Merging and Memorization in search trees : on the exact solution of scheduling problems

Lei Shang (PhD Student)
Pr. Vincent T'Kindt (Director)

Université François Rabelais Tours
Laboratoire d'Informatique (EA 6300)
Equipe ROOT (CNRS ERL 6305)

Outline

1. Problem: $1||\sum T_i$
2. Branch & Merge (theoretical guarantee)
3. Memorization (practical efficiency)
4. Extension : Branch & Memorize framework on sequencing problems

Problem: $1||\sum T_i$

- Jobset S, single machine, p_j=processing time, d_j=due date
- Objective: minimize the total tardiness $\sum_j \max(0, C_j - d_j)$
- NP-hard (ordinary sense)
- In theory (complexity):
  - Brute force $O(n!)$
  - Dynamic programming: $O^*(2^n)$ in time and space
  - Divide & Conquer: $O^*(4^n)$, polynomial space (Gurevich et al., 1987)
  - Branch & Reduce: $O^*(3^n)$, polynomial space (F. D. Croce et al, 2015)
  - Branch & Merge => $O^*((2+\epsilon)^n)$ in time and polynomial space

- In practice:
  - The B&B of Szwarc et al. => 500 jobs in 2001. (900 jobs today!)
  - Memorization => 1200 jobs.

In Theory

Objective

- Exact algorithms with worst-case running time/space guarantee ($O^*(c^n)$, with c a constant as small as possible)

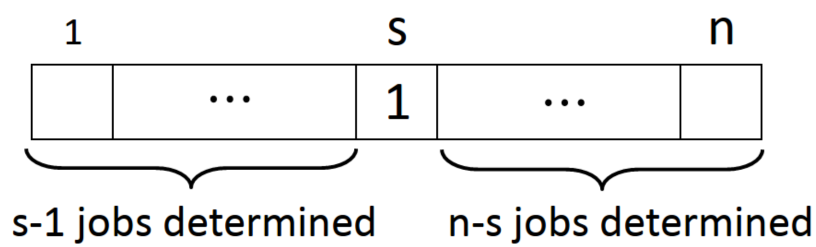
Notation

- LPT (Longest Processing Time first) job sequence: $(1,2,..,n)$
- EDD (Earliest Due Date first) job sequence: $(e_1, e_2, .., e_n)$

In Theory

Lawler's Property (1977)

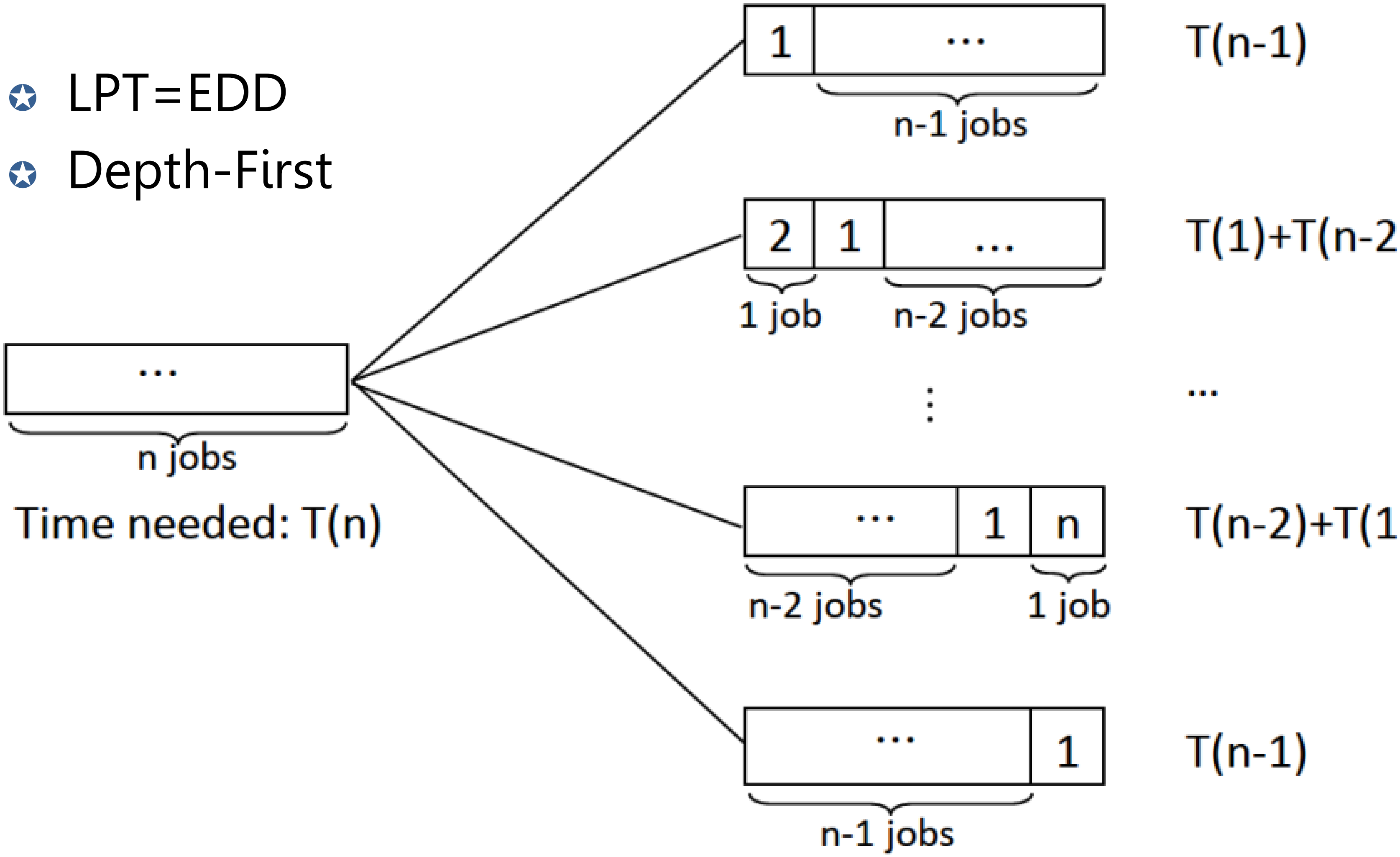
- Let job 1 $= e_h$, then job 1 can only be set in position $s \geq h$
- Jobs preceding 1 are: $B_1 = \{e_1, e_2, .., e_{h-1}, e_{h+1}, .., e_s\}$
- Jobs following 1 are: $A_1 = \{e_{s+1}, e_{s+2}, .., e_n\}$



- => Worst case: LPT=EDD

Branch & Reduce

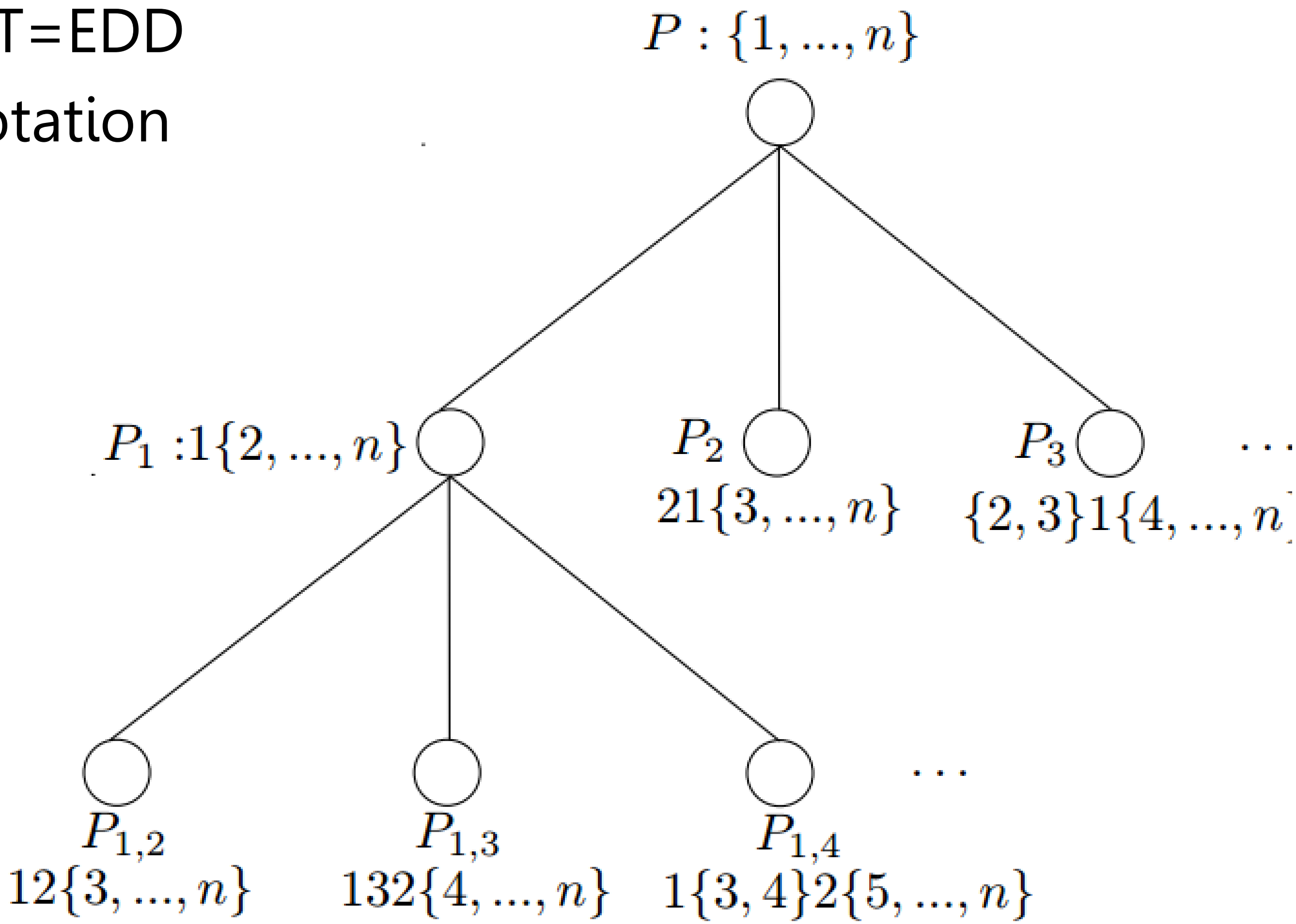
- LPT=EDD
- Depth-First



$$T(n) \leq 2\,T(n\text{-}1) + 2T(n\text{-}2) + \cdots + 2\,T(1) \Rightarrow T(n) = O(3^n)$$

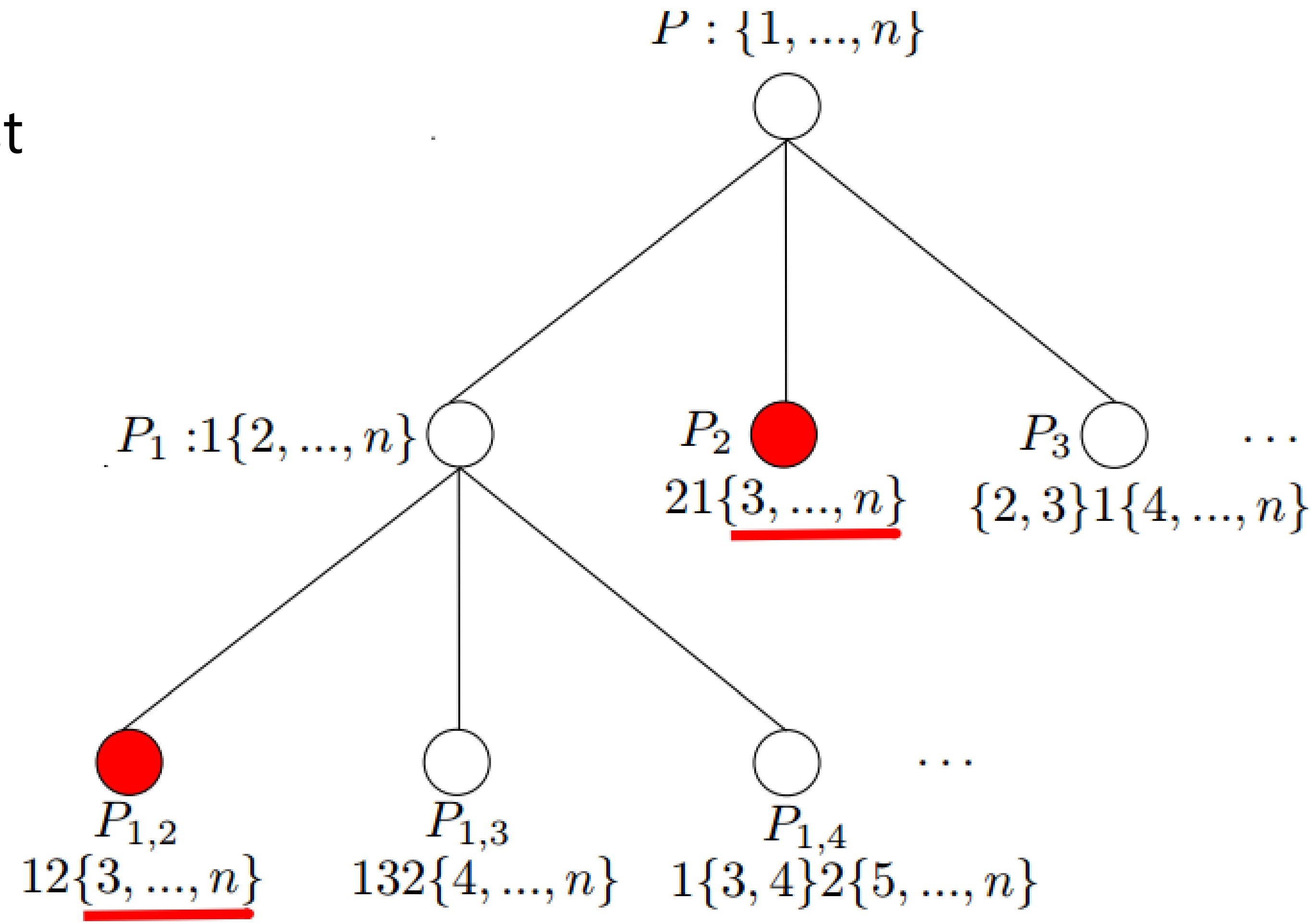
Branch & Reduce

- LPT=EDD
- Notation



Branch & Reduce: observations

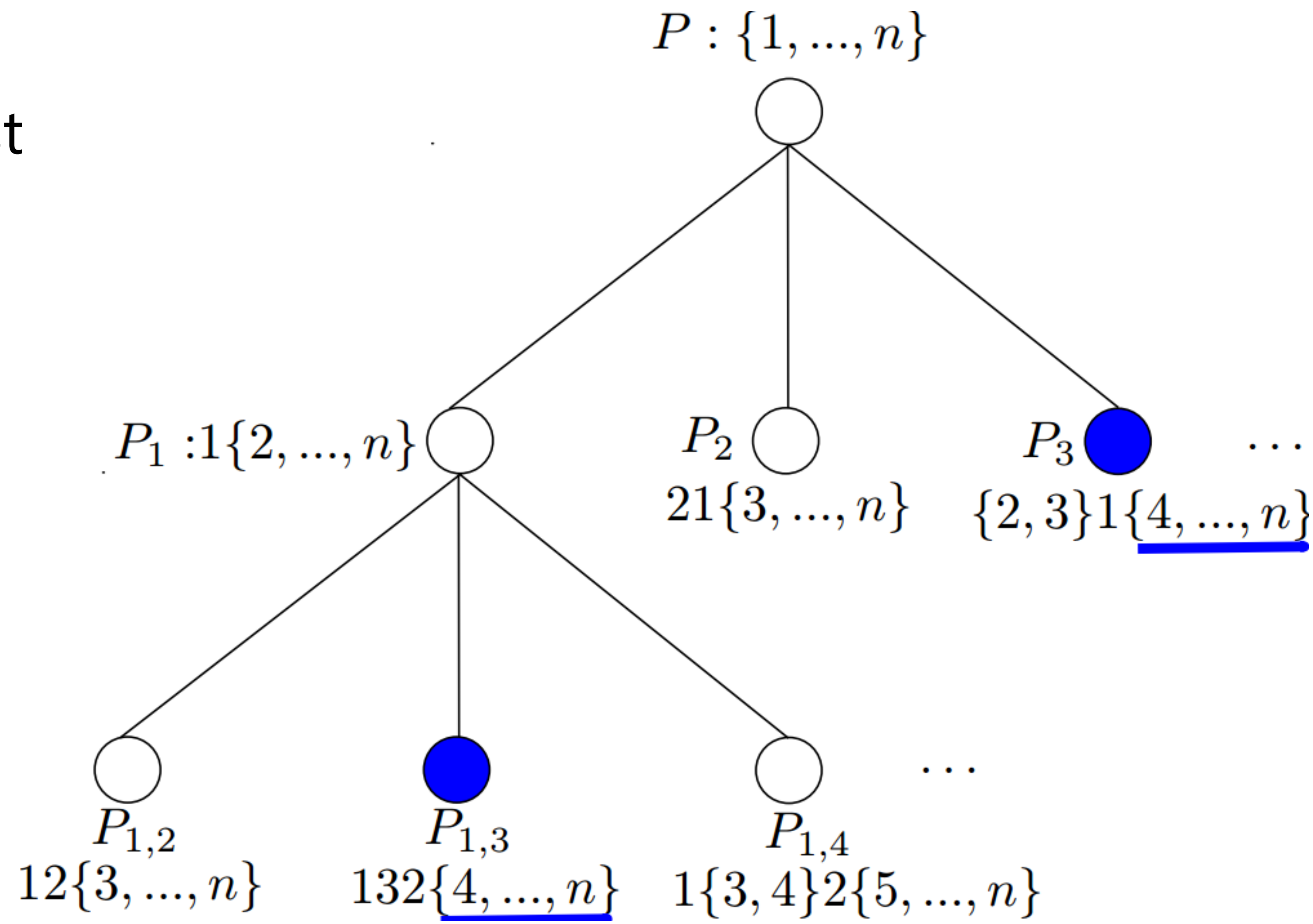
- LPT=EDD
- Depth-First



- **Some sub-problems are solved repeatedly!**

Branch & Reduce: observations

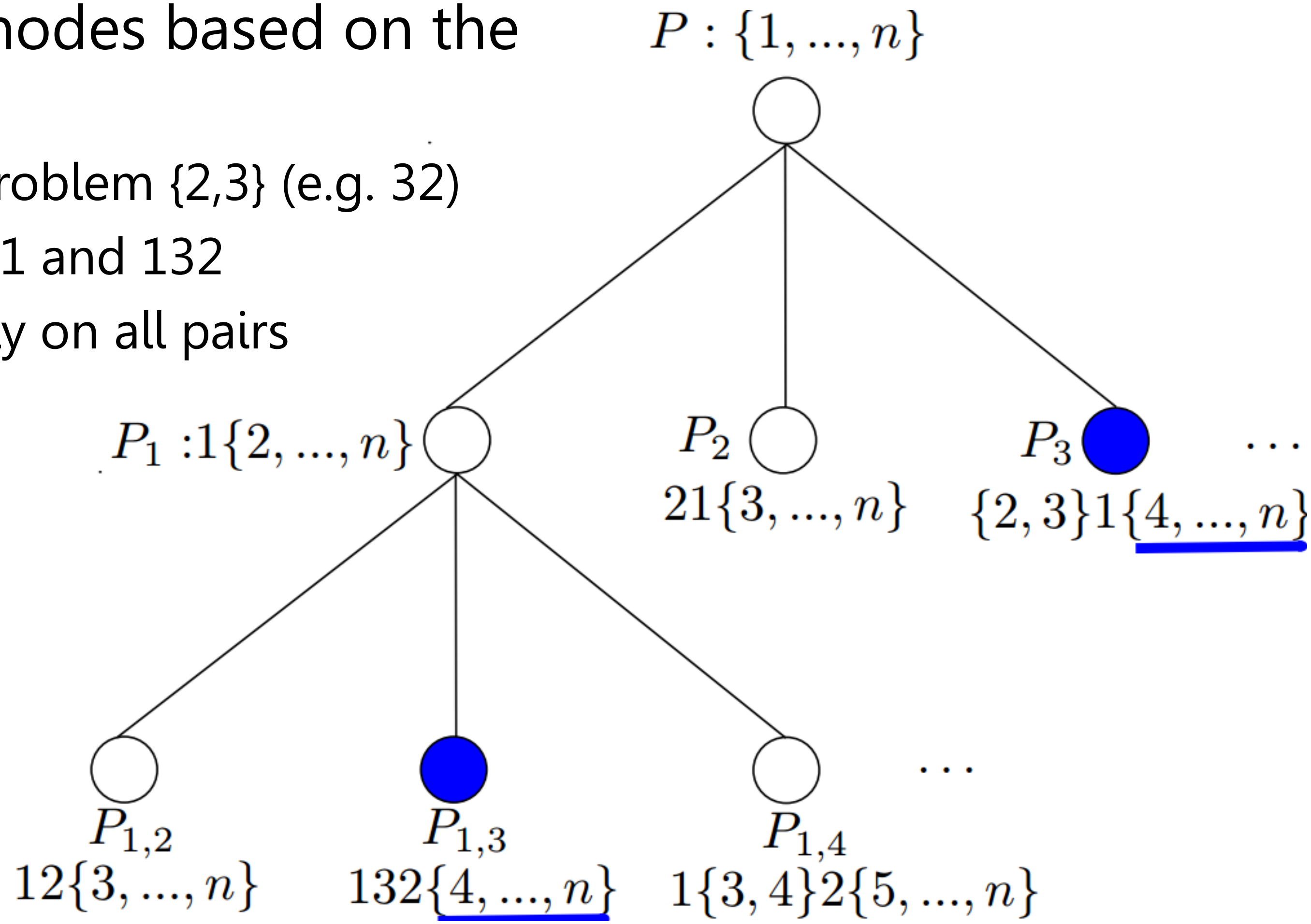
- LPT=EDD
- Depth-First



- **Some sub-problems are solved repeatedly!**

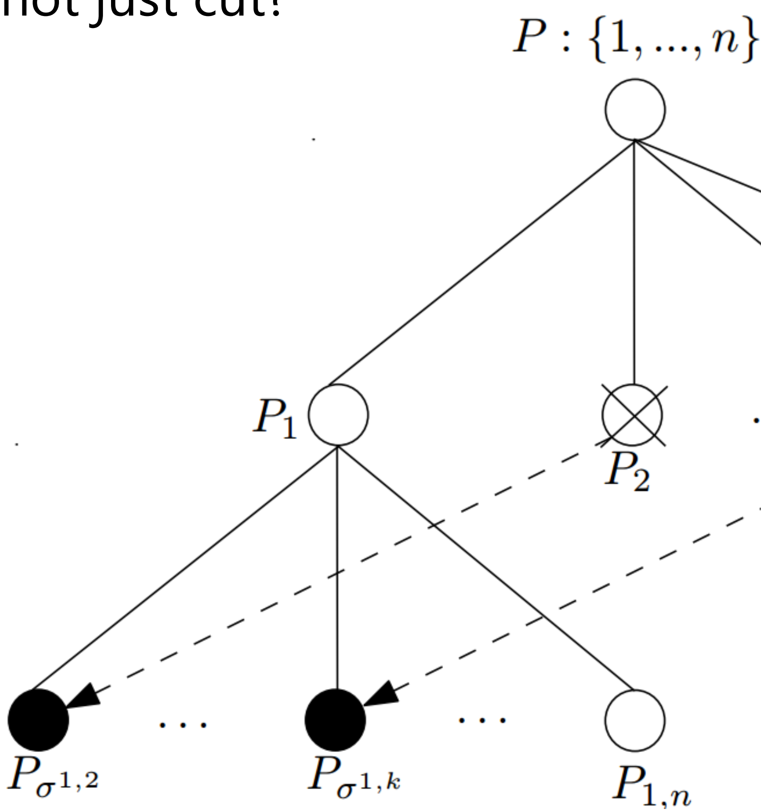
Branch & Merge (left)

- Idea: merge nodes based on the fixed part
  - Solve sub-problem {2,3} (e.g. 32)
  - Compare 321 and 132
  - Cannot apply on all pairs



Branch & Merge (left)

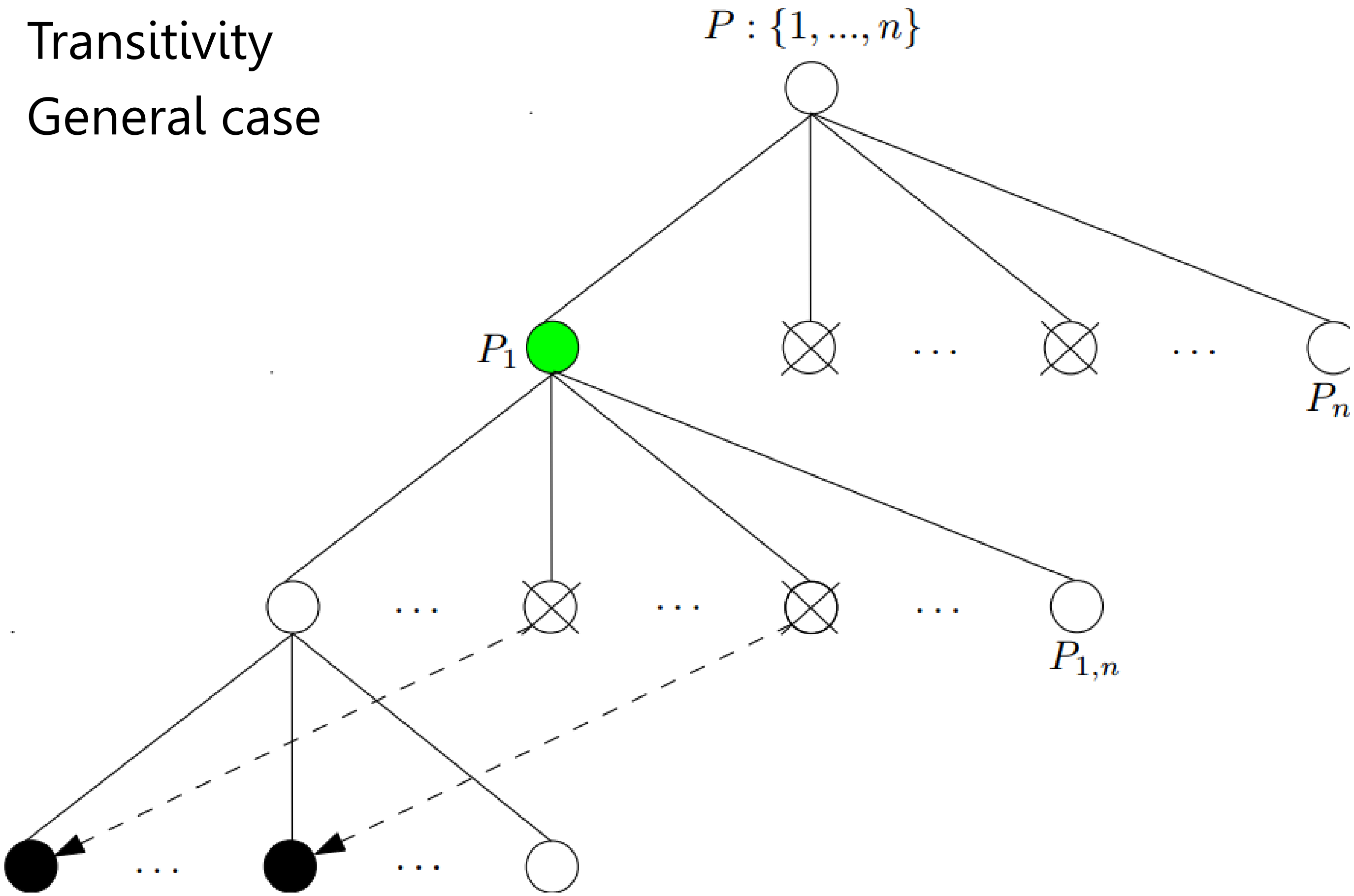
- Idea: merge identical nodes based on the fixed part
  - On first k nodes, k is a constant
  - Why not just cut?



P_1 :$1\{2, ..., n\}$
P_2 :$21\{3, ..., n\}$
P_k :$\{2..k\}1\{k+1..n\}$
$P_{\sigma 1,2}$:$\text{BEST}(12, 21)\{3..n\}$
$P_{\sigma 1,k}$:$\text{BEST}(\{2..k\}1, 1\{3..k\}2)\{k+1..n\}$

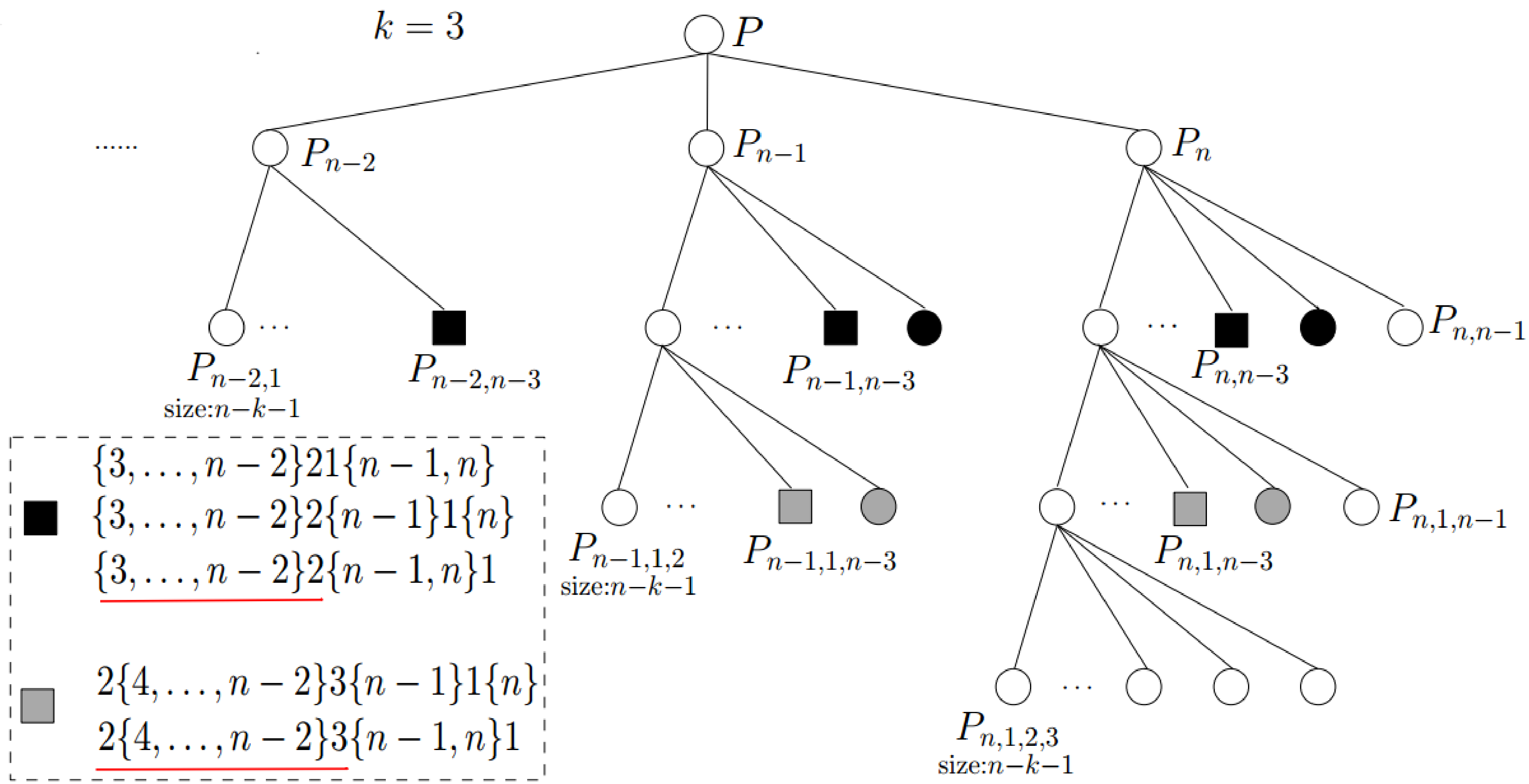
Branch & Merge (left)

- Idea: merge identical nodes based on the fixed part
  - On first k nodes, k is a constant
  - Transitivity
  - General case

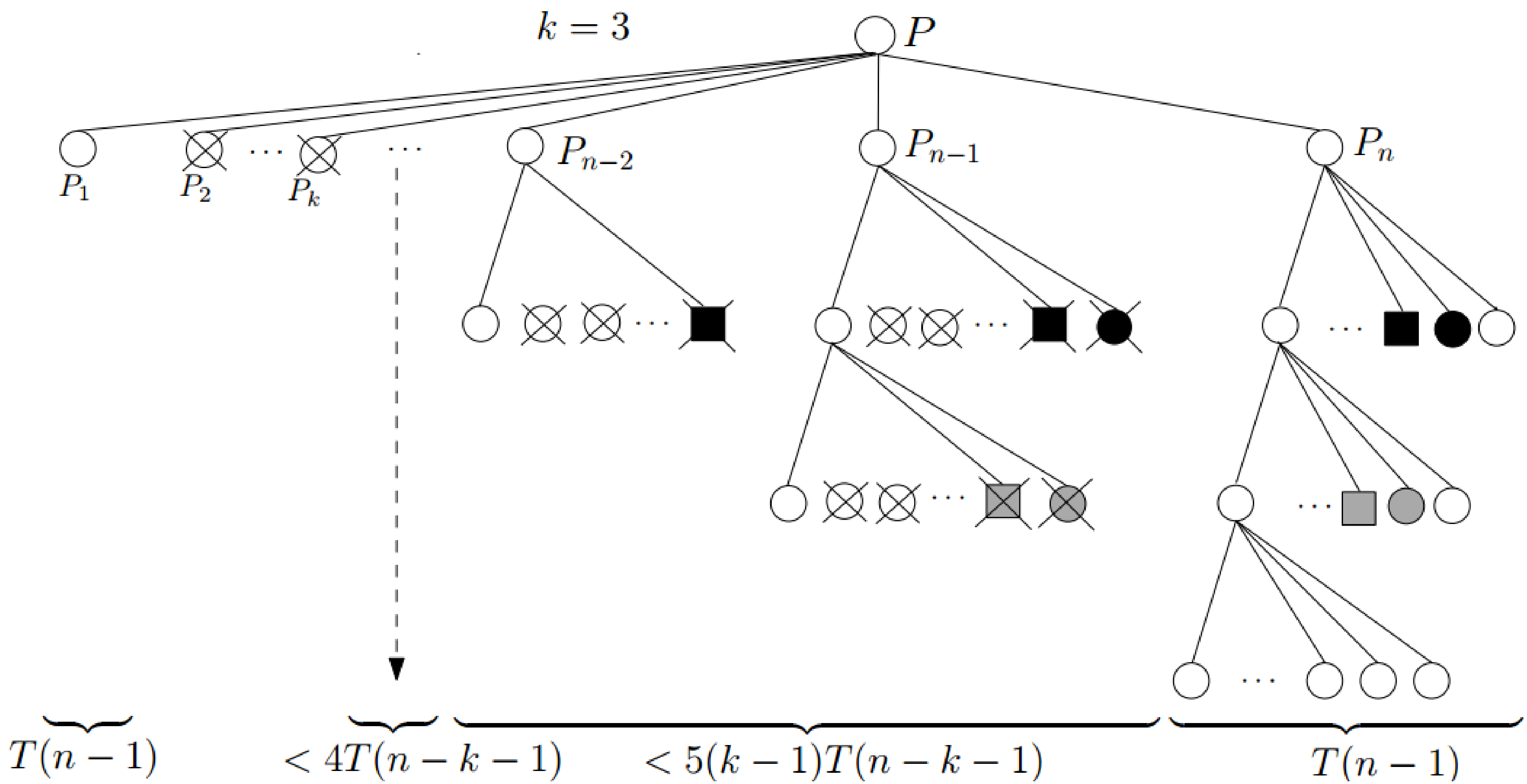


Branch & Merge (right)

- More complex…



Branch & Merge



- Recurrence: $T(n) \leq 2T(n-1) + (5k-1)T(n-k-1) + O(p(n))$

Branch & Merge

- $T(n)$ converges to $O^*(2^n)$. $T(n) = O^*(2.0367^n)$ when $k = 10$

k	$T(n)$
5	$\mathcal{O}^*(2.3065^n)$
10	$\mathcal{O}^*(2.0367^n)$
15	$\mathcal{O}^*(2.0022^n)$
20	$\mathcal{O}^*(2.0001^n)$

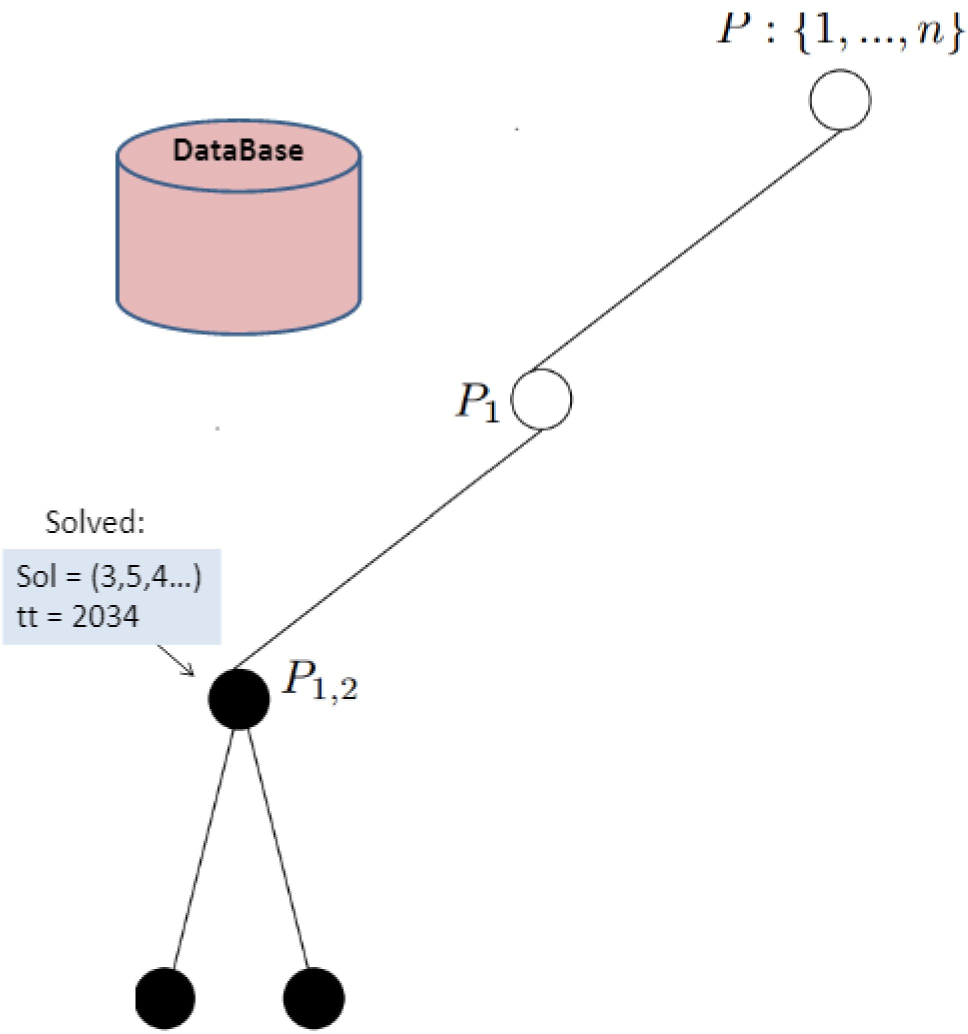
Summary 1

- Branch & Merge in ~ $O^*(2^n)$ time and polynomial space
- Can be generalized to other problems: branch smartly
- Work done together with:
  - Federico Della Croce
  - Vincent T'Kindt
  - Michele Garraffa

In Practice

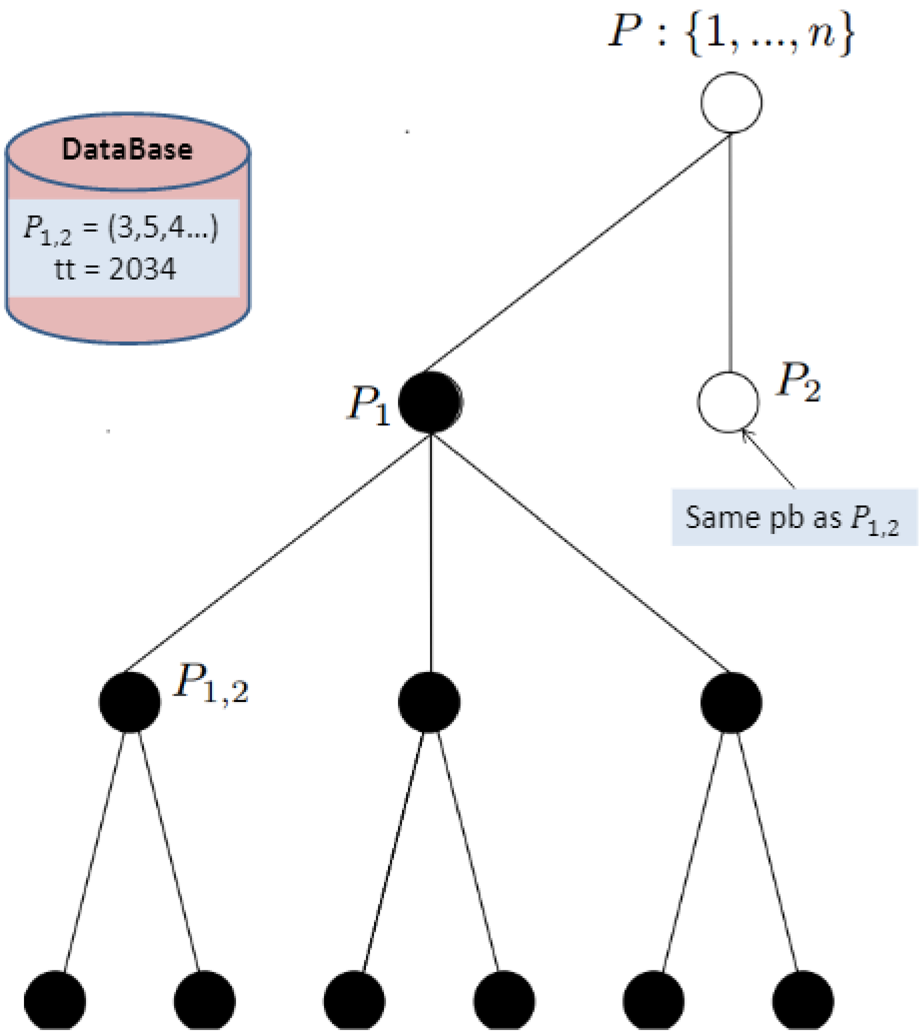
- BB2001: Szwarc et al. 2001
  - Solved 500 jobs in 2001 (900 jobs today!)
  - **Split**: decompose by precedence relations
  - **PosElim**: eliminates bad branching positions
  - **Memorization**: avoids solving a problem twice by storing its solution (basically merging without moving nodes)
- Without Split, PosElim
  - Branch & Merge is clearly more efficient than Branch & Reduce
- With Split, PosElim
  - Split & PosElim: break the structure of merging
- Memorization is more practical, even though theoretically exponential space.

In Practice: Memorization



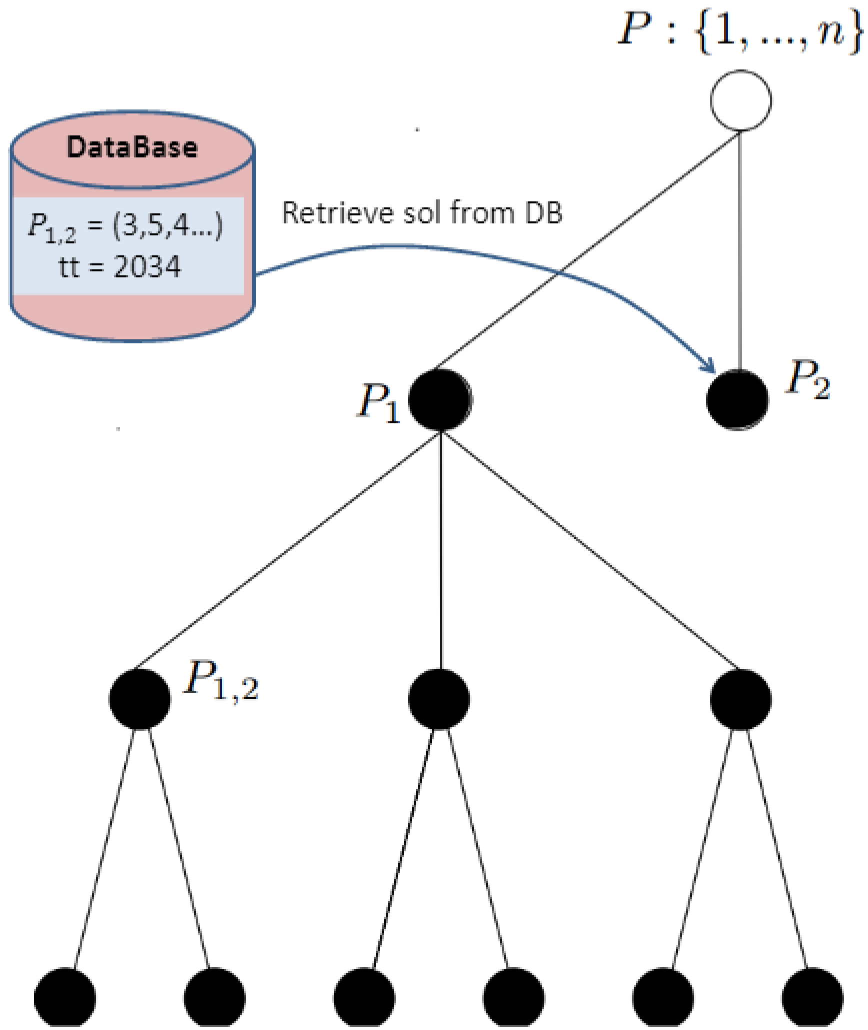
- « Never solve a problem twice »

In Practice: Memorization



- « Never solve a problem twice »

In Practice: Memorization



- « Never solve a problem twice »

The power of Memorization

- BB2001 of Szwarc et al. has no LB procedure:
  - Paradox (Szwarc et al. 2001): removal of LB evaluation drastically accelerate the solution.
  - => cut a sub-problem many times by computing LB is slower than solving it once and memorize the solution

- Can be further boosted!

Enhanced Paradox

- Enhanced Paradox (our work)
  - Removing **Split** from BB2001 drastically accelerate the solution
  - **Split** : decompose the problem by precedence relations

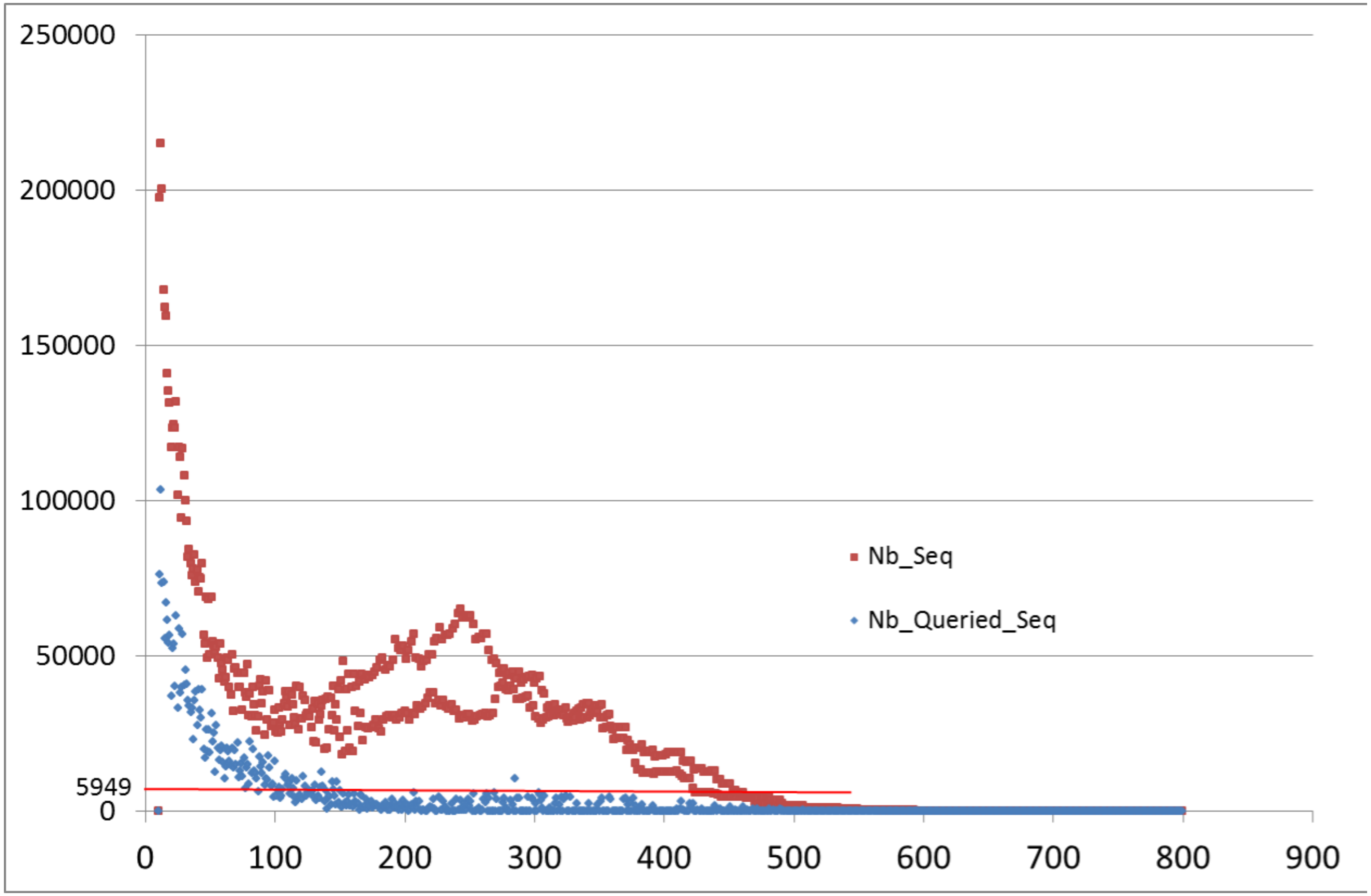
TMin (s)	TAvg (s)	TMax (s)	#Nodes	#Hit	#SolMem
0.0	192.81	2963.0	880268	227203	111175
0.0	8.0	114.0	3053648	899031	1262895

Table: Results for instances of size 700

- But…the memory is filled quickly (solve up to 700 jobs)

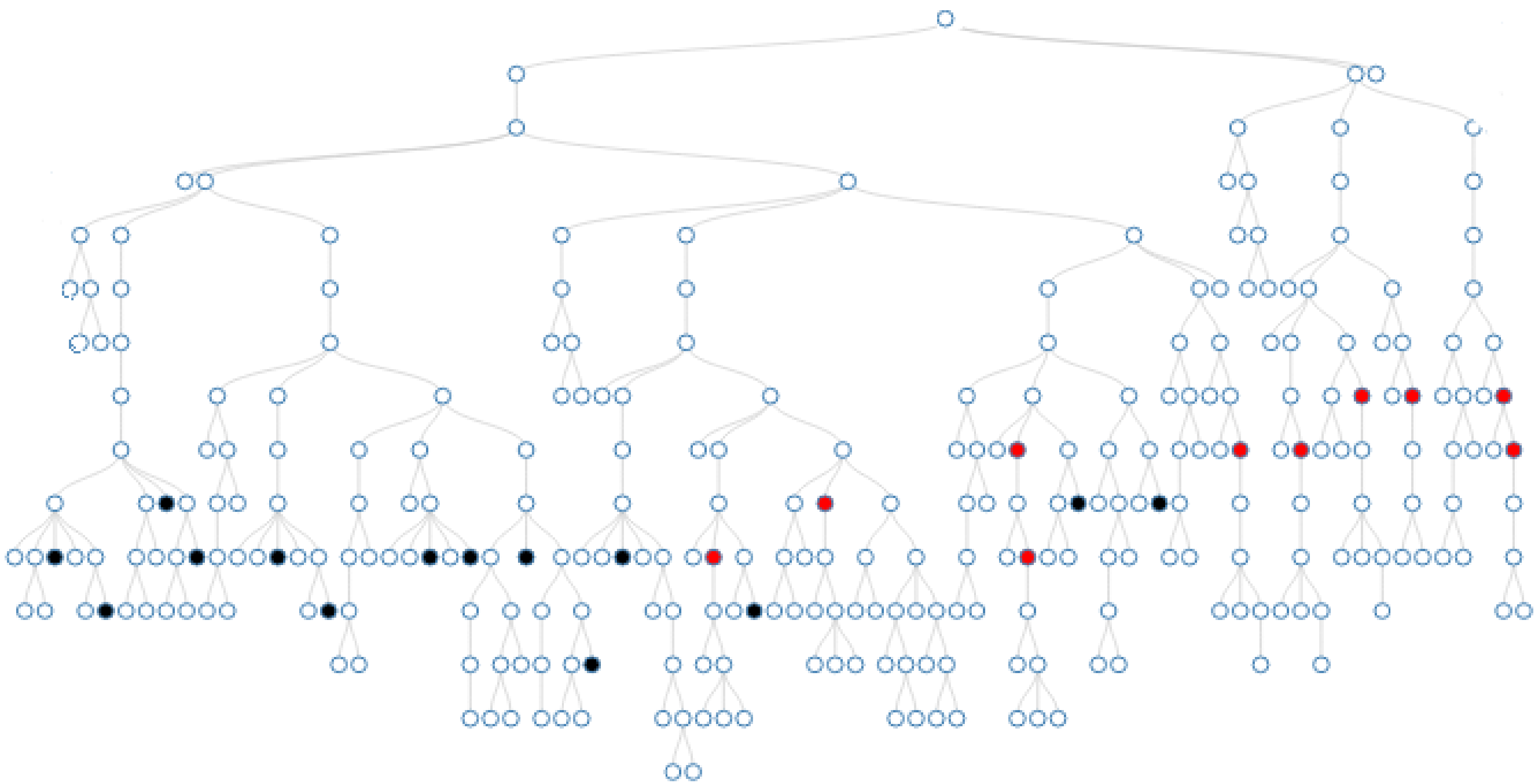
Memory Analysis

- Are all memorized solutions useful ?



Memory Analysis

- Are all memorized solutions useful ?



Memory Cleaning Strategies

- LUFO (Least Used First Out)
  - Attach a counter (`nbUsed`) to each solution
  - When a solution is used: `nbUsed=nbUsed+1`
  - Memory full: `nbUsed=nbUsed-1` for all solution, remove a solution if its `nbUsed<0`
- Also tested:
  - FIFO (First In First Out)
  - BEFO (Biggest Entry First Out)

	TMin	TAvg	TMax	#Nodes	SizeMem
FIFO-800	0.0	60.0	3144.0	16161758	1727397
BEFO-800	0.0	59.0	4828.0	6356245	2006948
LUFO-800	0.0	19.0	275.0	5408511	1354477
LUFO-1200	0.0	192.0	3763.0	28223765	1424612

Summary 2

- An enhanced paradox for $1||\sum T_i$
- An efficient memory cleaning strategy: LUFO
- Solve instances with up to 1200 jobs (from 900)
- Work done together with:
  - Federico Della Croce
  - Vincent T'Kindt

Further: a Branch & Memorize framework

- We have witnessed the power of Memorization
- Can be applied on other problems?
- Three problems are considered: $1|r_i|\sum C_i$, $1\left|\tilde{d}\right|\sum w_i C_i$, $F2||\sum C_i$
- Treated in T'Kindt et al. (2004).
  - Different search strategies are revisited
  - The so-called DP property is implemented
    - Consider two nodes: 123{4,..,n} vs 132{4,..,n}
    - If 123 dominates 132, then the second node should be cut
    - Use memory to store the prefixed part

Further: a Branch & Memorize framework

A framework: different ways of doing Memorization:

- Solution Memorization ($1||\sum T_i$)
  - Depth-first

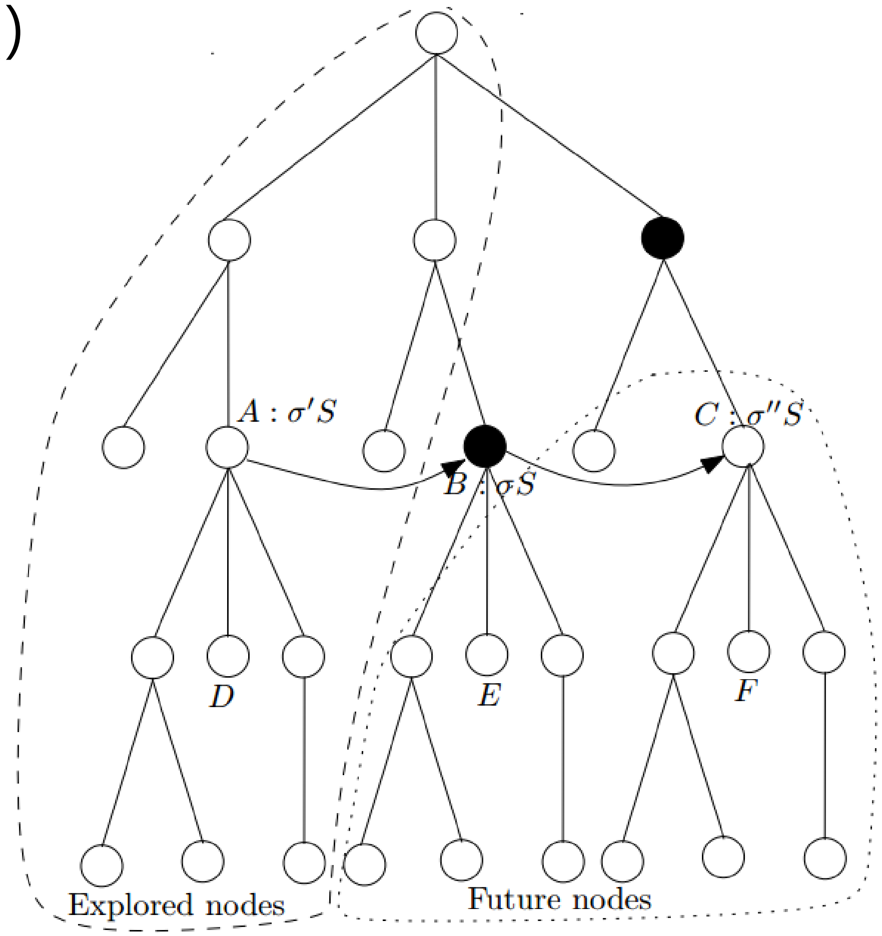


Figure 1: Solution Memorization

Further: a Branch & Memorize framework

A framework: different ways of doing Memorization:

- Passive Node Memorization
  - Memorize the current best solution for the fixed part given by branching
  - Used for cutting
  - Consider σ' dominates σ and σ'' (breadth-first)

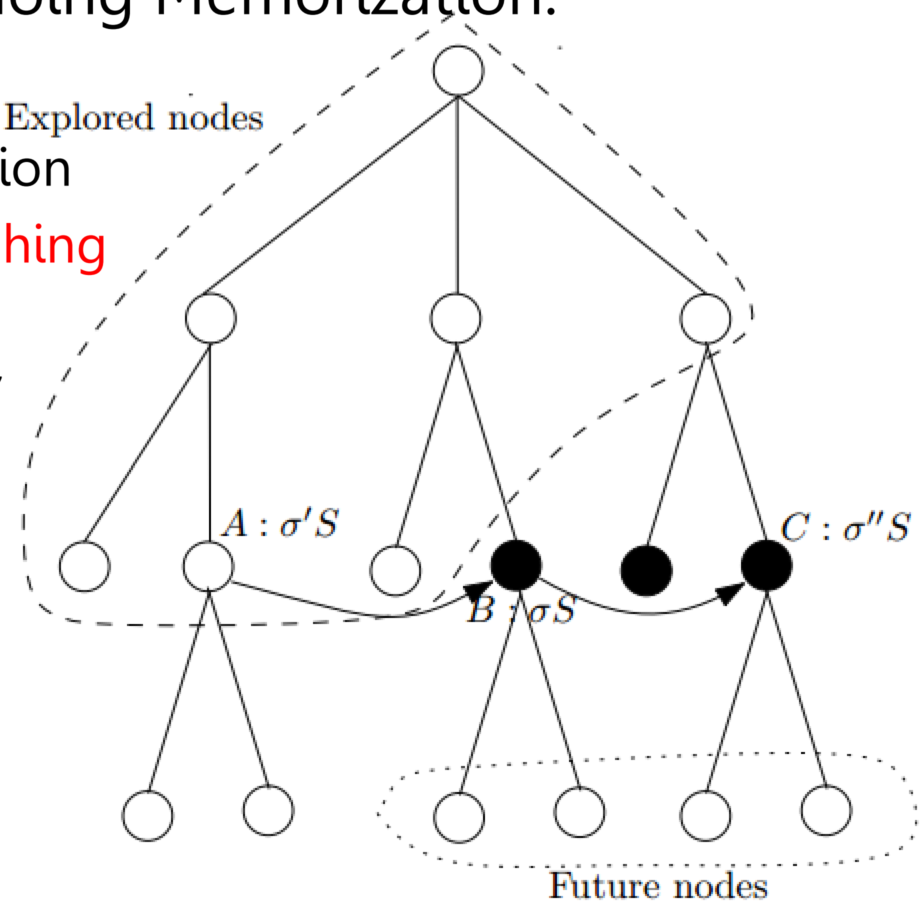


Figure 2: Passive node memorization)

Further: a Branch & Memorize framework

Different ways of doing Memorization:

- Predictive Node Memorization
  - Memorize the current best solution for the fixed part given by active search
  - Passive Node Memo + Local search
    - Dominance Rules Relying on Scheduled Jobs (Jouglet et al. 2004)
  - Used for cutting
  - Consider π dominates σ and σ''

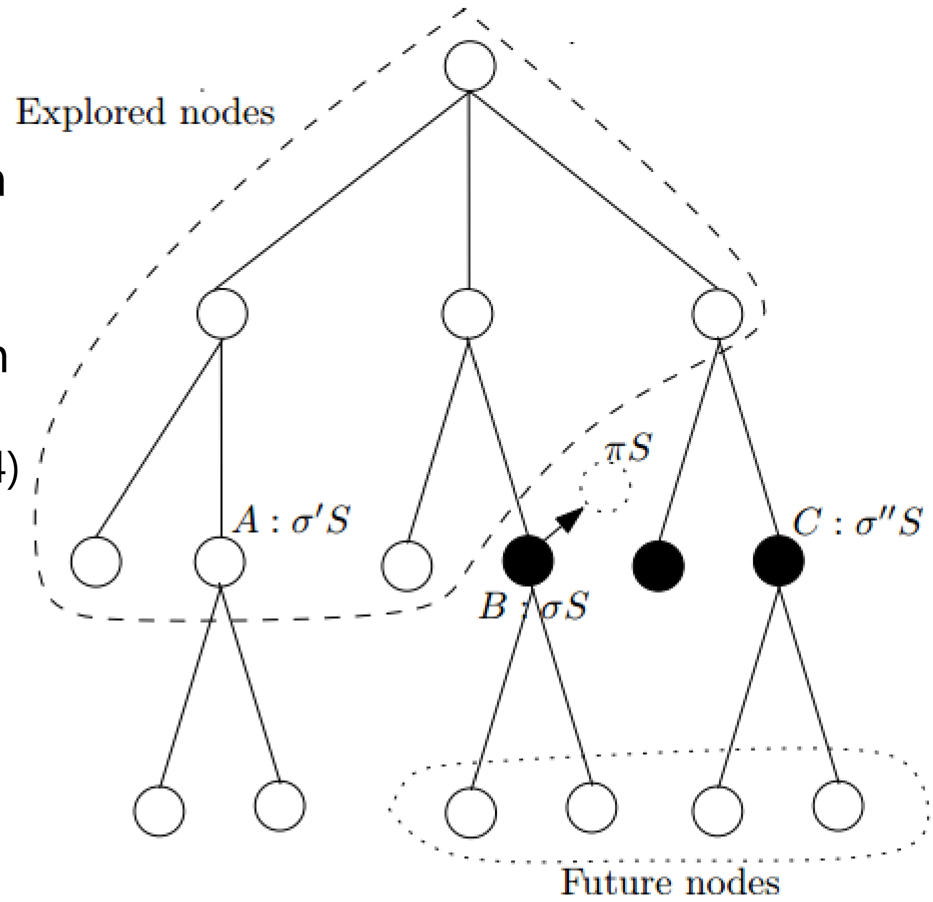


Figure 3: Predictive node memorization

Choose the right Memo scheme

Given a branching algorithm, choose a Memorization scheme

- Branching scheme
- Search strategy
- Other properties: whether « Decomposable »…

Choose the right Memo scheme

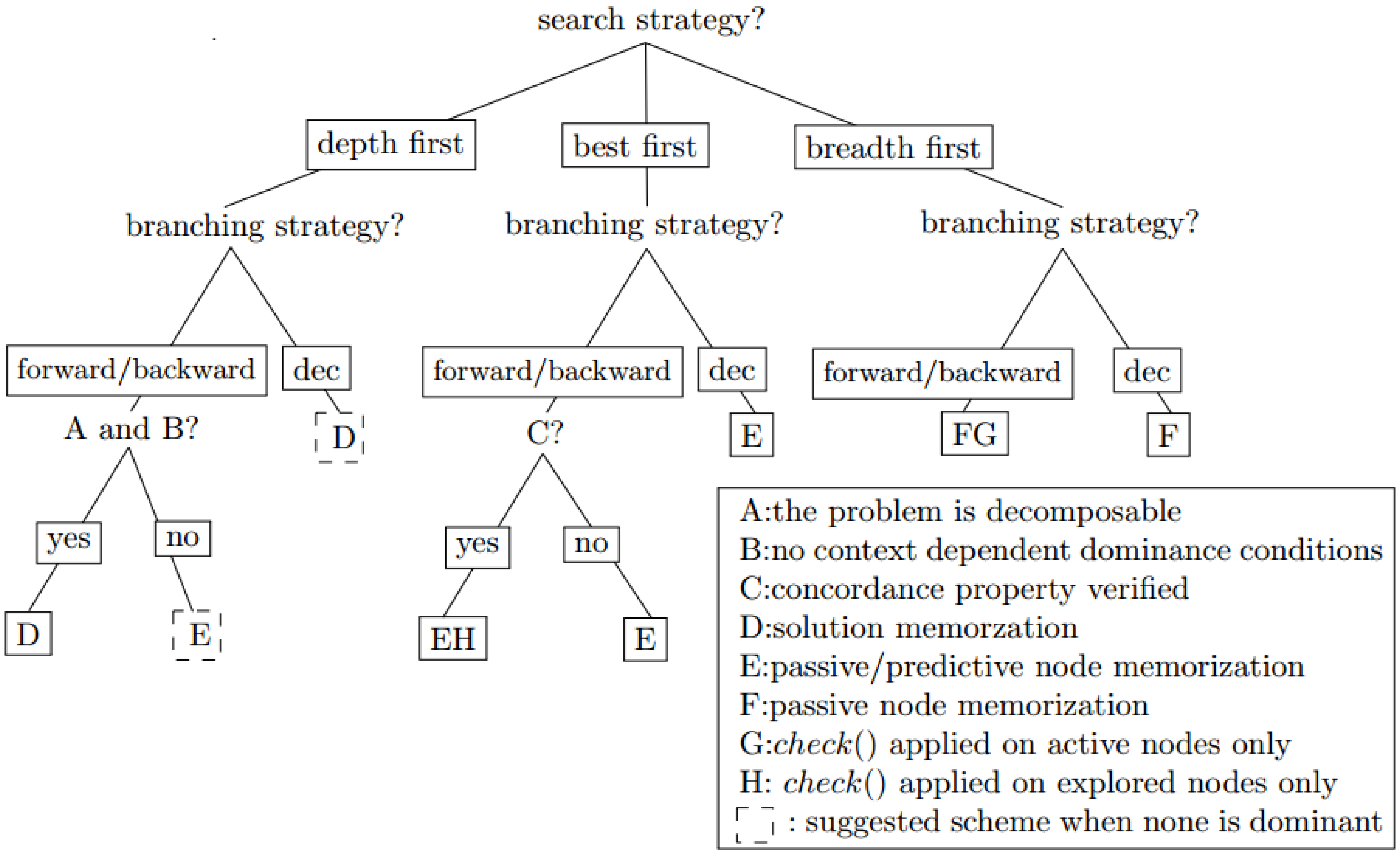


Figure 4: Decision tree for choosing the memorization scheme

Further: a Branch & Memorize framework

- The evidence of the power of memorization

Problem	Largest instances solved		Features of the best algorithm with memorization	Best in literature?
	Without memorization	With memorization		
$1\|r_i\|\sum C_i$	80 jobs	130 jobs	depth first+ *predictive node memorization*	yes
$1\|\tilde{d}_i\|\sum w_iC_i$	40 jobs	130 jobs	breadth first+ *passive node memorization*	yes
$F2\|\|\sum C_i$	30 jobs	40 jobs	best first+ *passive node memorization*	no
$1\|\|\sum T_i$	300 jobs	1200 jobs	depth first+ *solution memorization*	yes

Conclusion

- Part 3: work done together with:
  - Federico Della Croce
  - Vincent T'Kindt

- For theoretical guarantee: branch smartly and Merge !
- For practical efficiency: Branch & Memorize
  - Memorization is a powerful technique for scheduling problems
  - Should be considered as an essential building block of branching algorithms
  - The choice of branching scheme and search strategy are important


THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
DANKSCHEEN
BIYAN
SHUKRIA
TINGKI
YAQHANYELAY
TASHAKKUR ATU
SUKSAMA
EKHMET
GRAZIE
MEHRBANI
PALDIES
MAAKE
KOMAPSUMNIDA
GOZAIMASHITA
EFCHARISTO
JUSPAXAR
BOLZÏN
MERCI
SPASSIBO
SNACHALHUYA
NUHUN
CHALTU
WABEEJA
MAITEKA
YUSPAGARATAM
HUI
DHANYABAAD
ATTO
SPASIBO
DENKAUJA
UNALCHEESH
GUI
HATUR
EKOJU
SIKOMO
MAKETAI
MINMONCHAR
MERASTAWHY
GAEJTHO
SANCO
LAH
AGUYJE
FAKAAUE
TAVTAPUCH
MEDAWAGSE
BAIKA