前回の続き

【組合せ最適化問題の解法(1)】 まずこれらの方法で試す

●腕ずくの方法(brute force method):

コンピュータの高速な計算力に頼って、**起こりうる全ての場合を調べて**、
その中で最も良いものを選ぶ方法。(列挙法・全数探索とも呼ばれる)
〔特徴〕
◎ 原理が単純でわかりやすい
◎ 実行が可能であるならば、必ず最適解が見つかる
(組合せ最適化問題の解候補は有限な離散集合なので)
小規模問題では実用的
× 場合の数が組合せ爆発を起こす場合は実行不能

●欲張り法(greedy method):

各選択の時点で、**目先の利益が最大**になるよう選び続ける方法
〔特徴〕
◎ 原理が単純で分かりやすい
× 局所的な目先の最適化を行い続けることが全体の最適化になるとは
一般にはいえない…最適解が見つかる保障は一般にはない
◎ 多くの場合、最適ではないがそれに近いものが求まる
◎ グラフ最短経路探索問題など一部の問題では、最適解が得られる
(ダイクストラの方法) また、後述の動的計画法とも関連

【例】 ナップサック問題に対する欲張り法

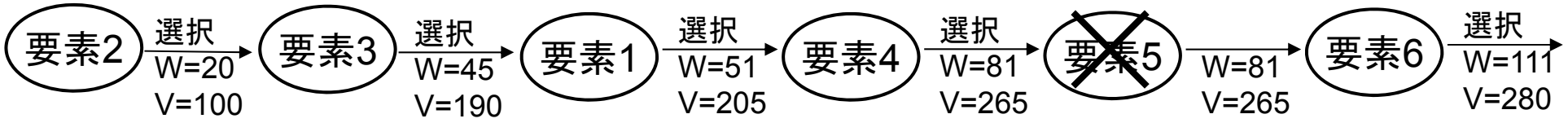
ナップサックの容量： 112

要素	1	2	3	4	5	6	7	8
重さ	6	20	25	30	40	30	60	10
価値	15	100	90	60	40	15	10	3
単位重さあたりの価値	2.5	5	3.6	2	1	0.5	0.166	0.3

（1）要素を**単位重さあたりの価値の高い順に並べ替える**。 i ＝1 とする。

（2）もし i 番目の要素を選択したら、選択した要素全体の重さがナップサック容量を超えない場合はその i 番目の要素を選択する。さもなければその要素を選択しない。

（3）i が要素数 n と等しくなったら終了。さもなければ i を i+1 として（2）へもどる。

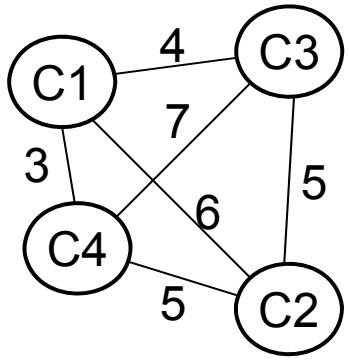


要素数10, 容量100, 価値上限100の問題で欲張り法により最適解を得る割合16～17%　だが最適解と欲張り法の解の違いは1割程度

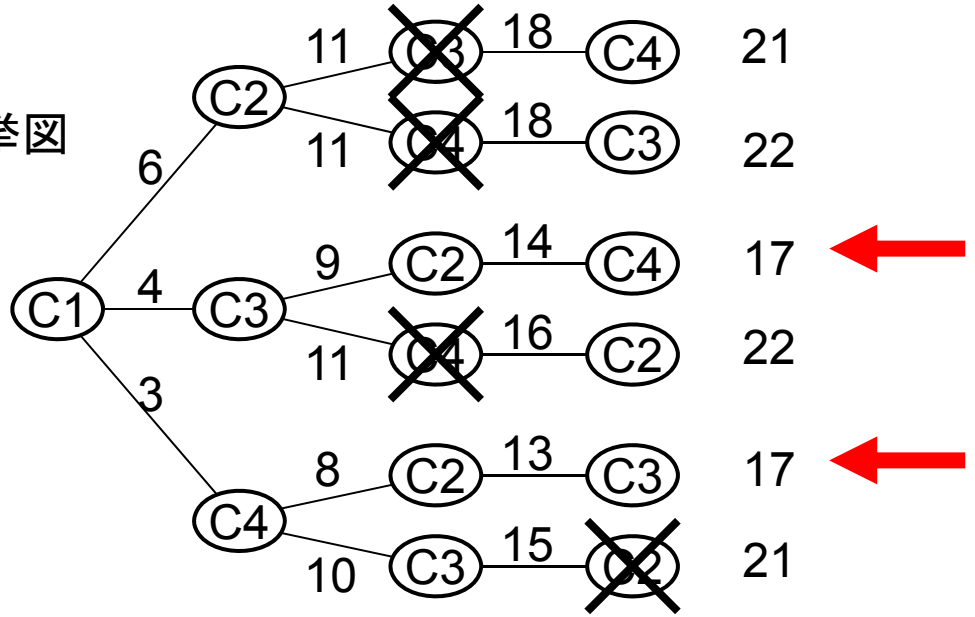
【組合せ最適化の解法(2)】　分枝限定法

全ての解候補を列挙して，その中で評価値最大(最小)のものを見出す：腕ずくの方法

例） 4都市TSP



列挙図



問題の規模が大きくなると、組合せ爆発のため実行不能！そこで…

→ **列挙図の木の生成途中で、**

分枝限定法 (branch and bound method)

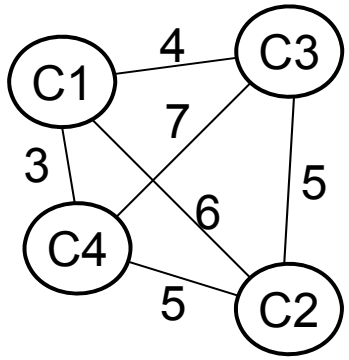
◎…「腕ずく法」と「欲張り法」の中間的なバランスの良い方法

△…**解の可能性が無い枝だけを刈れば最適解が保障**されるが、
大規模問題では実行不能になるので、最適解をあきらめて可能性の小さい枝を刈ることが多い

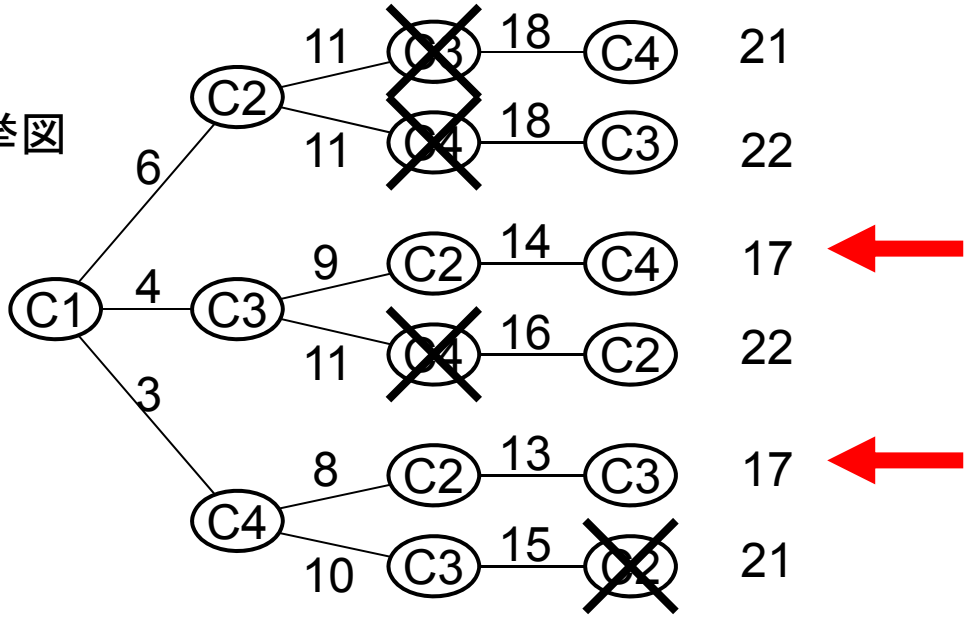
【組合せ最適化の解法(2)】　分枝限定法

全ての解候補を列挙して，その中で評価値最大(最小)のものを見出す：腕ずくの方法

例） 4都市TSP



列挙図



問題の規模が大きくなると、組合せ爆発のため実行不能！そこで…

→ **列挙図の木の生成途中で、解となる可能性の無い(または小さい)枝を刈る**

分枝限定法 (branch and bound method)

◎…「腕ずく法」と「欲張り法」の中間的なバランスの良い方法

△…**解の可能性が無い枝だけを刈れば最適解が保障**されるが、
大規模問題では実行不能になるので、最適解をあきらめて可能性の小さい枝を刈ることが多い

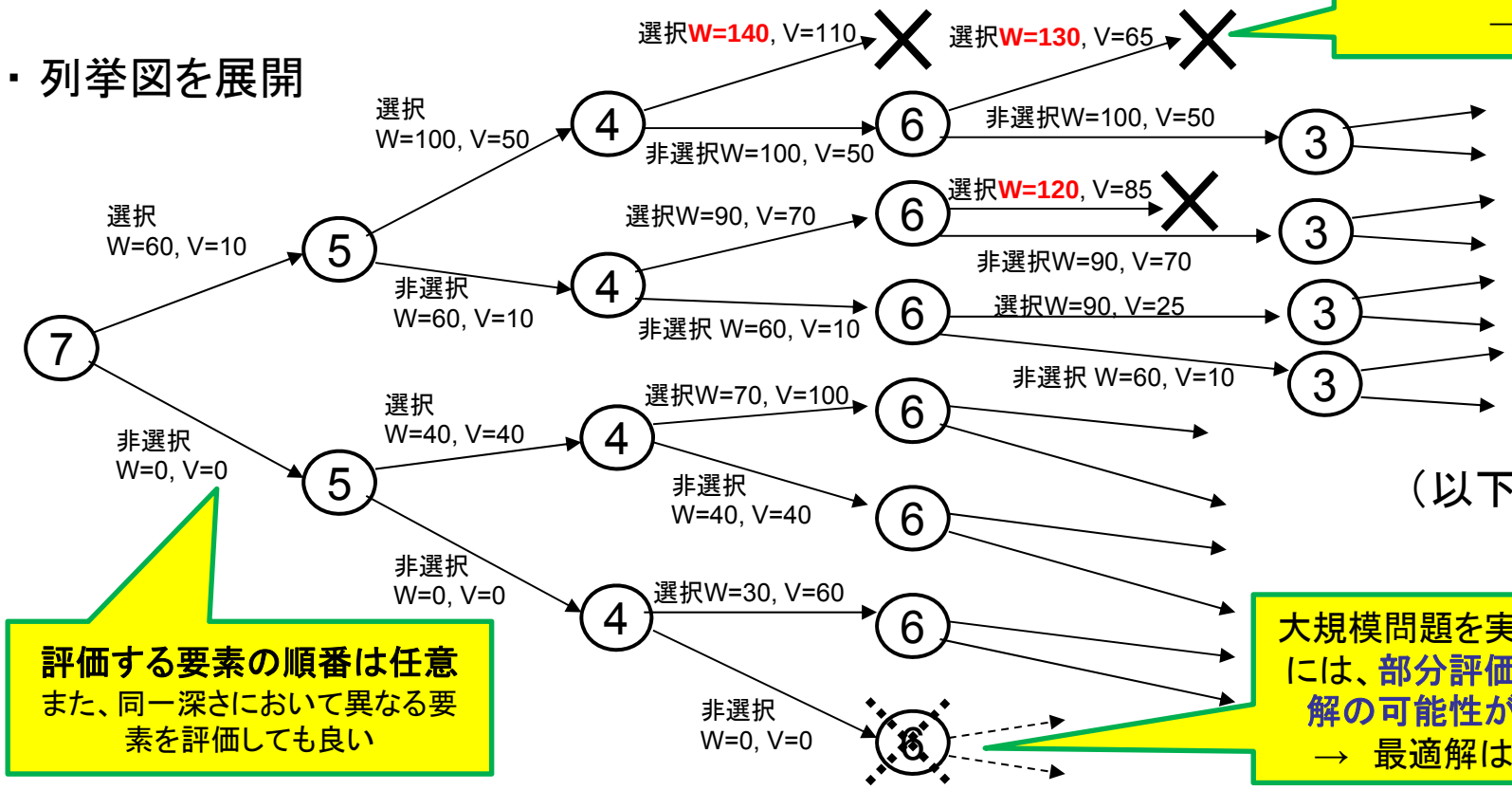
【例】 ナップサック問題に対する分枝限定法

ナップサックの容量W_{max} = 112

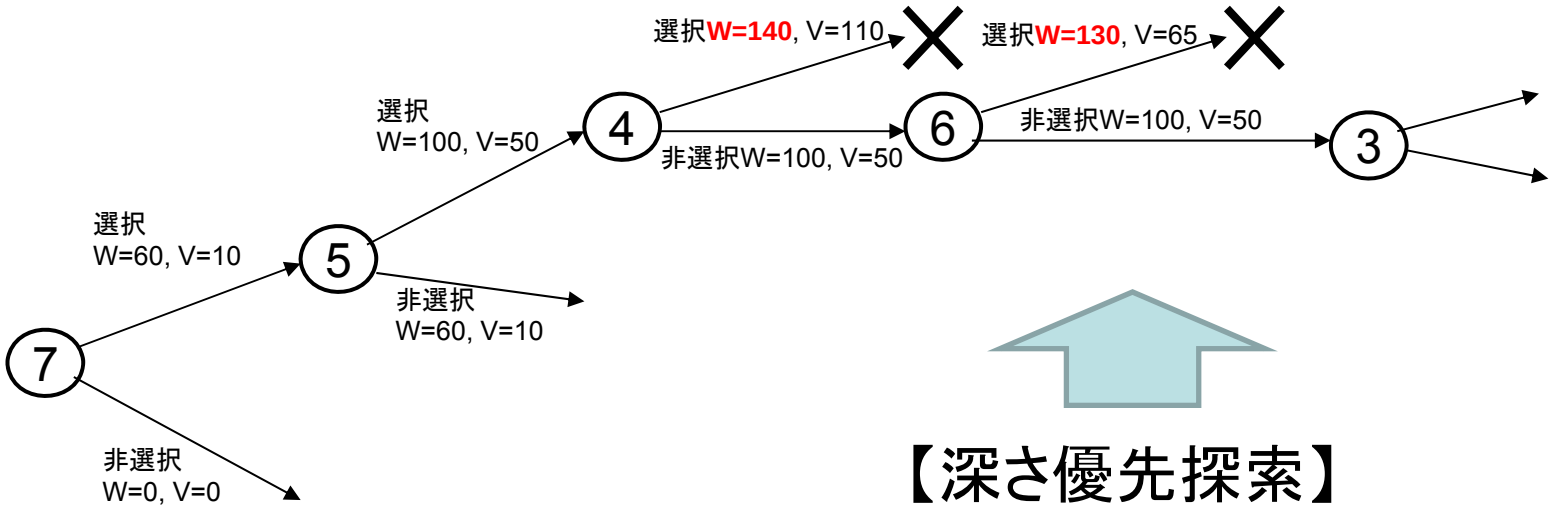
要素	1	2	3	4	5	6	7	8
重さ	6	20	25	30	40	30	60	10
価値	15	100	90	60	40	15	10	3

・腕づくの方法（全数探索）の場合、検討する解候補数は2^8＝256

・列挙図を展開



分枝限定法における枝の展開:「深さ優先探索」と「幅優先探索」



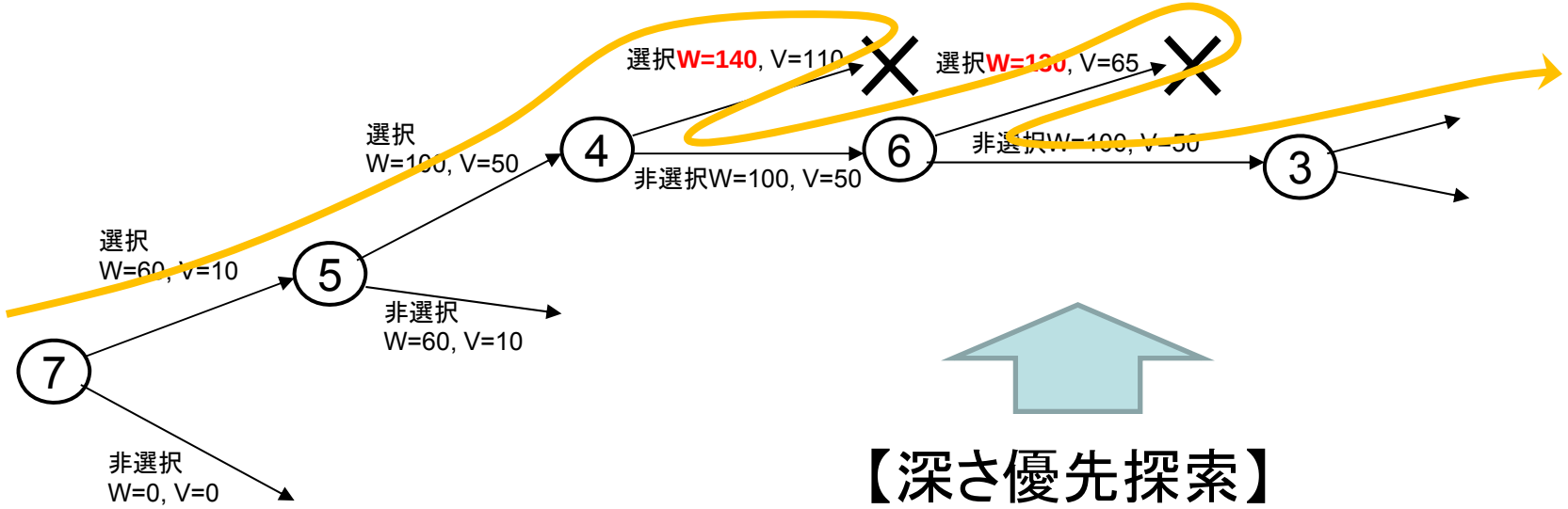
【深さ優先探索】

探索の各時点で、最大の深さの部分問題から優先的に探索する

特徴

- ○ 実行可能解を素早く見つける
- ○ 探索の過程で必要となる記憶容量は小さい

分枝限定法における枝の展開:「深さ優先探索」と「幅優先探索」



【深さ優先探索】

探索の各時点で、最大の深さの部分問題から優先的に探索する

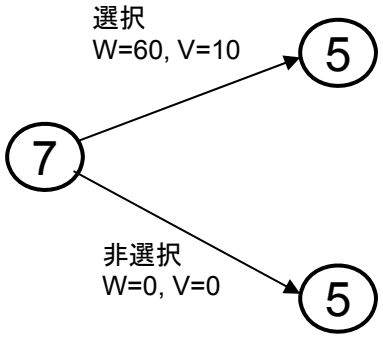
特徴

○ **実行可能解**を素早く見つける

○ 探索の過程で必要となる記憶容量は小さい

分枝限定法における枝の展開:「深さ優先探索」と「幅優先探索」

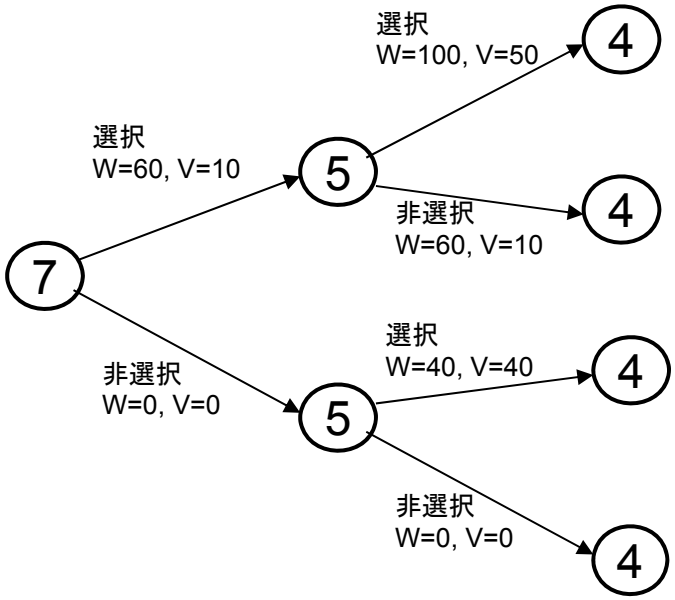
【幅優先探索】



探索の各時点で、
深さが最小の部分問題
から優先的に探索する

分枝限定法における枝の展開:「深さ優先探索」と「幅優先探索」

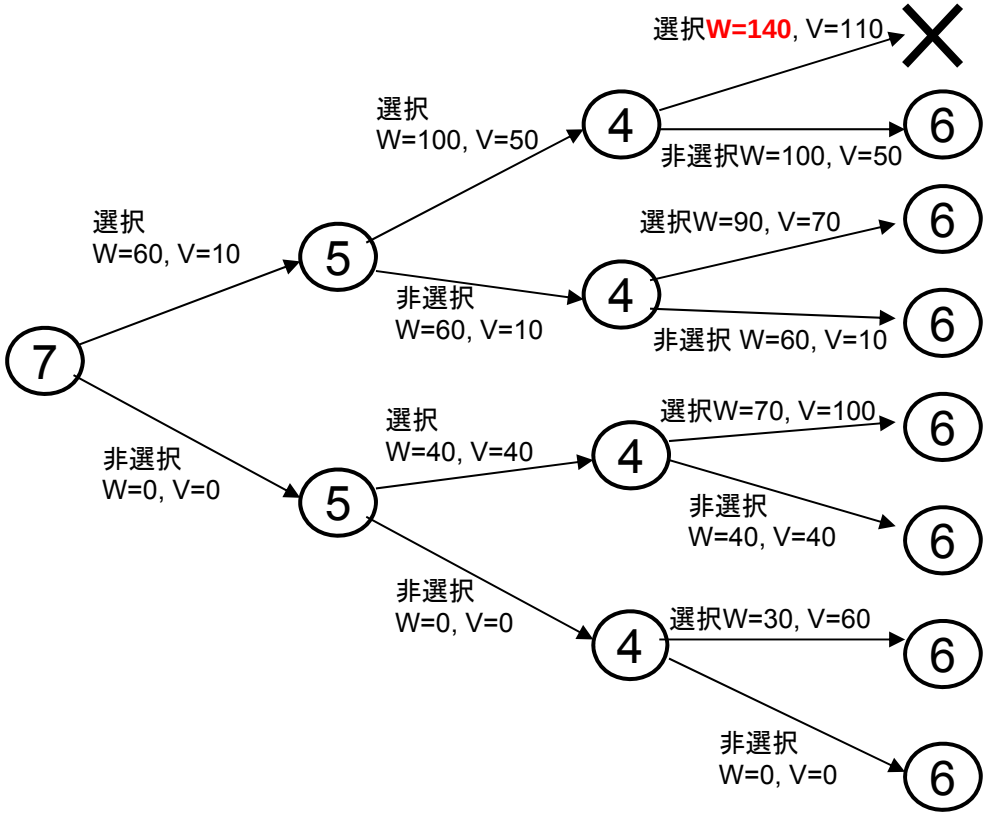
【幅優先探索】



探索の各時点で、
深さが最小の部分問題
から優先的に探索する

分枝限定法における枝の展開:「深さ優先探索」と「幅優先探索」

【幅優先探索】



探索の各時点で、
深さが最小の部分問題
から優先的に探索する

経路探索アルゴリズム「ダイクストラ法」は、
「深さ＝コスト」とした幅優先探索の一種でもある

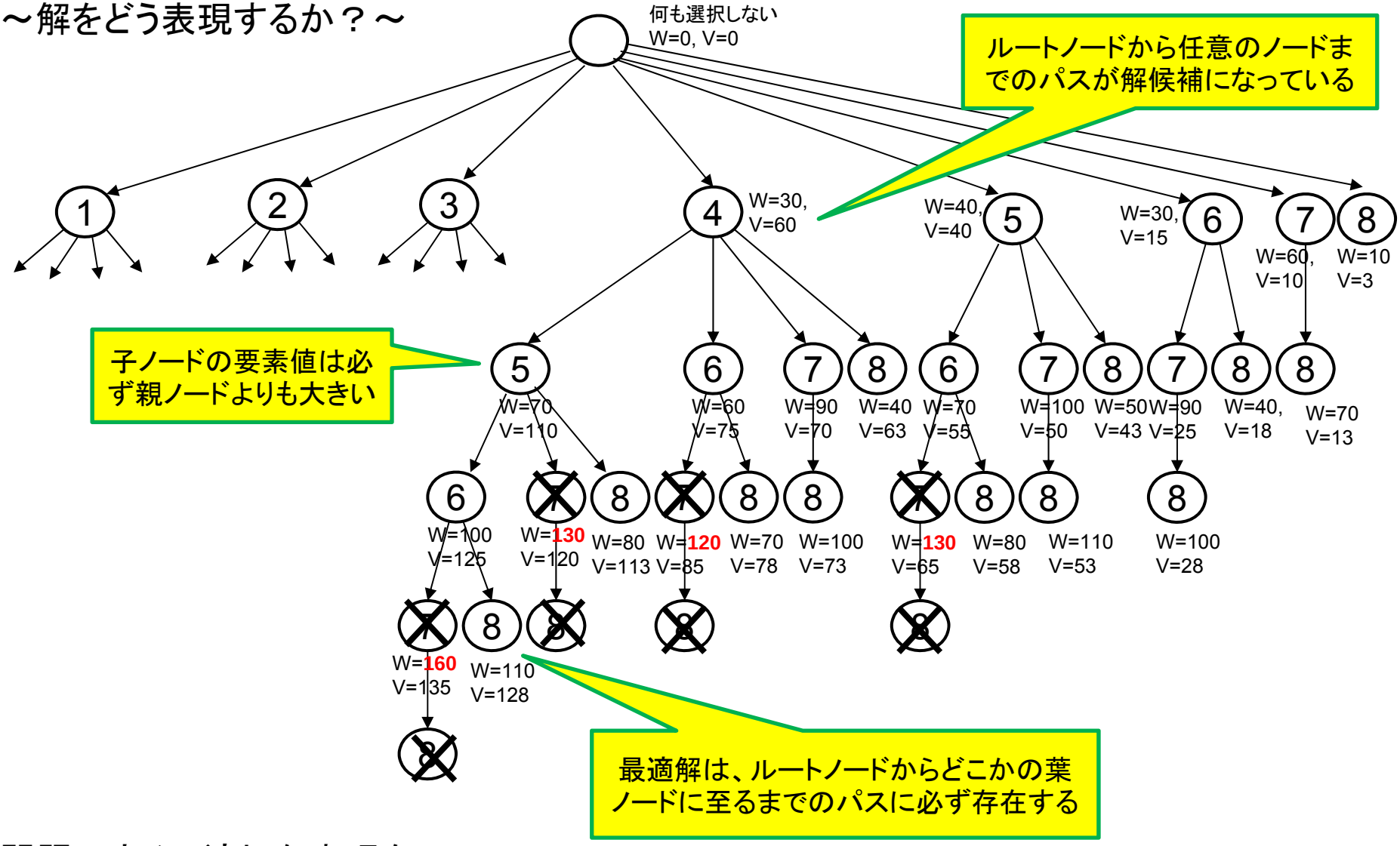
特徴

○ **最適解**を素早く見つける

X 大きな記憶容量を要す

分枝限定法における様々な「列挙図」表現

～解をどう表現するか？～



問題に応じて適切な表現を
選択・考案せよ

まとめ

組合わせ最適化　と　分枝限定法

【分枝限定法とは？】

1）解候補をツリー状の列挙図のように列挙・評価していくが、ツリー生成途中で
　　解の可能性の無い（あるいは可能性が低い）枝を刈ることで探索空間を減らす
2）解の可能性の無い枝だけを刈れば最適解を得る

【深さ優先探索】と【幅優先探索】

・分枝限定法における枝の展開方法　…　深さ優先のほうが省メモリ

【解候補および列挙図の表現方法】

・問題に応じて（あるいは好みに応じて）適切な表現を選択・考案せよ
　→　実問題においては、この「表現方法」を見つけることこそエンジニアの仕事

【レポート課題】　2019.01.11

下記のナップサック問題を「欲張り法」および「分枝限定法」を用いてそれぞれ解を求めよ。

レポートにはそれぞれの方法での導出過程を明記すること。

ナップサック容量：100

要素 1	weight=44	value=73
要素 2	weight=22	value=84
要素 3	weight=27	value=91
要素 4	weight=32	value=97
要素 5	weight=4	value=2
要素 6	weight=14	value=53
要素 7	weight=24	value=89
要素 8	weight=17	value=56
要素 9	weight=14	value=38
要素 10	weight=14	value=40

【提出期限】 2019年1月25日（金）午後5時
【提出先】　講義後木村に直接手渡し
　　または　W2号館6階634号室