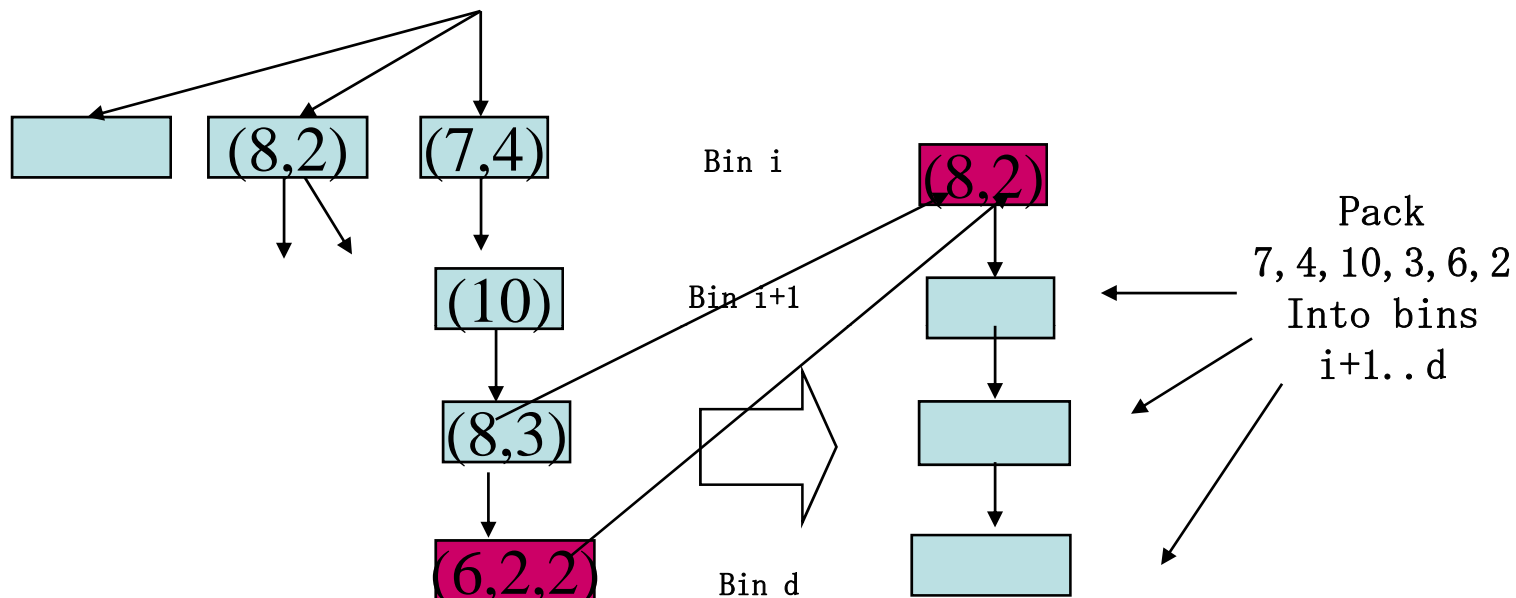
Path Symmetry

- Can prune node N at depth d if:
 - N has an ancestor A at depth $i < d$ such that there exists a sibling of A, B, s.t. there is a *feasible* rearrangement of items in bins $i, i+1, \dots, d$ s.t. the new contents of bin i is equivalent to B.



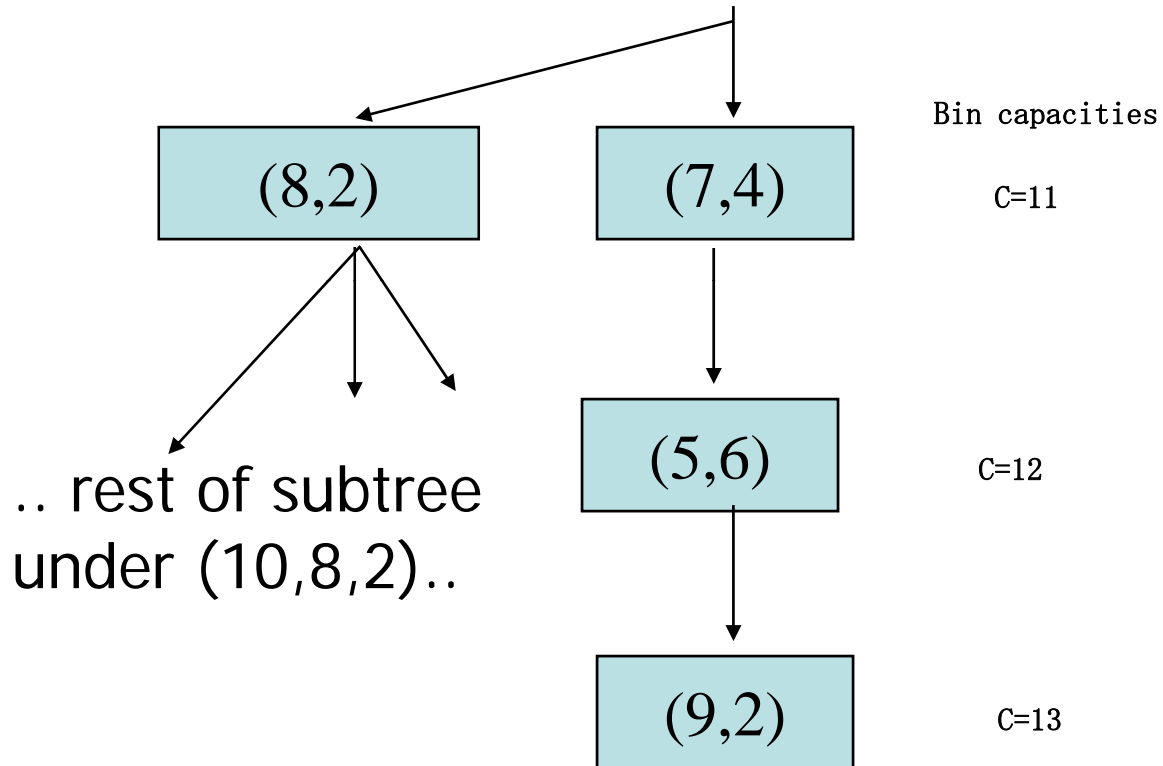
Detecting Path Symmetry

- :

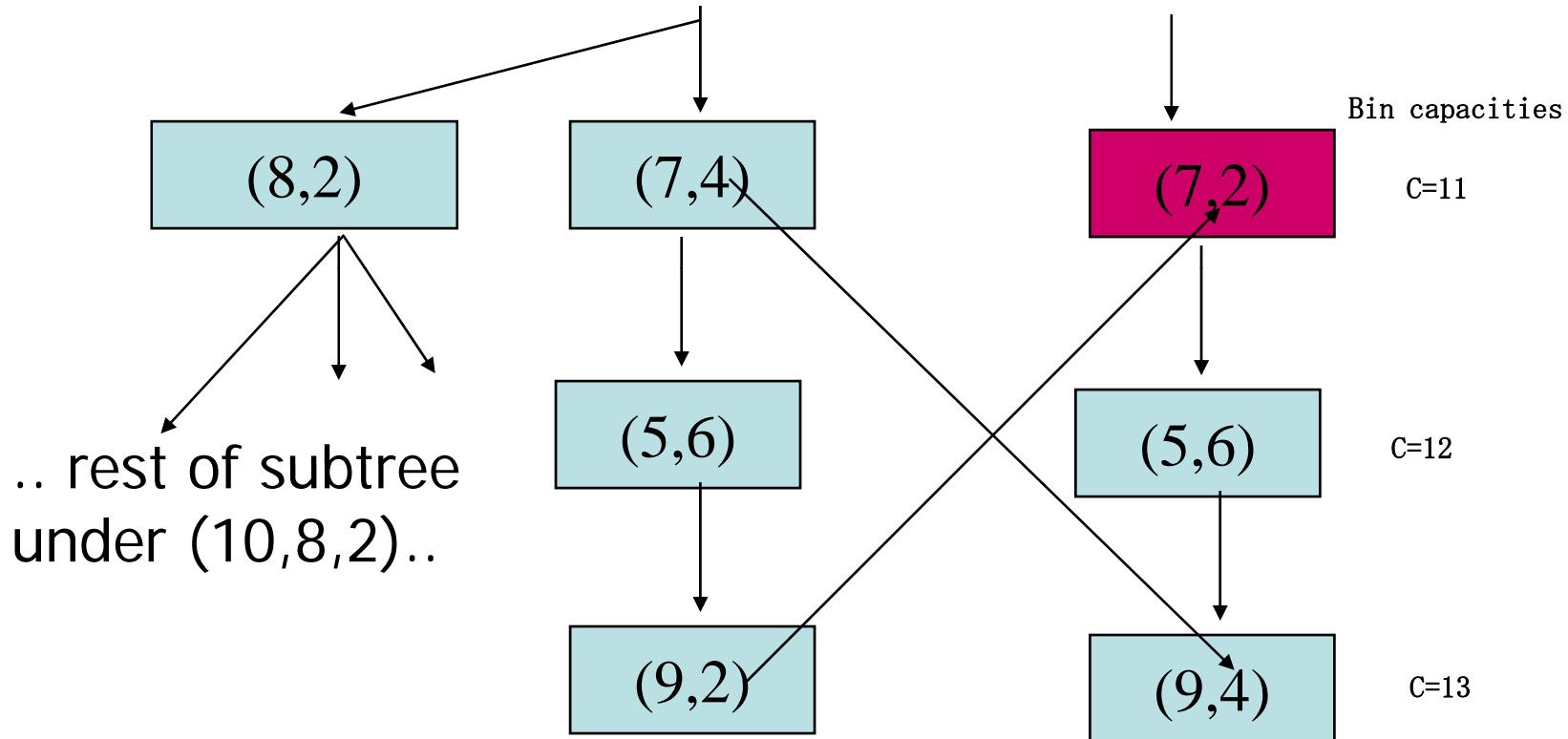


- **Symmetry detection by solving a small-scale bin packing problem.**
 - Bin-packing is strongly NP-complete!
 - Solve using backtracking / branch-and-bound
 - Finds all path symmetries, but slow
 - Solve using approximation algorithm (e.g., First-fit decreasing)
 - Might miss symmetry, but fast

Path Dominance



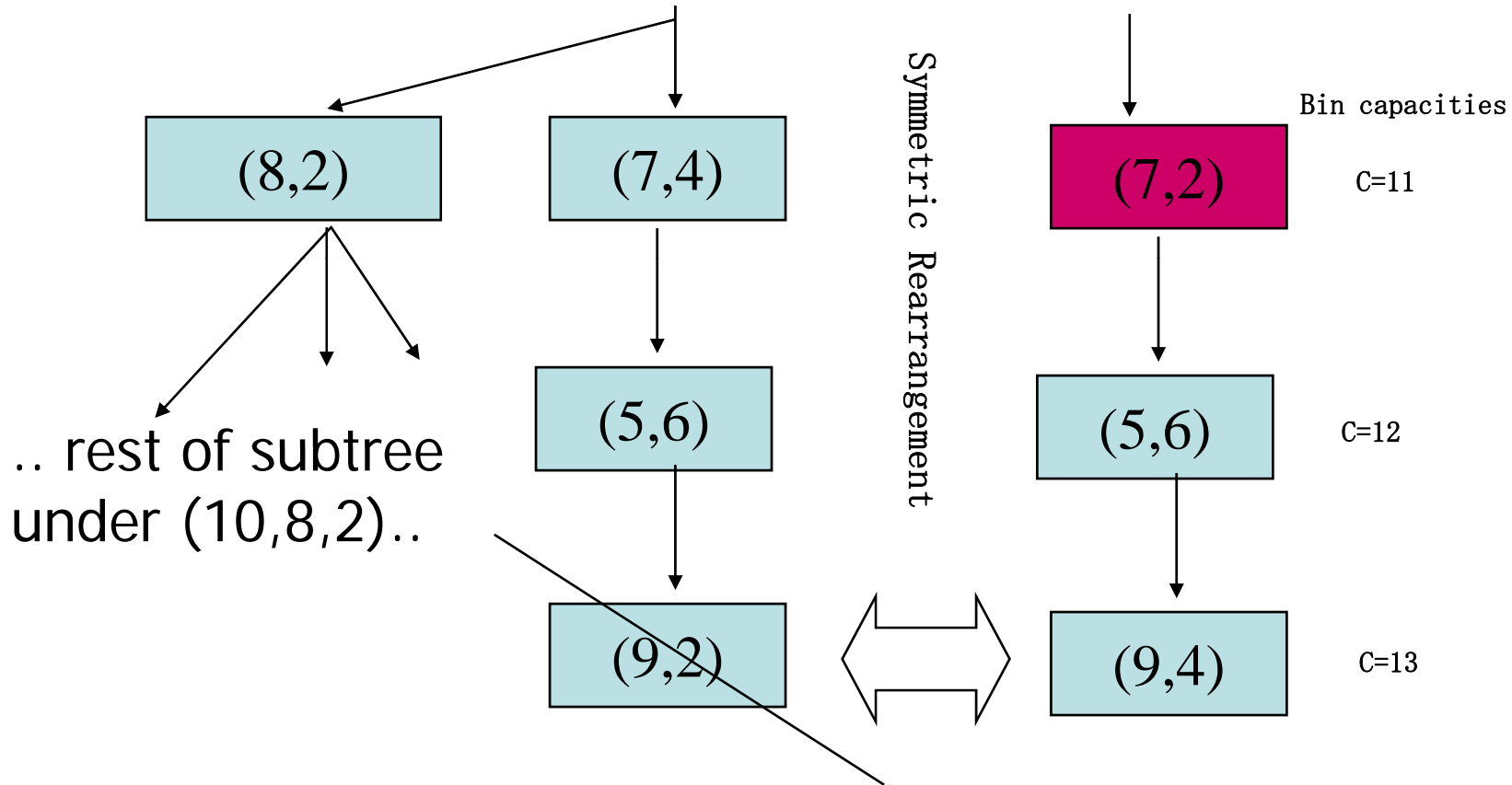
Path Dominance



Apply symmetry to rearrange items without changing optimal value

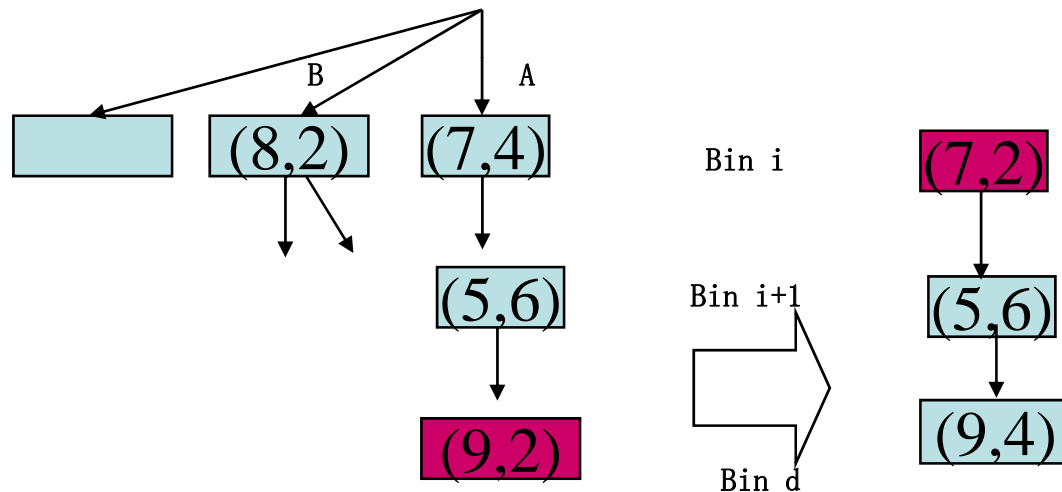
C=10

Path Dominance



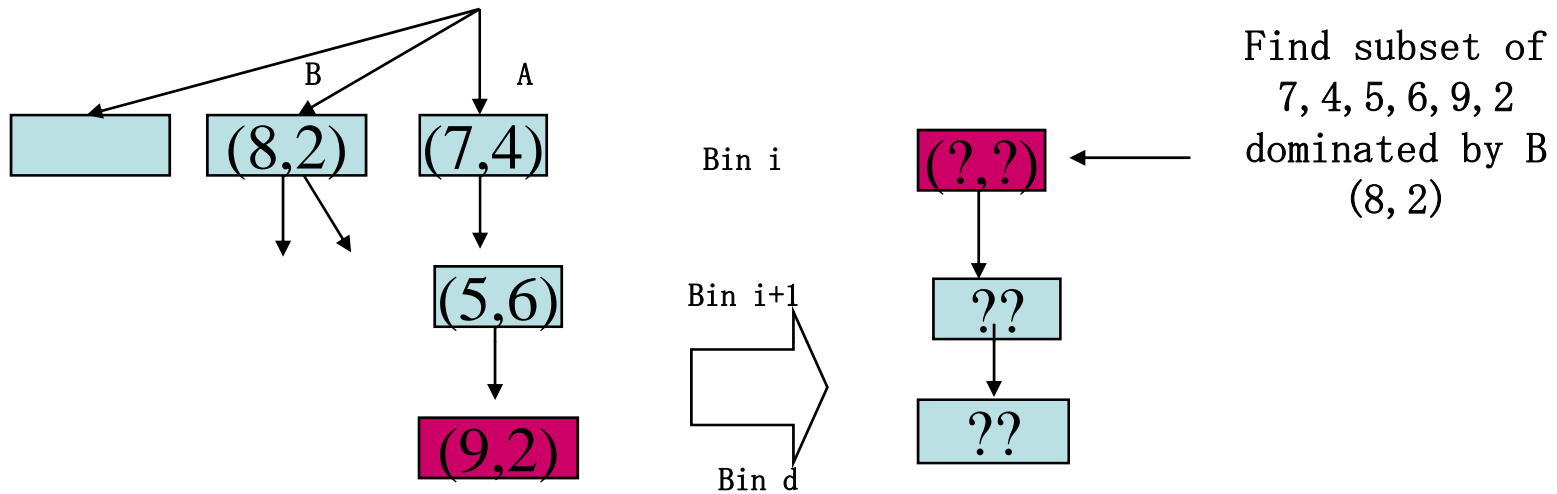
Path Dominance

- Can prune node N at depth d if:
 - N has an ancestor A at depth $i < d$ such that there exists a sibling of A, B, s.t. there is a *feasible* rearrangement of items in bins $i, i+1, \dots, d$ s.t. the new contents of bin i is dominated by B.



Detecting Path Dominance

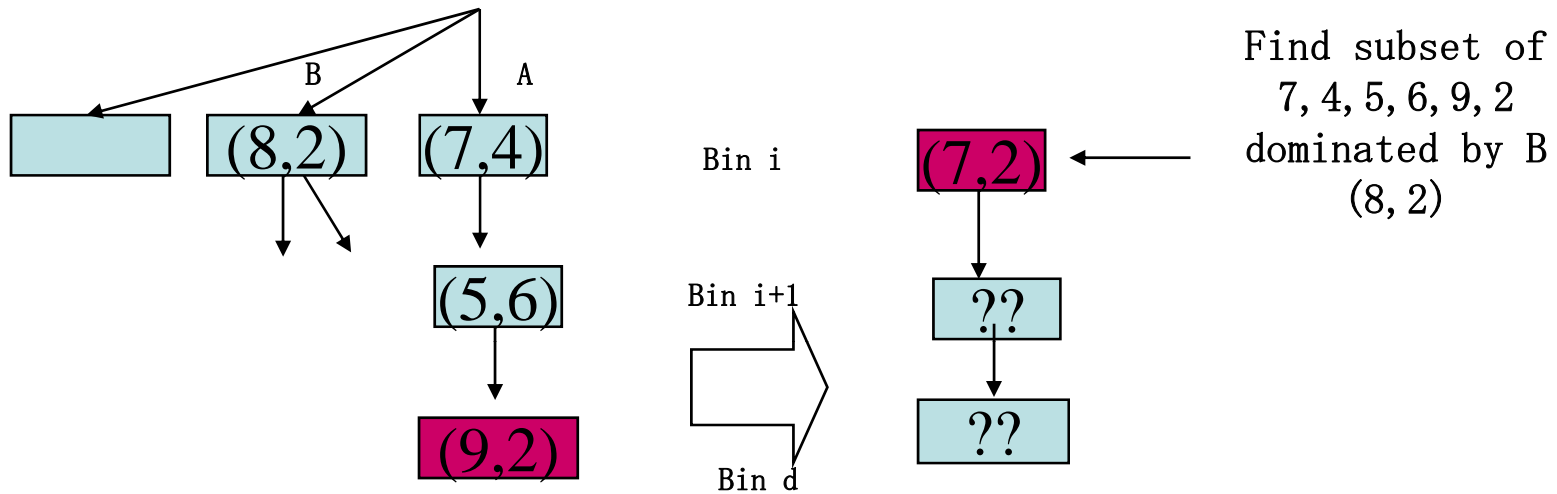
- :



- **Step 1: Identify subset of items in bins i, \dots, d which is dominated by B**

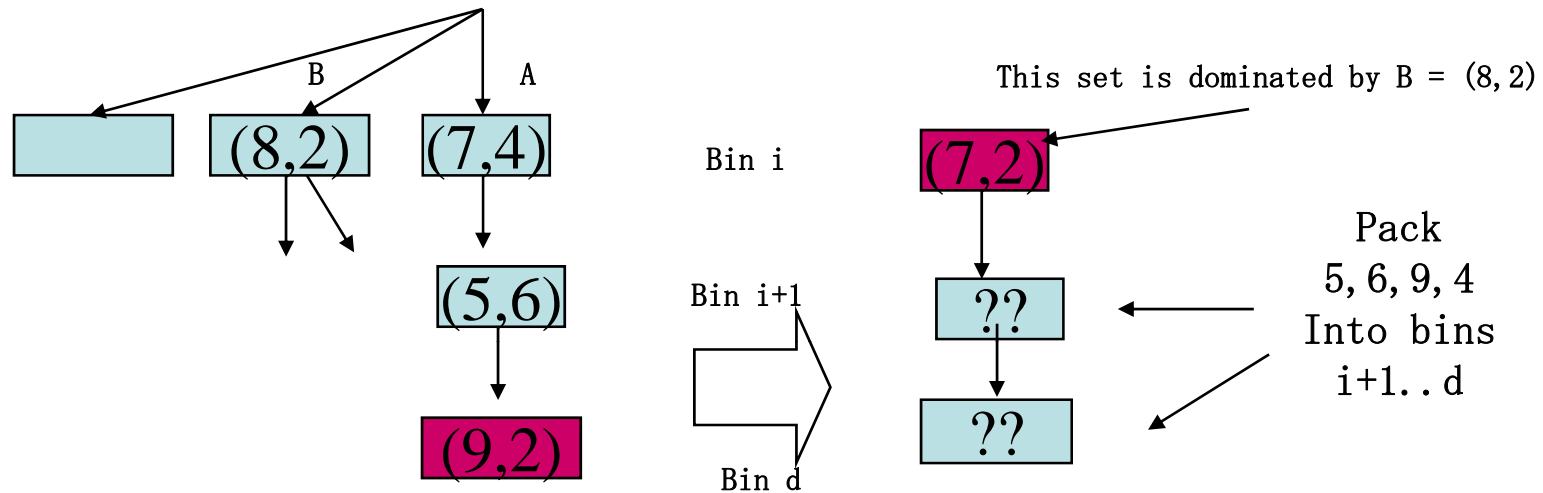
Detecting Path Dominance

- :



- **Step 1: Identify subset of items in bins i, \dots, d which is dominated by B**

Detecting Path Dominance



- **Step 2: Pack remaining items into Bins $i+1, \dots, d$.**
 - Solve a small bin packing problem
 - Solve using backtracking / branch-and-bound
 - Finds all path symmetries, but slow
 - Solve using approximation algorithm (e.g., First-fit decreasing)
 - Might miss symmetry, but fast

Approximate Checks

- **Detect most symmetries with relatively low overhead per search node**
- **First-fit heuristic for bin packing subproblem**
- **Selective “locking” of bins/items in order to reduce search space**
- **Better methods? Work in progress**

Experimental Comparison

- **Mulknap (Pisinger, 1999)**

- Previous state of the art for Multiple Knapsack
- Item-oriented branch-and-bound
- At each node:
 - Upper bound: Surrogate relaxed Multiple Knapsack problem
 - solve single 0-1 Knapsack problem for remaining items, where capacity = sum of remaining bins.
 - Reductions
 - Capacity Tightening
 - Bound-and-Bound technique: Compute lower bound (subset-sum heuristic), and if lower bound = upper bound, backtrack.\

- **Bin Completion**

- Bin-oriented search with dominance-based pruning
- same upper bound as Mulknap
- same reduction algorithm as Mulknap

- **Bin Completion with Path Symmetry, approximate check (first-fit decreasing heuristic for bin packing subproblem; selective locking of bins/items)**

- **Bin Completion with Path Dominance, exact check (backtracking for bin packing subproblem)**

Experimental Evaluation

- **30 instances per set (m, n pair)**
- **600 second time limit per instance**
- **“Uncorrelated Problems”**
 - weight w_j uniformly chosen from $[10, 1000]$
 - profit p_j uniformly chosen from $[10, 1000]$
- **Focus on hard problems (low n/m ratio)**

Uncorrelated MKP instances

	Mulknab (Pisinger, 1999)	Bin Completion	<u>Fastest</u> Bin Completion +Path Symmetry, Approximate checks	<u>Most Efficient</u> Bin Completion +Path Dominance, Exact checks
10 bins 20 items	52.08	< 0.001	< 0.001	< 0.001
20 bins 40 items	> 600 (30)	49.436 (7) 7,545,714	0.10 (0) 3341	1.513 (0) 1924
10 bins 30 items	29.96 (11)	9.130 (0) 1,662,504	0.431(0) 6761	0.786 (0) 5031

- **Reported values:**
 - Average time, in seconds (for successful runs) –
 - 1.7GHz Athlon for Mulknab (gcc -O3)
 - 2.7GHz Core2 (single-threaded) for all Bin completion implementations (CMUCL Common Lisp)
 - (# of timeouts out of 30 runs, given 600 seconds)
 - Nodes (for successful runs)
- **Similar results with subset-sum instances, highly correlated, and weakly correlated instances with various # of bins and items (see paper)**

Summary of Results

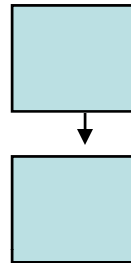
- **Also compared with special cases of path symmetry and path dominance originally proposed in (Fukunaga and Korf, JAIR 2007)**
- **Fastest configuration (Bin-completion + Path symmetry using approximate checks)**
 - 2-3x faster runtimes than best previous algorithm (bin-completion with 2-swap dominance, Fukunaga and Korf, 2007), searches >10x fewer nodes.
- **Most efficient configuration (Bin-completion + Path dominance using exact checks)**
 - ~2x more efficient than fastest configuration, but significantly slower.
 - Future work: better approximation to close this 2x gap?
- **Symmetry is highly effective for MKP**
 - Fastest configuration searches at least 1-2 orders of magnitude fewer nodes than “pure bin-completion” (no symmetry).
 - Fastest configurations runs up to 3 orders of magnitude faster than “pure bin completion”

Related Work

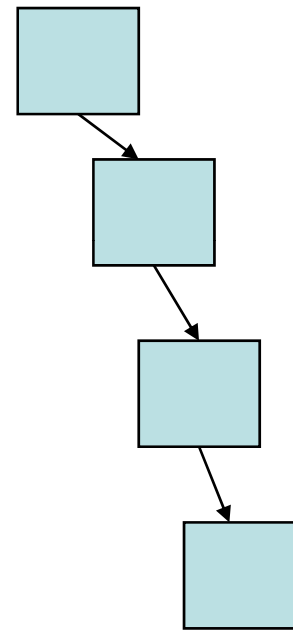
- **Our MKP symmetry detection is an instance of the Symmetry Breaking via Dominance Detection (SBDD) approach (Fahle, Schamberger, Sellmann, 2001, Focacci and Milano 2001).**
- **Most similar to pruning method of Focacci and Shaw, 2002.**

A generalized view of our symmetry detection algorithm: Comparison with Focacci and Milano, 2002

i-variable partial solution
(Nogood)



j-variable partial solution

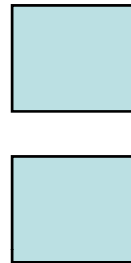


Can we prune this search node?

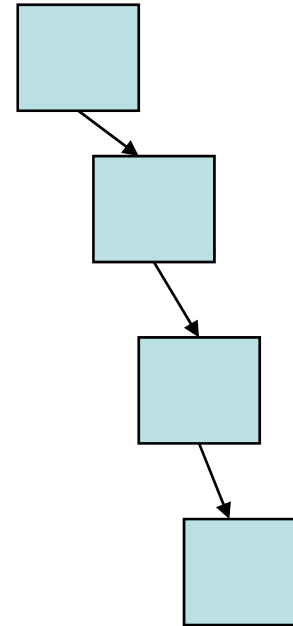
- **General approach: show that current j -variable partial solution can be pruned due to some i -variable nogood, $i < j$**
 - Same general idea as Focacci and Milano, 2002

Focacci and Milano, 2002 – Nogood Extension

i-variable partial solution
(Nogood)



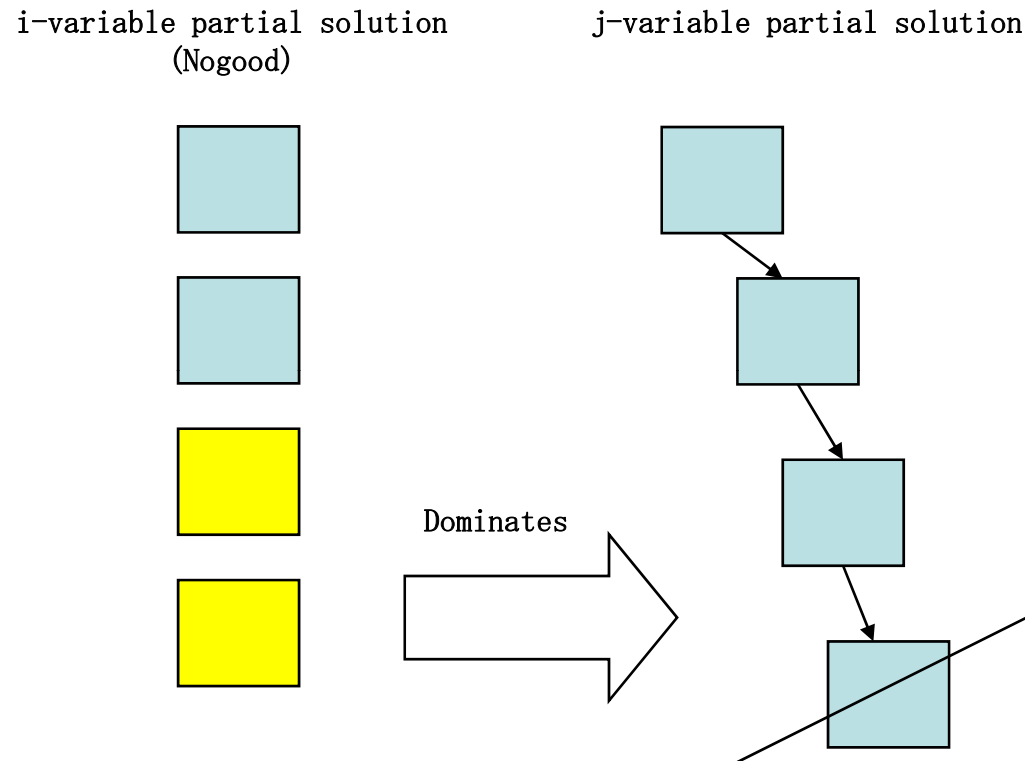
j-variable partial solution



Can we prune this search node?

- Extends Nogood to a j-variable partial solution which dominates current j-variable partial solution

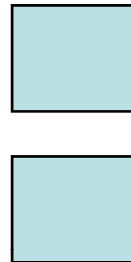
Focacci and Milano, 2002 – Nogood Extension



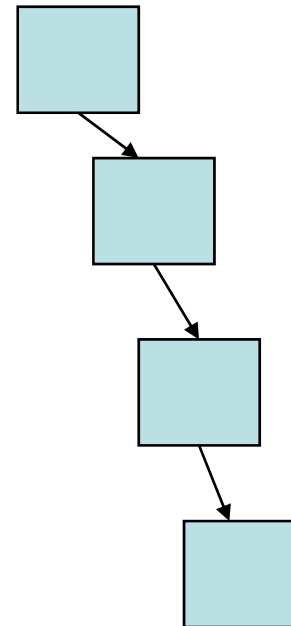
- Extends Nogood to a j-variable partial solution which dominates current j-variable partial colution

Our approach: Partial Solution Rearrangement

i -variable partial solution
(Nogood)



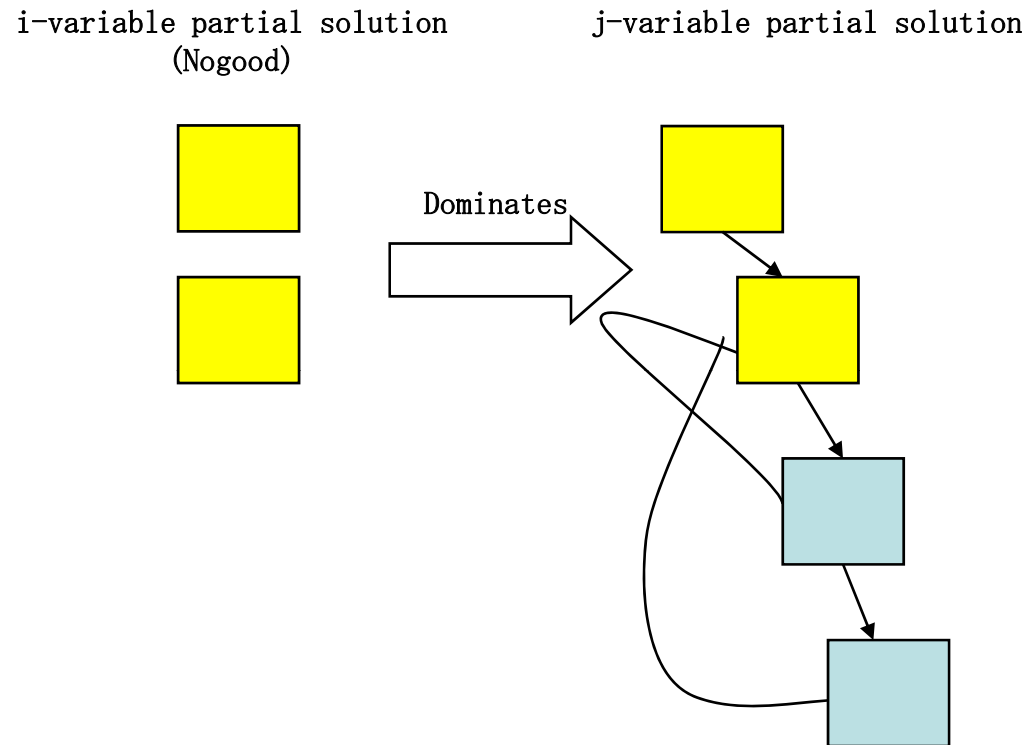
j -variable partial solution



Can we prune this search node?

- We rearrange the j -variable partial solution such that the i -variable subset is dominated by i -variable nogood

Our approach: Partial Solution Rearrangement



- We rearrange the j -variable partial solution such that the i -variable subset is dominated by i -variable nogood

Conclusion

- **Bin-completion + Symmetry detection**
 - Bin-oriented search space
 - SBDD-based Symmetry detection applies naturally in bin-oriented search tree structure
- **Symmetry detection for MKP highly effective (prunes > 99.9% of nodes in bin-completion search trees)**
- **Future Work: Same symmetry detection technique is applicable to other multicontainer packing/covering problems:**
 - Bin packing, bin covering, advertisement scheduling, N-dimensional vector packing.