Advanced Algorithms

Lecture 12
Backtracking &
Branch-and-Bound

Bin Yang
byang@cs.aau.dk

ILO of Lecture 12

- Backtracking, Branch-and-Bound
  - to understand the principles of **backtracking** and **branch-and-bound** algorithm design techniques;
  - to understand how these algorithm design techniques are applied to the example problems (**CNF-Sat** and **Knapsack**).

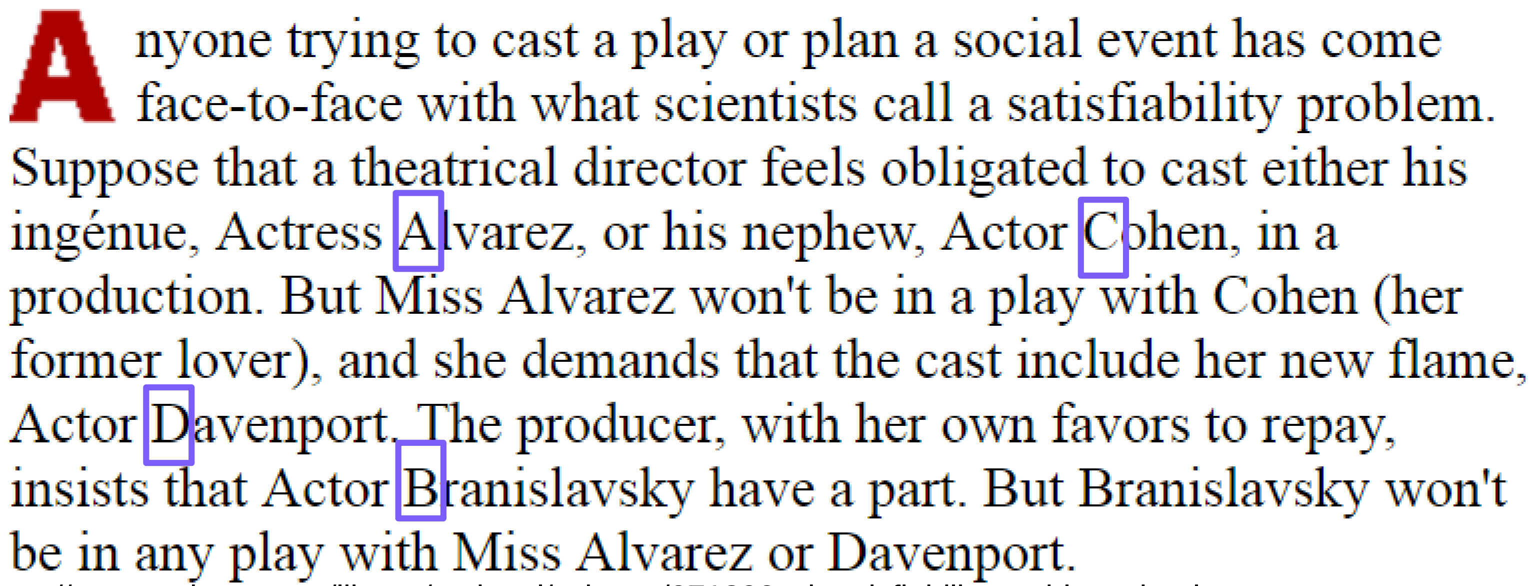
Coping with NP-complete problems

- Many interesting and important problems are NP-complete.
- Do we always surrender if we have an NP-complete problem?
- NO! We have some ways to deal with an NP-complete problem.
  - If the actual inputs are small, an algorithm with exponential running time may be acceptable.
  - Come up approaches to find *near-optimal* solutions in *polynomial* time.
    - Approximation algorithm (must be associated with an approximation ratio)
  - Use *heuristics* to speed up exponential running time.
    - Backtracking (for decision problems) and branch-and-bound (for optimization problems)

Agenda

- The CNF-Sat problem
- Backtracking
- Branch-and-Bound
- The Knapsack problem
- Summary

The Satisfiability Problem



Anyone trying to cast a play or plan a social event has come face-to-face with what scientists call a satisfiability problem. Suppose that a theatrical director feels obligated to cast either his ingénue, Actress Alvarez, or his nephew, Actor Cohen, in a production. But Miss Alvarez won't be in a play with Cohen (her former lover), and she demands that the cast include her new flame, Actor Davenport. The producer, with her own favors to repay, insists that Actor Branislavsky have a part. But Branislavsky won't be in any play with Miss Alvarez or Davenport.

http://www.nytimes.com/library/national/science/071399sci-satisfiability-problems.html

- Four actors become Boolean variables: a, b, c, d
  - True: play; false: not play
- $a \vee c$
- $a \rightarrow \neg c$
- d
- b
- $b \rightarrow (\neg a \wedge \neg d)$

Impossible to satisfy all constraints!

a=T, c=T; a=F, c=T; a=T, c=F;
a=T, c=F; a=F, c=T; a=F, c=F;
d=T
b=T
b=T, a=F, d=F

Formula satisfiability problem

- Given a Boolean formula that is composed of
  - n Boolean variables: $x_1, x_2, \ldots, x_n$;
  - m Boolean connectives: and ($\wedge$), or ($\vee$), not ($\neg$), implication ($\rightarrow$), if and only if ($\leftrightarrow$), and parentheses ().
- We want to see if there is a satisfying assignment for the Boolean formula.
  - A set of values for the variables such that make the formula be true.
- Previous example:
  - $(a \vee c) \wedge (a \rightarrow \neg c) \wedge d \wedge b \wedge (b \rightarrow (\neg a \wedge \neg d))$
  - No satisfying assignment.
- Another example:
  - $((a \rightarrow b) \vee \neg((\neg a \leftrightarrow c) \vee d)) \wedge \neg b$
  - Has a satisfying assignment, e.g., a=F, b=F, c=T, d=T
  - $((F \rightarrow F) \vee \neg((\neg F \leftrightarrow T) \vee T)) \wedge \neg F = (T \vee \neg T) \wedge \neg F = T$

CNF-Sat

- Conjunctive Normal Form (CNF) for Boolean formulas
  - A **literal** is a variable or its negation.
    - $a, b, \neg a, \neg b$
  - Each **clause** is a disjunction of **literals**
    - **OR** of one or more literals: $a \vee c$, $\neg a \vee d \vee d$
  - CNF is a conjunction of **clauses**
    - **AND** of clauses: $(a \vee c) \wedge (\neg a \vee d \vee d)$
- Any Boolean formula can be transformed to CNF
  - $\neg(a \vee c) = \neg a \wedge \neg c$
  - $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$
  - $a \wedge (b \vee (d \wedge e)) = a \wedge (b \vee d) \wedge (b \vee e)$
  - …

CNF-Sat brute force

- CNF-Sat is NP-complete
- How do we solve it then with brute force?
  - Consider all possible assignments of Boolean values to all variables in the formula:

x_1	x_2	x_3	…	x_n	formula
F	F	F	…	F	?
F	F	F	…	T	?
…			…		?
T	T	T	…	T	?

  - What is the run-time?

Agenda

- The CNF-Sat problem
- Backtracking
- Branch-and-Bound
- The Knapsack problem
- Summary

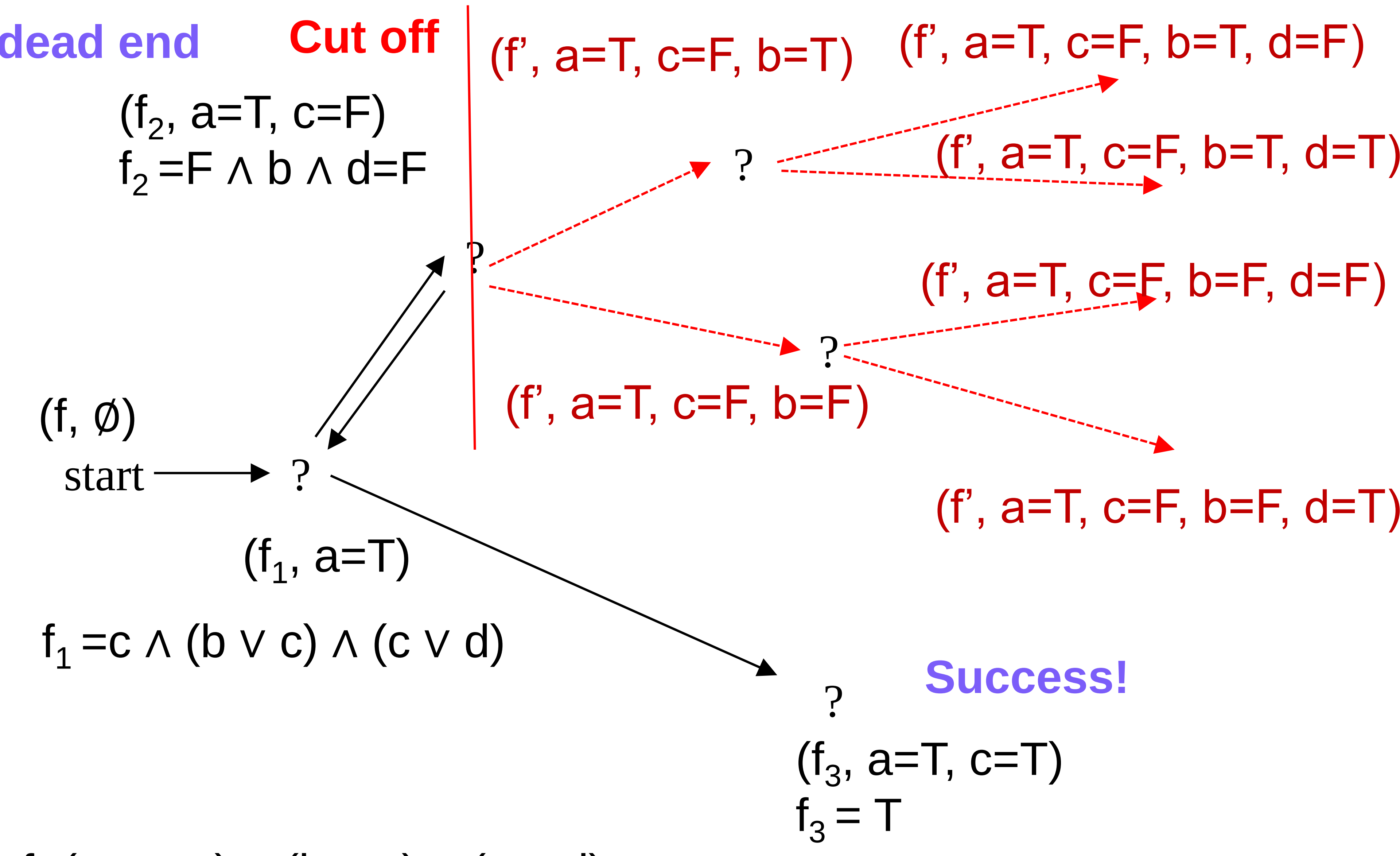
Backtracking

- We can do better in practice:
  - We use the structure of an NP-complete problem:
    - If we have a certificate, we can check.
      - This is very efficient since all NP-complete problems can be verified in polynomial time.
    - A certificate is constructed by making a number of choices.
    - What are these choices for the CNF-Sat?
      - Assign T or F to variables.

Backtracking

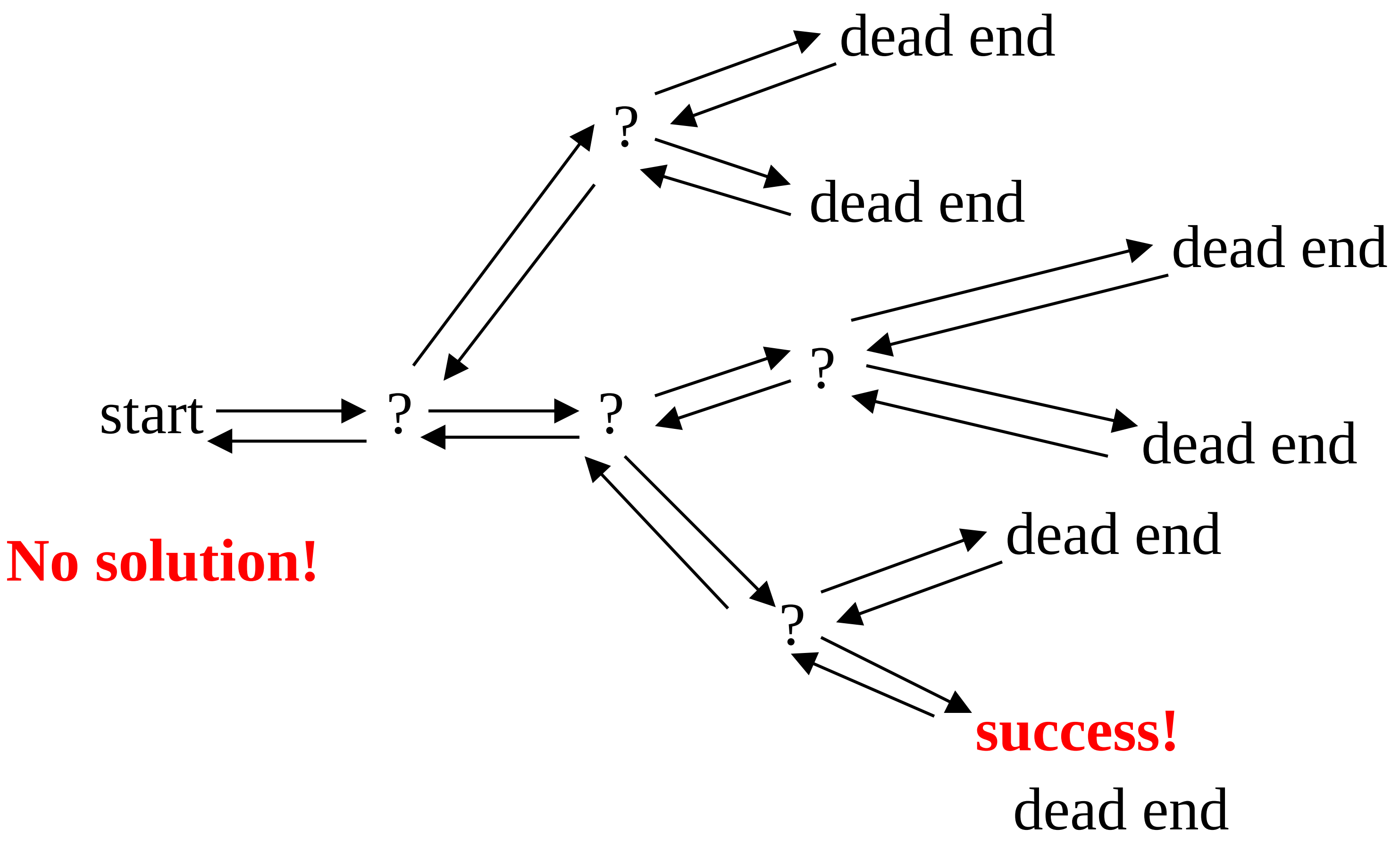
- A backtracking algorithm searches through a large (possibly even exponential-size) set of possibilities in a **systematic** way
- It traverses through possible search paths to locate ***solutions*** or ***dead ends***.
- The configuration of a path consists of a pair (X, Y)
  - X is the remaining sub-problem to be solved
  - Y is the set of choices that have been made to get to this sub-problem x from the original problem instance.
- ***Dead end:*** a configuration (X, Y) that cannot lead to a valid solution no matter how additional choices are made
  - Cuts off all future searches from this configuration and ***backtracks*** to another configuration.

Backtracking for CNF-Sat



f=(¬a ∨ c) ∧ (b ∨ c) ∧ (c ∨ d)

Backtracking



Backtracking

- **Backtracking** algorithm design technique:
  - Have a ***frontier:*** a set of configurations.
  - *Observation* 1: sometimes we can see that a configuration is a ***dead end*** – it can not lead to a solution.
    - We *backtrack*, discarding this configuration (cut-off).
  - *Observation* 2: If we have several configurations, some of them may be more “promising” than the others (*Heuristics*).
    - We consider the promising configurations first.

Template for Backtracking

```
Backtracking(P)          // Input: problem P
F ← {(P, ∅)}             // Frontier set of configurations
while F ≠ ∅ do
  Remove (X,Y) ∈ F - the most "promising" configuration
  Expand (X,Y), by making a choice(es)
  Let $(X_1,Y_1)$, $(X_2,Y_2)$, ..., $(X_k,Y_k)$ be new configurations
  for each new configuration $(X_i,Y_i)$ do
     "Check" $(X_i,Y_i)$
     if "solution found" then
        return the solution derived from $(X_i,Y_i)$
     if not "dead end" then
        F ← F ∪ {$(X_i,Y_i)$}    // else "backtrack"
return "no solution"
```

Details to fill in

- Important details in a backtracking algorithm:
  - Define a way of selecting the most “promising” configuration.
    - LIFO (stack)– **depth-first** search
    - FIFO (queue) – **breadth-first** search
    - Some ***heuristic*** ordering – ***best-first*** search
  - Specify the way of extending a configuration (X, Y) into sub-problem configuration(s).
  - Describe how to perform a consistency check for a configuration (X, Y) for “solution found” and whether it is a “dead end”.

Fill in the details for CNF-Sat

- CNF-Sat : What is a configuration?
  - Choices made so far Y: An assignment to a subset of variables
  - Sub-problem X: CNF with the remaining variables
- What is a promising configuration?
  - Formula with the smallest clause (break ties by taking the shortest formula)
    - Idea: to show that this is a dead end or a solution ASAP.
  - Other choices are possible, e.g., depth-first and breadth-first.
- How do we generate sub-problems ?
  - Take the smallest clause and pick a variable x, generate two sub-problems:
    - One corresponds to x = T
    - The other one corresponds to x = F

Fill in the details for CNF-Sat

- Details on generating sub-problems
- For each choice of assignment to *x* (i.e., T or F) do:
  - 1. Assign the value to *x* everywhere in the formula
    - If a literal is assigned with ***T*, *its clause disappears***
      - E.g., $(a \vee c) \wedge (b \vee c)$, when a=T, then the formula becomes $b \vee c$.
    - If a literal is assigned with ***F*, *the literal disappears***
      - E.g., $(a \vee c) \wedge (b \vee c)$, when a=F, then the formula becomes $c \wedge (b \vee c)$.
  - 2. If this results in a clause with a single literal (a unit clause), assign T to that single literal and propagate as in step 1.
    - E.g., $c \wedge (b \vee c)$, make c=T, formula becomes empty.
  - Keep doing 2 while there are unit clauses.
- How to do consistency check:
  - Dead-end: a single-literal clause is forced to be 0
    - $c \wedge \neg c$, when c=T, the first clause disappears;
    - The second clause has a single-literal which is forced to be 0.
  - Solution: all clauses disappear, i.e., ***empty formula***.

Davis-Putnam procedure

Here, Φ is the formula, but not an empty set.

Other heuristics can be used, e.g., the formula with the smallest clause.

Two new configurations for x=T and x=F, respectively.

```
1: procedure DAVIS-PUTNAM($\phi$)
2:     $S \leftarrow \{(\phi, [])\}$        ▷ Initially, S contains only a pairing of $\phi$ with the empty truth assignment.
3:     while $S$ is not empty do
4:         $(\phi', t) \leftarrow$ an arbitrary element of $S$
5:         $x \leftarrow$ an arbitrary literal in $\phi'$
6:         $S \leftarrow (S - \{(\phi', t)\}) \cup$ CHASE$(\phi, x, t) \cup$ CHASE$(\phi, \overline{x}, t)$
7:     end while
8: end procedure
```

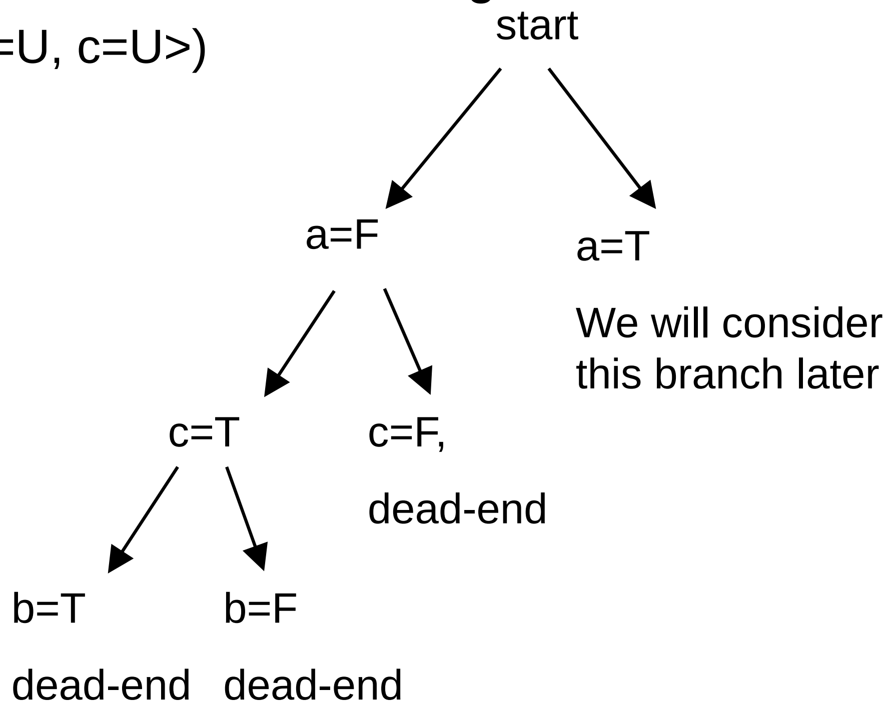
set x to be true

Update the choices made so far.
Update the sub-problem to be solved.

```
9: function CHASE($\phi$, $x$, $t$)
10:     Set $x$ to $true$ in $t$
11:     Delete all clauses from $\phi$ that contain the literal $x$
12:     Delete the literal $\overline{x}$ from all clauses in $\phi$
13:     if $\phi$ is empty then abort from DAVIS-PUTNAM with $t$        ▷ $t$ is a satisfying assignment
14:     elseif $\phi$ contains an empty clause then return {}        ▷ $t$ contains bad decisions
15:     elseif $\phi$ contains a unit clause $(y)$ then return CHASE$(\phi, y, t)$        ▷ Continue the chase
16:     else return $\{(\phi, t)\}$
17: end function
```

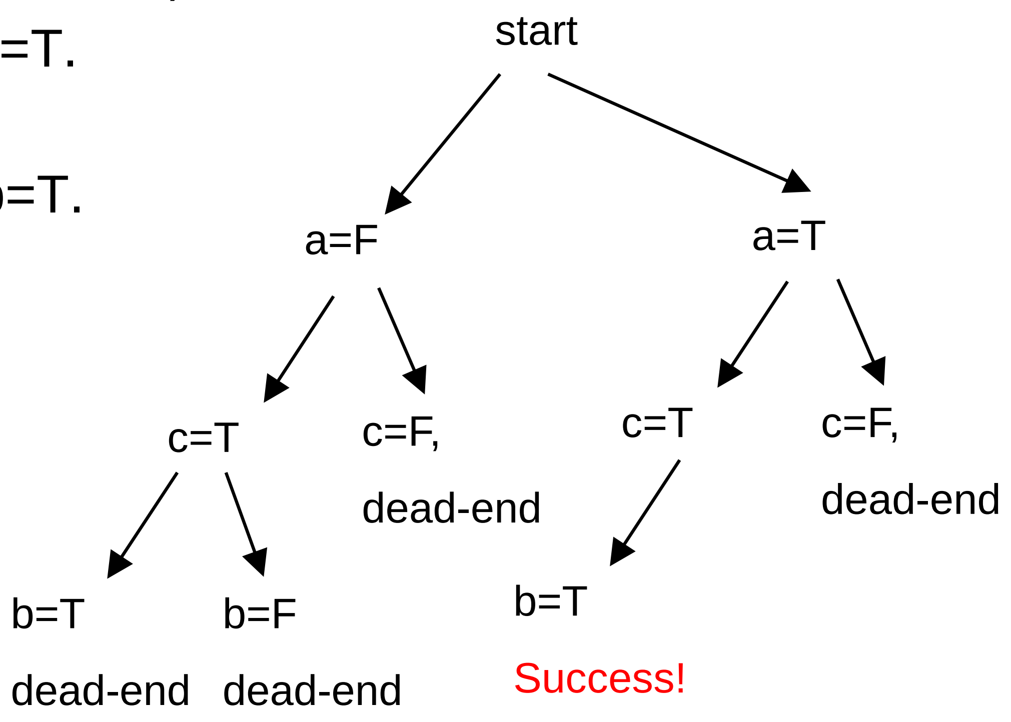
Example

- P=(a ∨ c) ∧ (¬ a ∨ c) ∧ (b ∨ ¬ c) ∧ (a ∨ ¬ b)
- Current configuration: (P, <a=U, b=U, c=U>)
- Pick a variable
  - Consider (¬ a ∨ c) and pick ¬ a, set it to T, meaning that a=F
    - (c ∧ (b ∨ ¬ c) ∧ (¬ b), <a=F, b=U, c=U>)
    - c is single literal, so set c=T.
      - (b ∧ (¬ b), <a=F, b=U, c=T>)
    - b is single literal, so set b=T.
      - ((¬ b), <a=F, b=T, c=T>)
      - ¬ b=F, so dead-end.



Example

- (a ∨ c) ∧ (¬ a ∨ c) ∧ (b ∨ ¬ c) ∧ (a ∨ ¬ b)
  - Consider (¬ a ∨ c) and pick ¬ a, set it to F, meaning that a=T
    - (c ∧ (b ∨ ¬ c), <a=T, b=U, c=U>)
  - c is single literal, so set c=T.
    - (b, <a=T, b=U, c=T>)
  - b is single literal, so set b=T.
    - (∅, <a=T, b=T, c=T>)



Agenda

- The CNF-Sat problem
- Backtracking
- Branch-and-Bound
- The Knapsack problem
- Summary

Optimization problem

- We have seen how backtracking helps us solve **decision** problems.
- Can we use a backtracking algorithm to solve an **optimization** problem?
  - Idea: Still use a backtracking algorithm but modify it slightly.
    - Maintain a best solution B.
    - We cannot stop when the first solution is found. Instead,
    - When a solution S is found:
      - If S is better than the best solution seen so far, denoted as B, then update B=S;
      - Otherwise discard the solution.
    - Continue until all possible solutions are found.

Pruning

- This works, but we can do better – discard solutions earlier:
  - For *minimization* problems:
    - We estimate a *lower-bound lb* on the cost of a solution derived from a configuration *C.*
    - We discard *C*, whenever *lb*(*C*) is *larger than* the cost of the *smallest* solution found so far, which is maintained in *B*.
  - For *maximization* problems:
    - We estimate an *upper-bound ub* on the cost of a solution derived from *configuration C*.
    - We discard *C*, whenever *ub*(*C*) is *smaller than* the cost of the *largest* solution found so far, which is maintained in *B*.
  - This is called ***pruning.***
    - For example, if a partially constructed cycle *P* in TSP problem is longer than the best solution found so far, we can discard *P.*
- Backtracking together with pruning constitute the ***branch-and-bound*** algorithm design technique.

Branch-and-Bound

```
Branch-and-Bound(P) // Input: maximization problem P
F ← {(P, $\emptyset$)}     // Frontier set of configurations
B ← ($-\infty$, $\emptyset$)      // Best cost and corresponding solution
while F $\neq \emptyset$ do
   remove (X,Y) $\in$ F - the config. with the largest ub(X,Y)
   Expand (X,Y), by making a choice(es)
   Let ($X_1$,$Y_1$), ($X_2$,$Y_2$), ..., ($X_k$,$Y_k$) be new configurations
   for each new configuration ($X_i$,$Y_i$) do
      "Check" ($X_i$,$Y_i$)
      if "solution found" then
         if the cost c of ($X_i$,$Y_i$) is more than B.cost then
            B ← (c,($X_i$,$Y_i$))
      if not "dead end" then
         if ub($X_i$,$Y_i$) is more than B.cost then  // pruning
            F ← F $\cup$ {($X_i$,$Y_i$)}    // else "backtrack"
return B
```

If a better solution is found, update the best solution B.

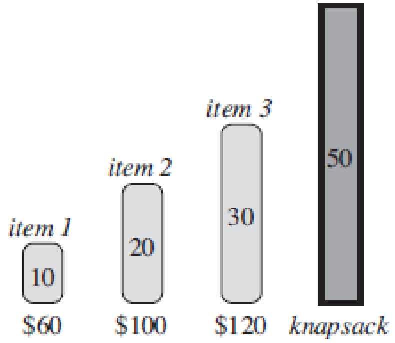
If upper bound is more than the best solution B, this means that we may getter a better solution if we keep exploring from Yi, so we need keep this new configuration. Otherwise, we prune this new configuration,

Agenda

- The CNF-Sat problem
- Backtracking
- Branch-and-Bound
- The Knapsack problem
- Summary

Knapsack problem

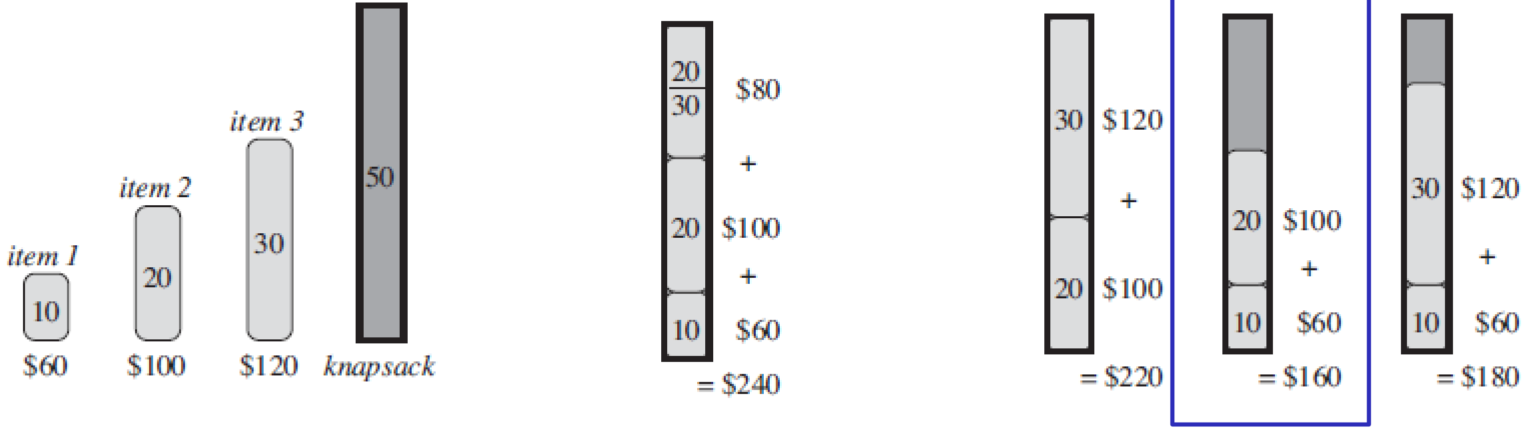
- Input:
  - A set of items, each with a given weight and value
    - $S = \{(w_1, v_1), (w_2, v_2), \ldots, (w_n, v_n)\}$,
  - a maximum weight or the knapsack capacity: *W*



- Output: $T \subset S$, such that:
  - $\sum_{s_i \in T} s_i.wi \leq$ W, the total weight does not exceed the maximum weight W.
  - $\sum_{s_i \in T} s_i.v_i$ is maximized. The total value is maximized.
- Example: a thief robbing a store that has n items.
- Two versions:
  - **0-1** *knapsack* problem: either take an item or leave it: NP-complete.
  - Fractional knapsack problem: can take fractions of items: greedy algorithm.

Fractional knapsack problem

- Sort items by gain, i.e., v_i/w_i
- Take in that order, cut the last one if it does not fit to fill exactly W.
  - Item1: 6, item 2: 5, item 3: 4
  - Fill in 10 of item 1, fill 20 of item 2, and fill 20 of item 3.
    - We get 60 + 100 + 80 = 240.



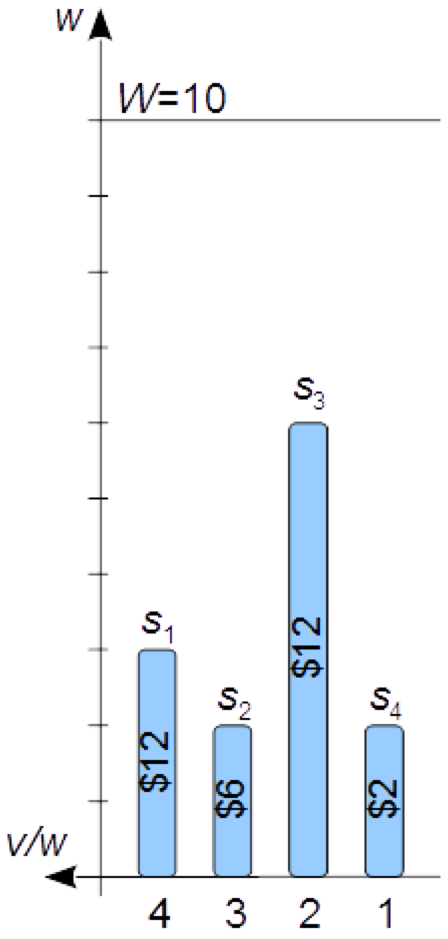
Greedy strategy

- Greedy strategy does not work for 0-1 knapsack.
  - But greed strategy gives a upper bound of the values.

Knapsack: branch-and-bound

- What is a configuration? (T, k)
  - T: the set of chosen items. *The choices that have been made.*
  - k: the index of the next item to be considered in the gain-sorted order. *The remaining sub-problem.*
  - In addition, two attributes is associated with a configuration
    - Current value $v = \Sigma_{s \in T} v(s)$;
    - Current weight $w = \Sigma_{s \in T} w(s)$
- How do we generate new configurations?
  - Binary choice: adding or not adding item k.
- How do we compute an upper bound of value?
  - $ub(T, k) = v + greedy\_solution(\{s_k,\dots,s_n\}, W - w)$
- Which configuration is the most promising?
  - The one with the largest upper bound!
- When do we see that a configuration is a dead-end ?
  - Current weight w is over W

Example

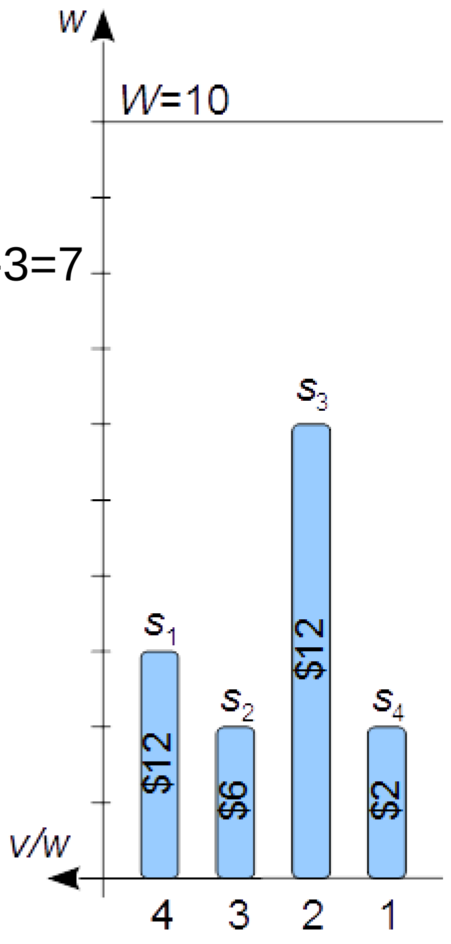


- Let's run the algorithm on our example.
  - Note that S1, S2, S3, and S4 are already ordered according to their gains.
    - S1: gain=4, value=12, weight=3.
    - S2: gain=3, value=6, weight=2.
    - S3: gain=2, value=12, weight=6.
    - S4: gain=1, value=2, weight=2.

Example

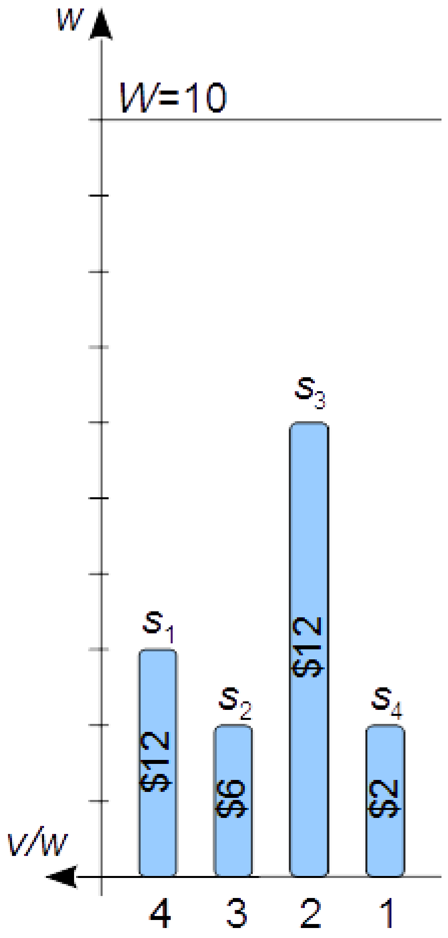
- Sort items by their gains.
  - S1: 4, S2: 3, S3: 2, S4: 1
- In the beginning, (T, k)=(Ø, 1)
  - Make two choices, including S1 or not.
  - ({S1}, 2): v=12, w=3
    - Upper bound: how much can you hold using W-w=10-3=7
    - Greedy: put 2 of S2, put 5 of S3. 6+10=16.
    - 12+16=28
  - (Ø, 2): v=0, w=0
    - Greedy: put 2 of S2, put 6 of S3, put 2 of S4
    - 6+12+2=20

28, ({S1}, 2)
20, (Ø, 2)



Example

- Expand ({S1}, 2) as its upper bound is the largest. Including S2 or not.
  - ({S1, S2}, 3): v=18, w=5, W-w=5
    - Greedy: put 5 of S3, 10.
    - Upper bound: 18+10=28.
  - ({S1}, 3): v=12, w=3, W-w=7
    - Greedy: put 6 of S3, put 1 of S4, 12+1=13.
    - Upper bound: 12+13=25.



28, ({S1}, 2)
20, (∅, 2)

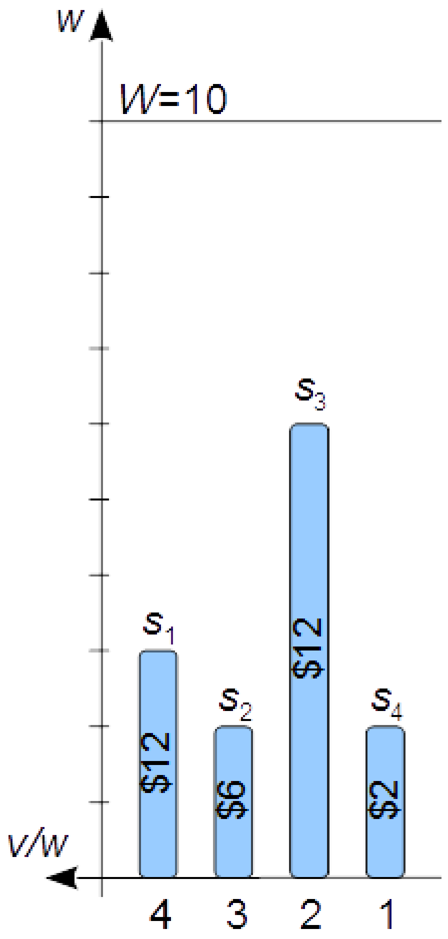
28, ({S1, S2}, 3)
25, ({S1}, 3)
20, (∅, 2)

Example

- Expand ({S1, S2}, 3) as its upper bound is the largest. Including S3 or not.
  - ({S1, S2, S3}, 4): w=11> W, dead-end.
  - ({S1, S2}, 4): v=18, w=5, W-w=5.
    - Greedy: put 2 of S4, 2.
    - Upper bound: 18+2=20.

28, ({S1, S2}, 3)
25, ({S1}, 3)
20, (∅, 2)

25, ({S1}, 3)
20, (∅, 2)
20, ({S1, S2}, 4)

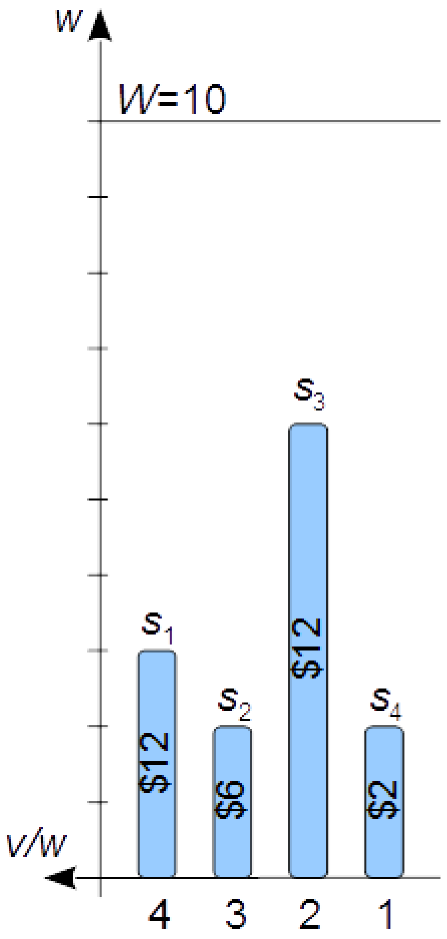


Example

- Expand ({S1}, 3) as its upper bound is the largest. Including S3 or not.
  - ({S1, S3}, 4): v=24, w=9, W-w=1
    - Greedy: put 1 of S4, 1.
    - Upper bound: 24+1=25.
  - ({S1}, 4): v=12, w=3, W-w=7
    - Greedy: put 2 of S4, 2.
    - Upper bound: 12+2=14.

25, ({S1}, 3)
20, (∅, 2)
20, ({S1, S2}, 4)

25, ({S1, S3}, 4)
20, (∅, 2)
20, ({S1, S2}, 4)
14, ({S1}, 4)



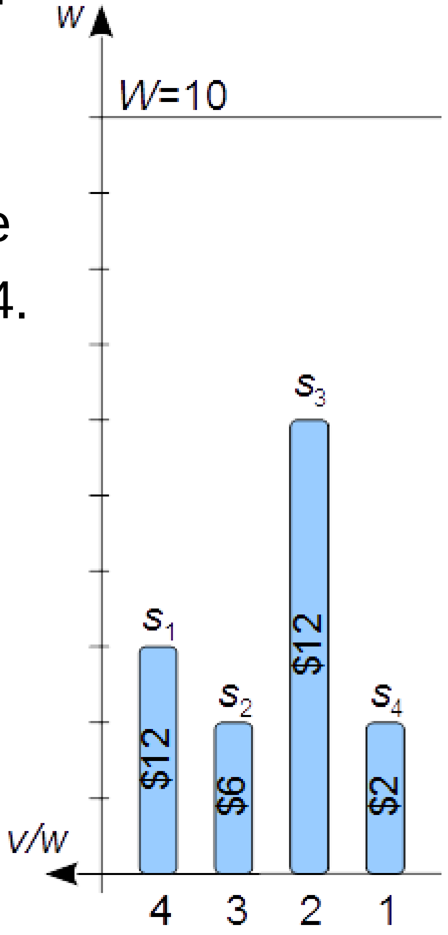
Example

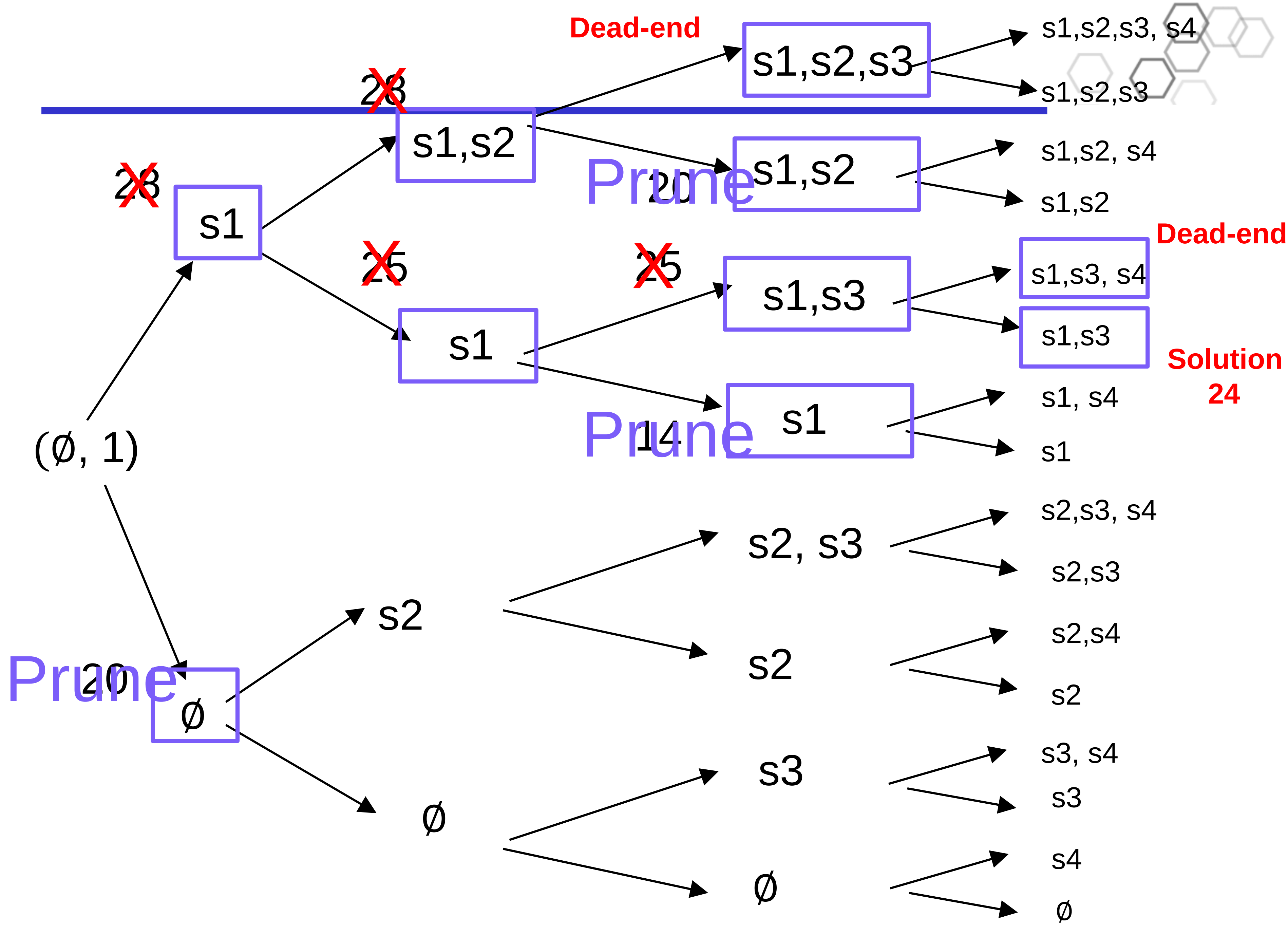
- Expand ({S1, S3}, 4) as its upper bound is the largest. Including S4 or not.
  - ({S1, S3, S4}, NULL): w=11 > 10. Dead-end.
  - ({S1, S3}, NULL): v=24, w=9, **solution**. Update B=24.

- Pruning:
  - All the remaining entries in the priority queue can be pruned as their upper bounds are smaller than B=24. Meaning that they are impossible to get a solution whose value can be greater than 24.

25, ({S1, S3}, 4)
20, (Ø, 2)
20, ({S1, S2}, 4)
14, ({S1}, 4)

20, (Ø, 2)
20, ({S1, S2}, 4)
14, ({S1}, 4)

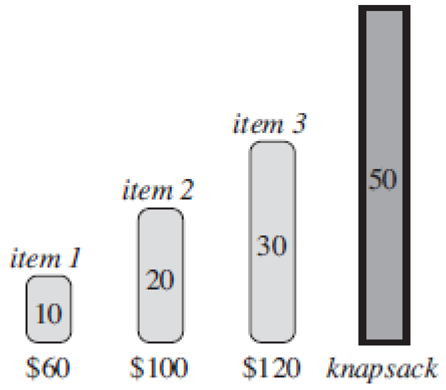




Dead-end
s1,s2,s3
s1,s2,s3, s4
s1,s2,s3
28
s1,s2
Prune
20
s1,s2
s1,s2, s4
s1,s2
28
s1
Dead-end
25
25
s1,s3
s1,s3, s4
s1,s3
s1
Solution 24
Prune
14
s1
s1, s4
s1
(∅, 1)
s2, s3
s2,s3, s4
s2,s3
s2
s2
s2,s4
s2
Prune
20
∅
s3
s3, s4
s3
∅
∅
s4
∅

Mini quiz (also on Moodle)

- How many configurations you need to check in total?
- The first one you need to check is (Ø, 1).



ILO of Lecture 12

- Backtracking, Branch-and-Bound
  - to understand the principles of **backtracking** and **branch-and-bound** algorithm design techniques;
  - to understand how these algorithm design techniques are applied to the example problems (**CNF-Sat** and **Knapsack**).

Self-study exercises 3

- You will see exam 2018.
- Get a feeling about how does an exam look like.

- Send your solutions to me by email byang@cs.aau.dk before 4 April.

Agenda

- The CNF-Sat problem
- Backtracking
- Branch-and-Bound
- The Knapsack problem
- Summary

Intended Learning Outcomes (ILO)

- After taking this course, you should acquire the following knowledge
  - Algorithm **design** techniques such as divide-and-conquer, greedy algorithms, dynamic programming, back-tracking, branch-and-bound algorithms, and plane-sweep algorithms;
  - Algorithm **analysis** techniques such as recursion, amortized analysis;
  - A collection of **core** algorithms and data structures to solve a number problems from various computer science areas: algorithms for external memory, multiple-threaded algorithms, advanced graph algorithms, heuristic search and geometric calculations;
  - There will also enter into one or more **optional subjects** in advanced algorithms, including, but not limited to: *approximate algorithms*, randomized algorithms, search for text, linear programming and number theoretic algorithms such as cryptosystems.

ILO

- Lecture 1: Dynamic programming (CLRS 15)
  - Principles of DP. Overlapping sub-problems and optimal sub-structure. Top-down with memoization and bottom-up.
  - Examples: Activity selection, edit distance.
- Lecture 2: All-pairs shortest paths (CLRS 25)
  - Distance matrix and predecessor matrix.
  - Repeated squaring and Floyd-Warshall algorithm.
- Lecture 3: Network flow algorithms (CLRS 26)
  - Flow networks, flows, maximum-flow.
  - Ford-Fulkerson method and Edmonds-Karp algorithm.
- Lecture 4: Greedy algorithm (CLRS 16)
  - Optimal sub-structure and greedy choice property.
  - Greedy exchange to prove greedy choice property.
  - Examples: Activity selection, Huffman coding.

ILO

- Lecture 5: Amortized analysis (CLRS 17)
  - Understand amortized analysis, difference from average-case analysis.
  - Aggregated analysis, accounting method, potential method.
- Lectures 6, 7, 8: Computational geometry (CLRS 33 + additional references)
  - Basic geometric operations in 2D, e.g., two line segments intersecting, orientation of two line segments?
  - Sweeping algorithms, whether any two LSs intersecting.
  - Graham's scan and Jarvis's march (output-sensitive) algorithm for convex hull.
  - Divide-and-conquer: to find the closet pair of points in a set of points and convex hull.
  - Range searching in d-dimensional space: balanced BST (only for 1D), kd-tree and range tree.

ILO

- Lecture 9: External-memory algorithms and data structures (CLRS 18 + additional references)
  - Running time is dominated by the number of I/O operations.
  - Balanced search trees, e.g., *B-trees.* Short and fat trees!
  - External memory sorting: multi-way merge-sort algorithm.
- Lecture 10: Multi-threaded algorithms (CLRS 27)
  - Nested parallelism (spawn, sync), parallel loops (parallel).
  - Important concepts: work, span, and parallelism.
  - MT Fib number, MT merge sort (MT merge, lower the span).
- Lecture 11 & 12: Algorithms for NP-complete problems (CLRS 35)
  - Approximation algorithms, approximation ratios.
    - Vertex cover, traveling salesman.
  - Backtracking and branch-and-bound
    - CNF-Sat, 0-1 Knapsack

Exam

- 4 June 2019.
- 10 to 12. Only 2 hours.
  - Previous years, it is 3 hours.
  - Less questions compared to previous years.
- Censor: Simonas Saltenis

- Individual and written, but open-book
  - Any electronic devices with communication capabilities, such as laptops and mobile phones, are **NOT** allowed.
  - But you can freely use your copies of slides from the lectures, textbooks, and other course material.

Exam

- Exam in two parts
  - Quizzes, to test knowledge. Like in-class mini-quizzes.
    - Choose the option(s) that you think is(are) correct out of a set of given options.
    - Ask you to fill in a number/character or a sequence of numbers/characters.
  - Open problems, with some sub-problems, to test competences and skills.
    - Given a real world problem, write pseudo code, give complexity analysis, etc.
    - Like exercises and self-study exercises.

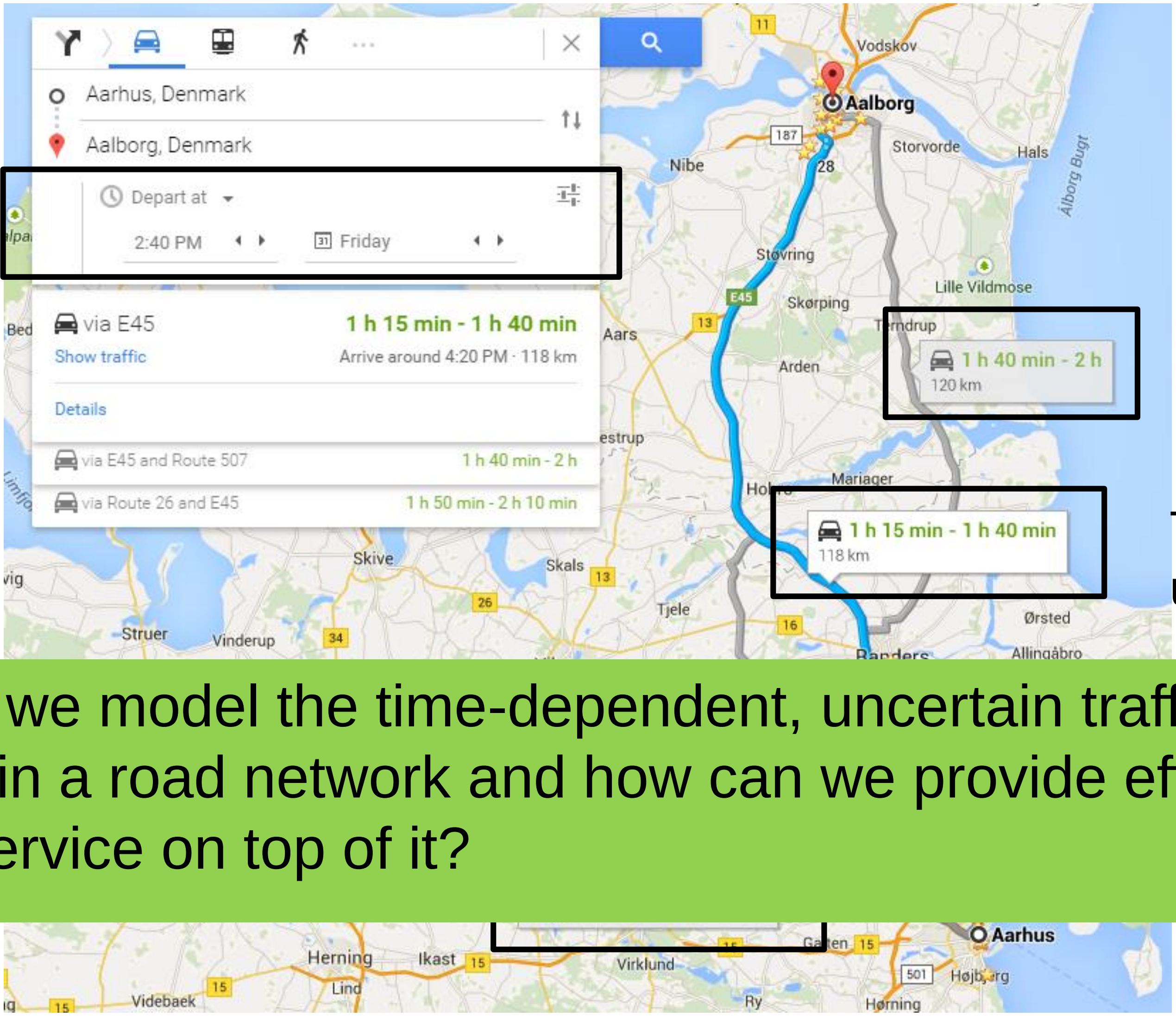
- The exam will have 100 points.
  - 50 for quizzes and 50 for open problems.

Daisy: Center for Data-Intensive Systems

- World leading research center on data management and machine learning.
  - Daisy ranks the second best among all research groups in Europe according to publication performance in the top data management outlets in the recent 10-year period.
  - Daisy Director, Prof. Christian S. Jensen, has the highest h-index among all computer science professors in Denmark.
- Many international collaborations with top universities.
- Many projects funded by, the EU, Independent Research Fund Denmark, and private foundations.

Data-Intensive Transportation Science

Traffic is time dependent



Traffic is also uncertain

How can we model the time-dependent, uncertain traffic situation in a road network and how can we provide efficient routing service on top of it?

Data-Intensive Transportation Science

- A weighted graph where edges weights are not just real values, but time-varying (e.g., peak vs off-peak) distributions.
- Google: [100, 120]@peak, [90, 110]@off-peak
- We try to model uncertainty with finer granularities.
  - {100:0.4, 110:0.2, 120:0.3}@peak
  - {90:0.5, 100:0.2, 110:0.1}@off-peak
  - {100:0.4, 110:0.2, 115:0.1, 120:0.1}@8.00 am
- These uncertain weights need to be learned and predicted from traffic data, e.g., GPS data. (Machine learning)
- Shortest path finding on such time-dependent and uncertain edge weights is more challenging than the traditional setting where Dijkstra’s algorithm works. (Algorithms)

Join Daisy!

- We have opening master theses/PhD positions/student programmers every year on different topics in machine learning and data analytics.
- I have two projects supported by Independent Research Fund Denmark.
- Astra: AnalyticS of Time seRies in spAtial networks
    - https://astra.cs.aau.dk/
    - How to use machine learning to predict time series from cyber-physical systems?
- A Data-Intensive Paradigm for Dynamic, Uncertain Networks
    - How to use artificial intelligence to make transportation greener and smarter?
    - https://dff.dk/cases/kunstig-intelligens-gor-trafikken-smartere-og-gronnere-1

Thank you!

- If you have questions regarding to the course, exercises, etc., drop by my office or send me an email.
- If you have any other comments, you are also very welcome to let me know.
- If you are interested in the research topics in Daisy, also feel free to contact me.
- Good luck for the exam!