Artificial Intelligence and Machine Learning (AIML)

2023–24

- **Last lecture**: combinatorial optimization in AI
- **This lecture**: exact SDP methods for combinatorial optimization

Exact methods: overview

- Many strategies for constructing practical exact methods: **divide-and-conquer**, **branch-and-bound**, **dynamic programming**

分支定界

Exact methods: overview

- Many strategies for constructing practical exact methods: **divide-and-conquer**, **branch-and-bound**, **dynamic programming**
- Usefulness/efficiency of each strategy depends upon the specific structure of the problem, no single strategy is best for all problems

Exact methods: overview

- Many strategies for constructing practical exact methods: **divide-and-conquer**, **branch-and-bound**, **dynamic programming**
- Usefulness/efficiency of each strategy depends upon the specific structure of the problem, no single strategy is best for all problems
- Choice of strategy therefore requires understanding of the specifics of the problem itself, available computational resources and/or availability of other information about the problem

Exact methods: overview

- Many strategies for constructing practical exact methods: **divide-and-conquer**, **branch-and-bound**, **dynamic programming**
- Usefulness/efficiency of each strategy depends upon the specific structure of the problem, no single strategy is best for all problems
- Choice of strategy therefore requires understanding of the specifics of the problem itself, available computational resources and/or availability of other information about the problem
- We will concentrate on **sequential decision process (SDP)** methods as they encompass many practical AI algorithms

AIML

Exact SDP methods

- An SDP algorithm is a **recursive** process which scans N input data items
  $x_1, \ldots, x_N$ in sequence, generating new candidate configurations by **extending**

Exact SDP methods

- An SDP algorithm is a **recursive** process which scans N input data items $x_1, \ldots, x_N$ in sequence, generating new candidate configurations by **extending** the existing configurations using each input data item in turn
- On each iteration, it **reduces** the set of candidate configurations by removing any which cannot be ultimately extended to an optimal configuration

Exact SDP methods

- An SDP algorithm is a **recursive** process which scans N input data items
  $x_1, \ldots, x_N$ in sequence, generating new candidate configurations by **extending** the existing configurations using each input data item in turn
- On each iteration, it **reduces** the set of candidate configurations by removing any which cannot be ultimately extended to an optimal configuration
- Finally, it selects an optimal configuration from the remaining candidates

Exact SDP methods

- An SDP algorithm is a **recursive** process which scans N input data items $x_1, \ldots, x_N$ in sequence, generating new candidate configurations by **extending** the existing configurations using each input data item in turn
- On each iteration, it **reduces** the set of candidate configurations by removing any which cannot be ultimately extended to an optimal configuration
- Finally, it selects an optimal configuration from the remaining candidates
- Can construct a "computational configuration graph", and special kinds of graphs arise due to the type of exact SDP algorithm: brute-force (**full tree**), greedy (**tree with a single optimal branch at each stage**), dynamic programming (**incomplete tree**)

AIML

Exact SDP methods: algorithm

- **Step 1**. *Initialization*: Start with $n = 0$, generate the "root" configuration(s) in the set of candidate configurations, S.
- **Step 2**. *Extension*: Set $n = n + 1$, and using input data item x_n, extend all candidate configurations in S, and append these to S.
- **Step 3**. *Reduction*: Remove any candidate configurations from S which cannot be extended to an optimal configuration.
- **Step 4**. *Iteration*: if $n < N$, go back to Step 2.
- **Step 5**. *Select best*: Select an optimal configuration X^* from the remaining candidate configurations in S.

Exact SDP methods: algorithm

- **Step 1**. *Initialization*: Start with $n = 0$, generate the "root" configuration(s) in the set of candidate configurations, S.

Exact SDP methods: algorithm

- **Step 1**. *Initialization*: Start with $n = 0$, generate the "root" configuration(s) in the set of candidate configurations, S.
- **Step 2**. *Extension*: Set $n = n + 1$, and using input data item x_n, extend all candidate configurations in S, and append these to S.

Exact SDP methods: algorithm

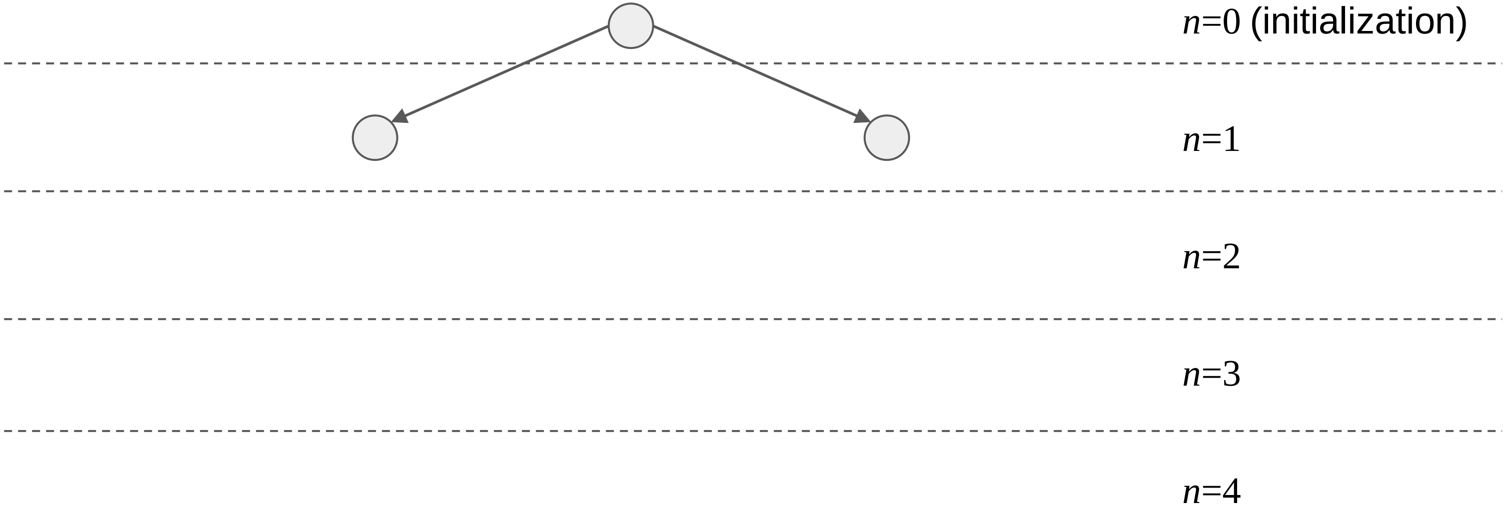
- **Step 1**. *Initialization*: Start with $n = 0$, generate the "root" configuration(s) in the set of candidate configurations, S.
- **Step 2**. *Extension*: Set $n = n + 1$, and using input data item x_n, extend all candidate configurations in S, and append these to S.
- **Step 3**. *Reduction*: Remove any candidate configurations from S which cannot be extended to an optimal configuration.

Exact SDP methods: algorithm

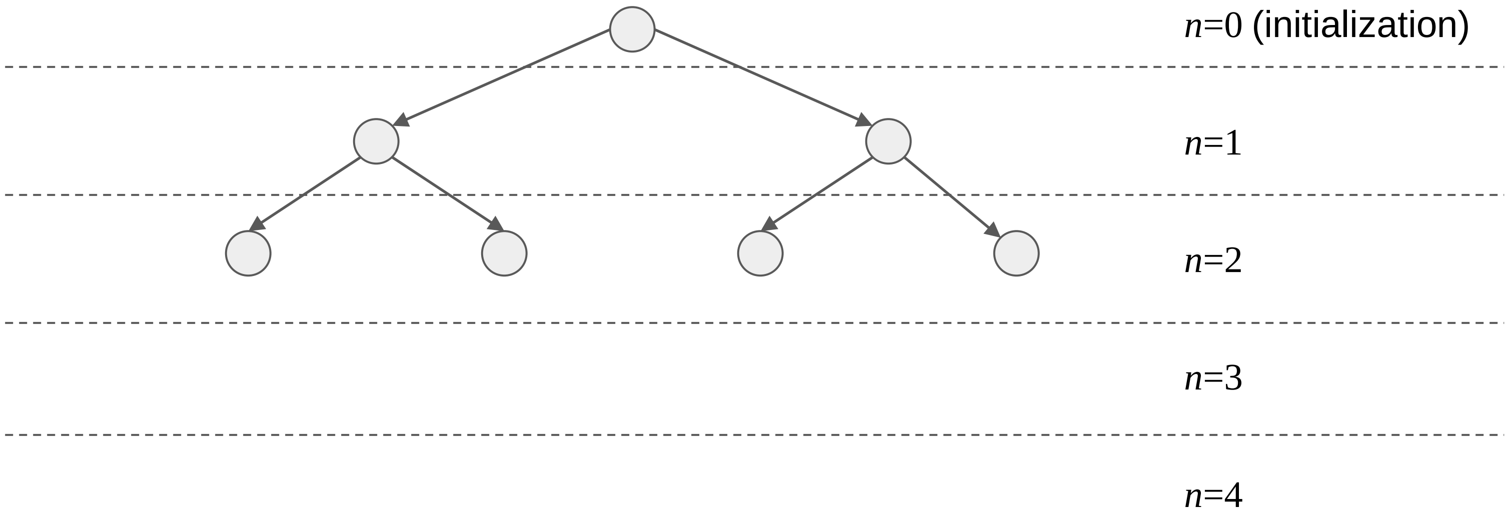
- **Step 1**. *Initialization*: Start with $n = 0$, generate the "root" configuration(s) in the set of candidate configurations, S.
- **Step 2**. *Extension*: Set $n = n + 1$, and using input data item x_n, extend all candidate configurations in S, and append these to S.
- **Step 3**. *Reduction*: Remove any candidate configurations from S which cannot be extended to an optimal configuration.
- **Step 4**. *Iteration*: if $n < N$, go back to Step 2.
- **Step 5**. *Select best*: Select an optimal configuration X^* from the remaining candidate configurations in S.

SDP exact: typical exhaustive computation graph

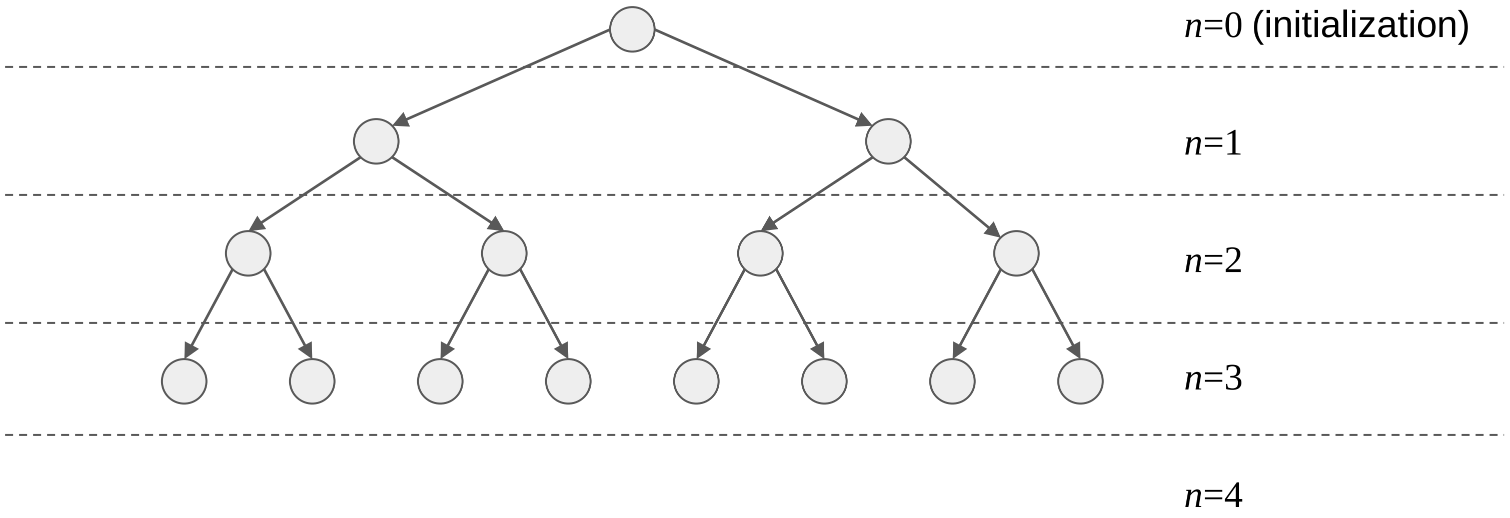
SDP exact: typical exhaustive computation graph



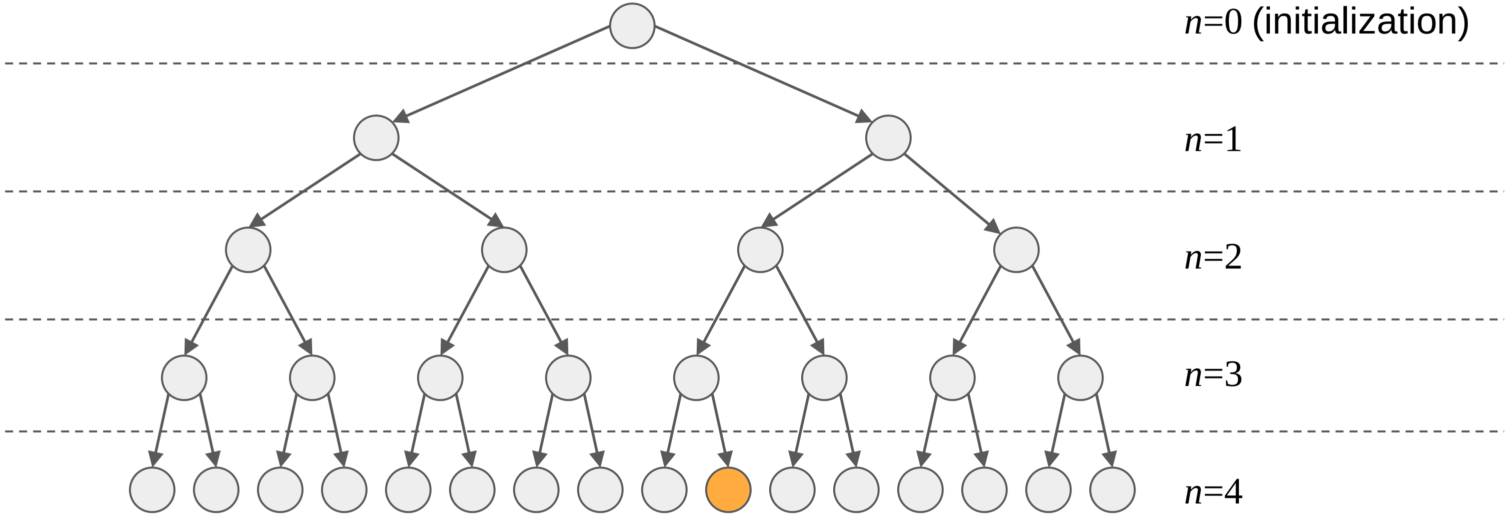
SDP exact: typical exhaustive computation graph



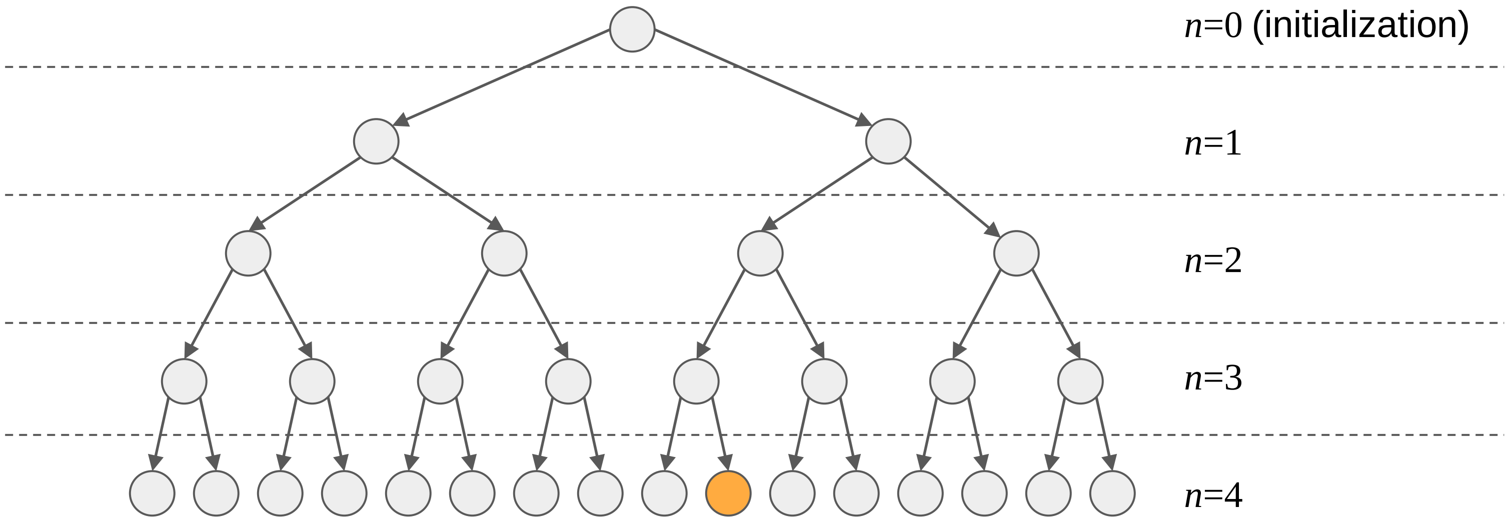
SDP exact: typical exhaustive computation graph



SDP exact: typical exhaustive computation graph



SDP exact: typical exhaustive computation graph



SDP exact: greedy

- From every exact configuration at stage $n < N$, there is exactly one possible optimal configuration obtained by extending the current exact configuration

SDP exact: greedy

- From every exact configuration at stage $n < N$, there is exactly one possible optimal configuration obtained by extending the current exact configuration
- Generate all possible extensions from stage n to stage $n + 1$, compute objective function, retain only the best one

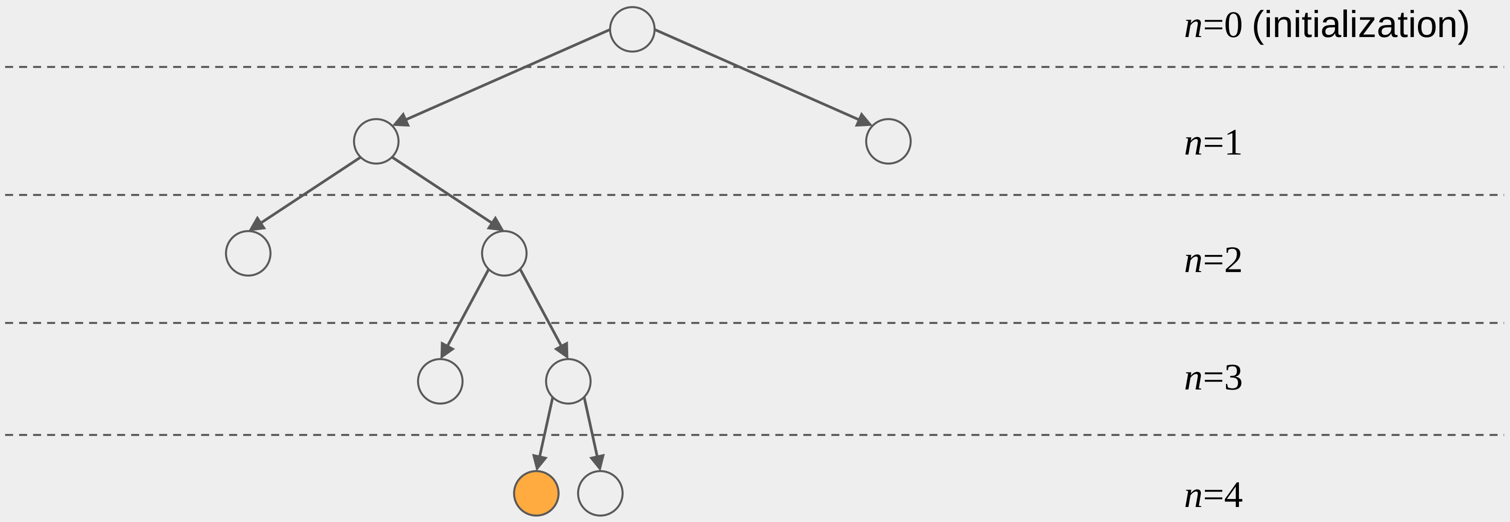
SDP exact: greedy

- From every exact configuration at stage $n < N$, there is exactly one possible optimal configuration obtained by extending the current exact configuration
- Generate all possible extensions from stage n to stage $n + 1$, compute objective function, retain only the best one
- **Efficiency**: avoid computing all configurations before selecting the best one and discarding sup-optimal solutions, only a small number of extensions at each stage; only one solution remaining (no selection required)

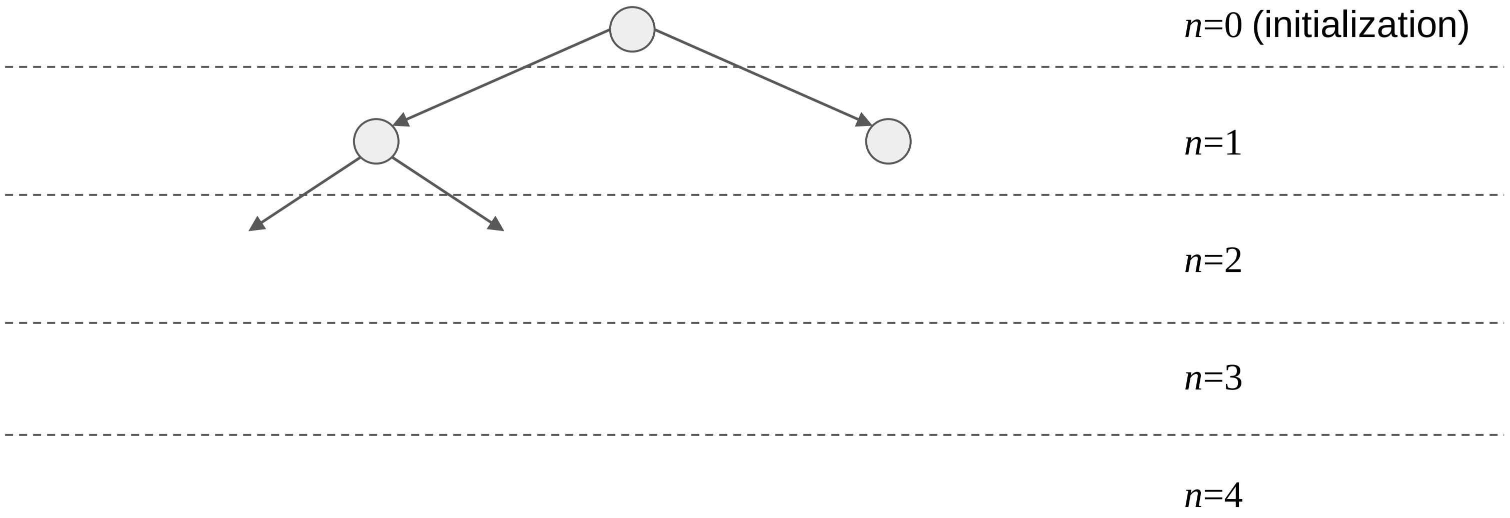
SDP exact: greedy

- From every exact configuration at stage $n < N$, there is exactly one possible optimal configuration obtained by extending the current exact configuration
- Generate all possible extensions from stage n to stage $n + 1$, compute objective function, retain only the best one
- **Efficiency**: avoid computing all configurations before selecting the best one and discarding sup-optimal solutions, only a small number of extensions at each stage; only one solution remaining (no selection required)
- **Complexity**: typically $O(Nk)$

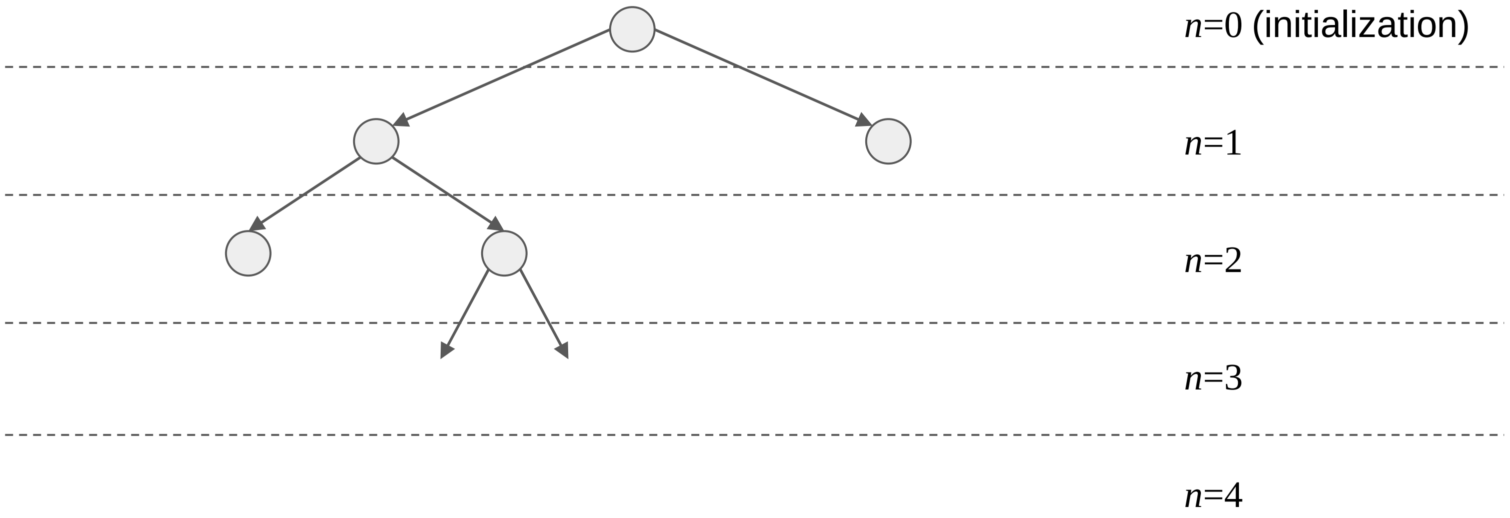
SDP exact: typical greedy computation graph



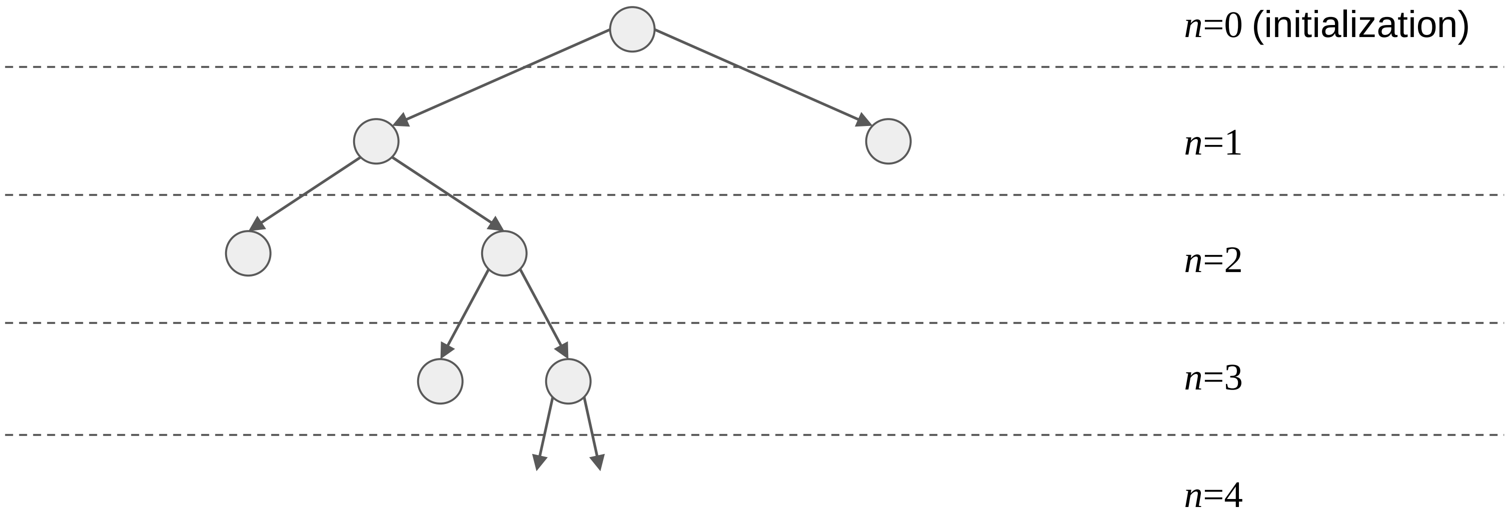
SDP exact: typical greedy computation graph



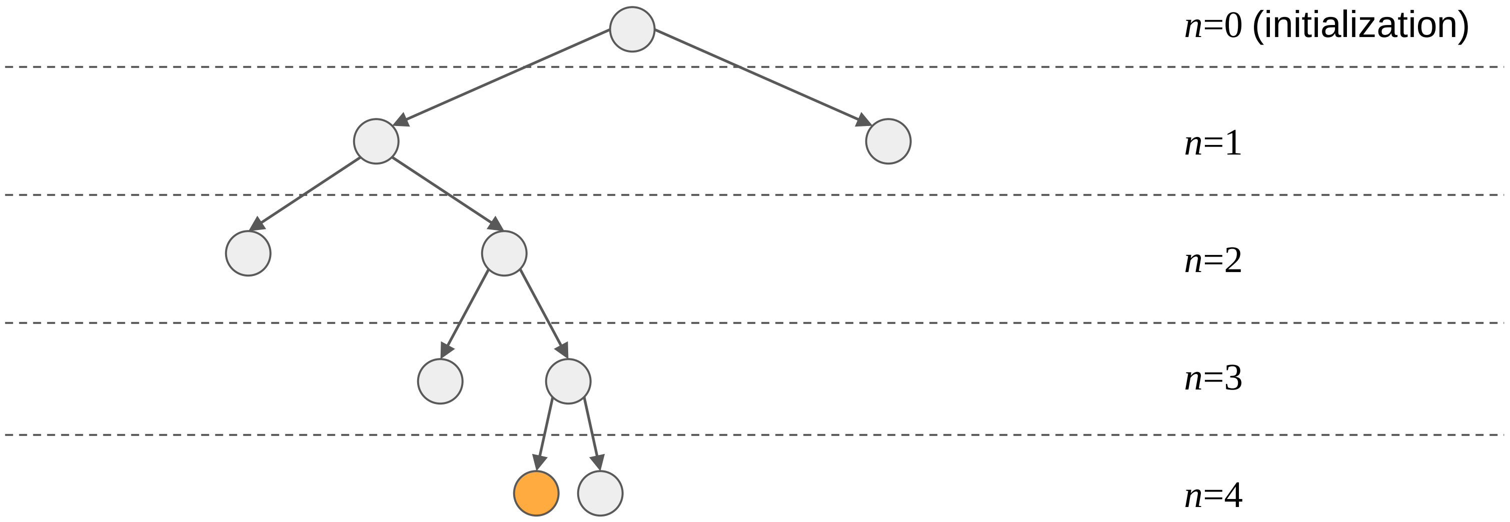
SDP exact: typical greedy computation graph



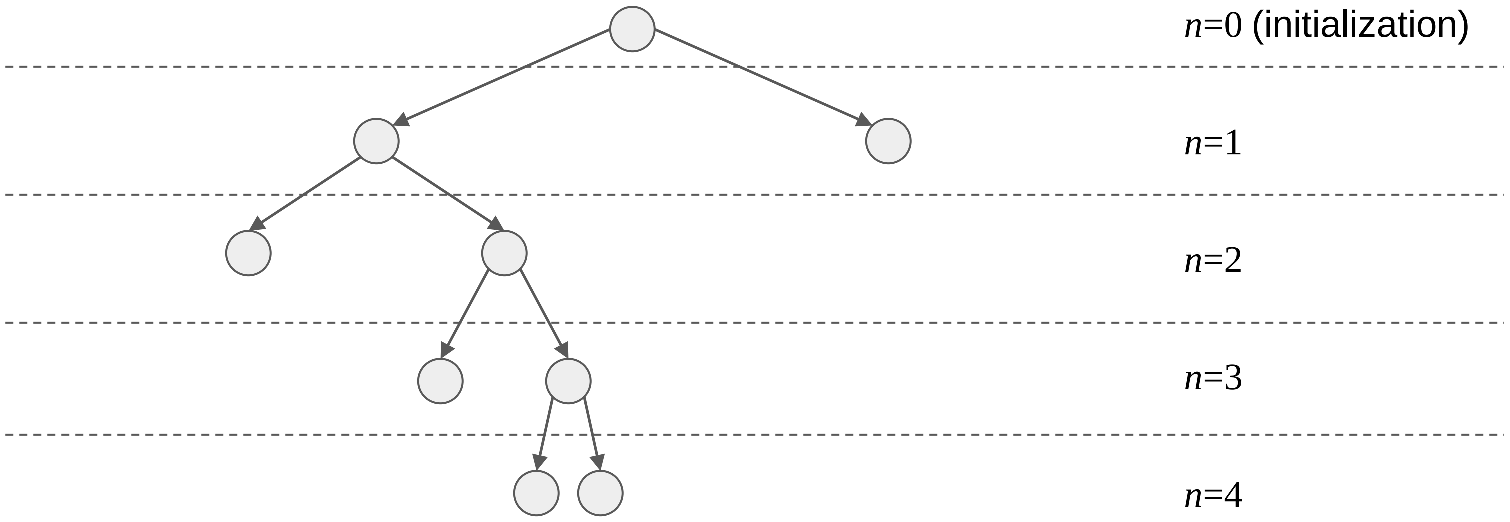
SDP exact: typical greedy computation graph



SDP exact: typical greedy computation graph



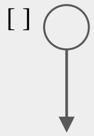
SDP exact: typical greedy computation graph



Insertion sort

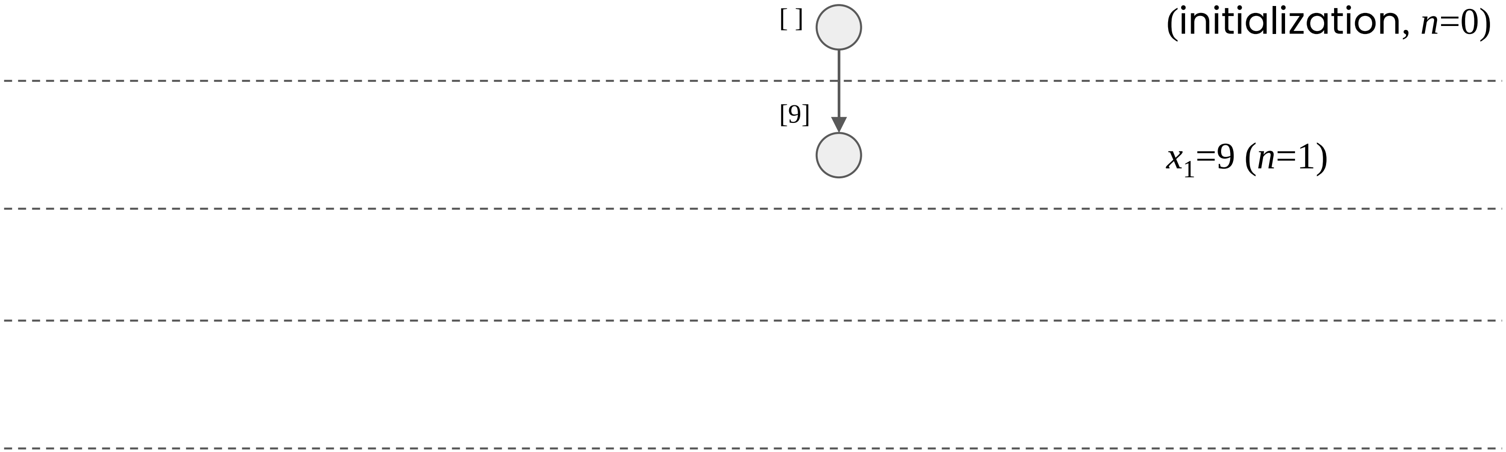
- A sorting algorithm, like selection sort
- Let’s say you want to sort an array A with four elements
- A = [9, 1, 7, 3]

SDP exact: O(N^2) insertion sort

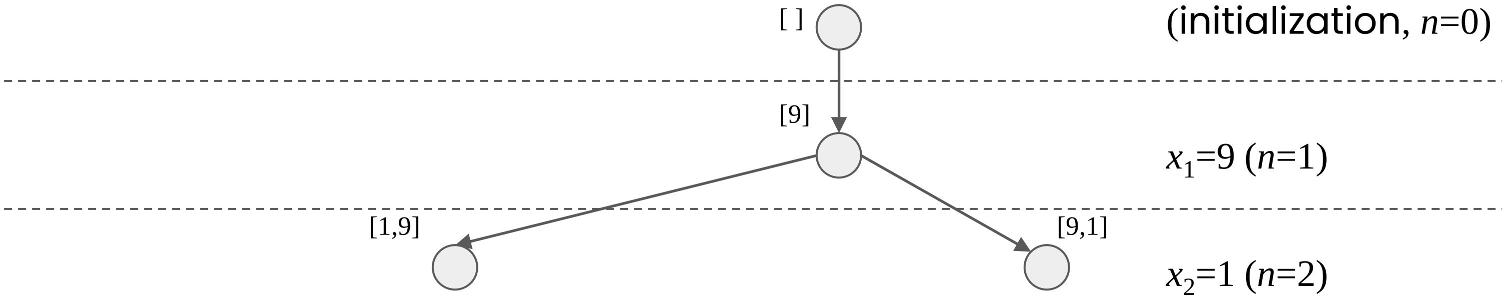


(initialization, n=0)

SDP exact: O(N^2) insertion sort



SDP exact: O(N^2) insertion sort

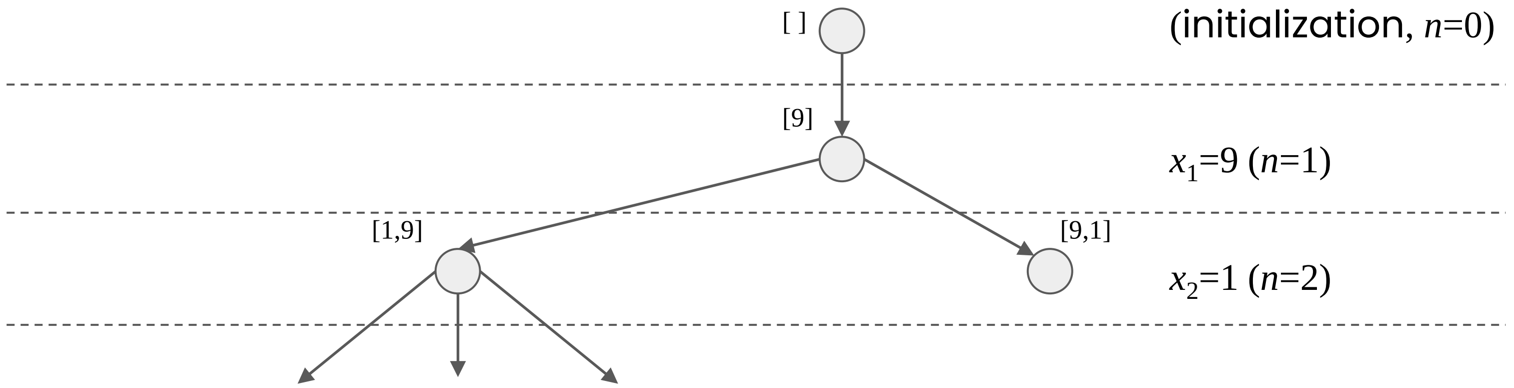


(initialization, n=0)

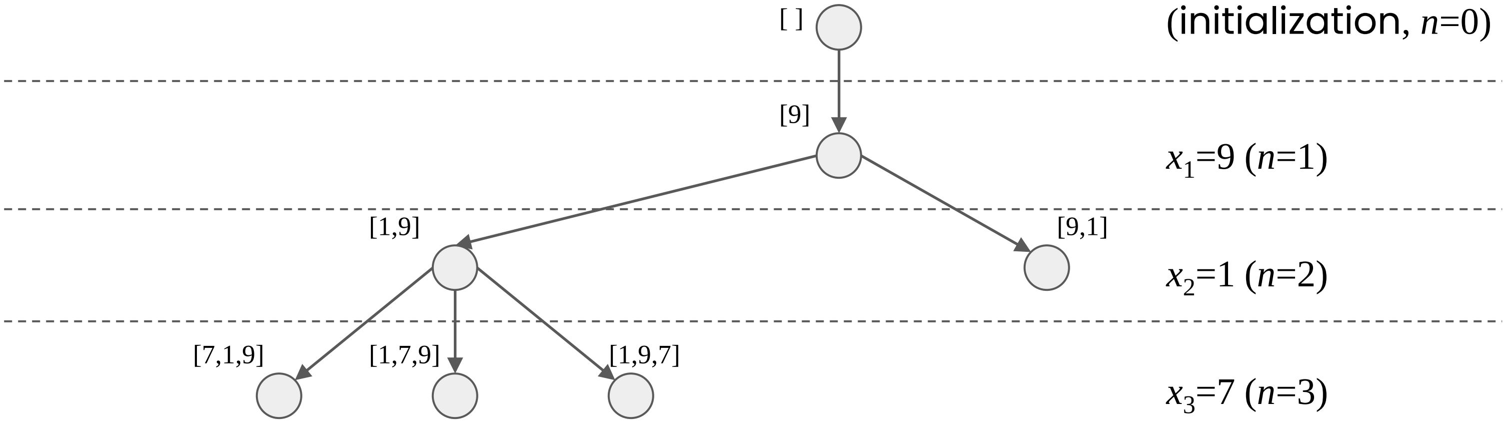
x_1=9 (n=1)

x_2=1 (n=2)

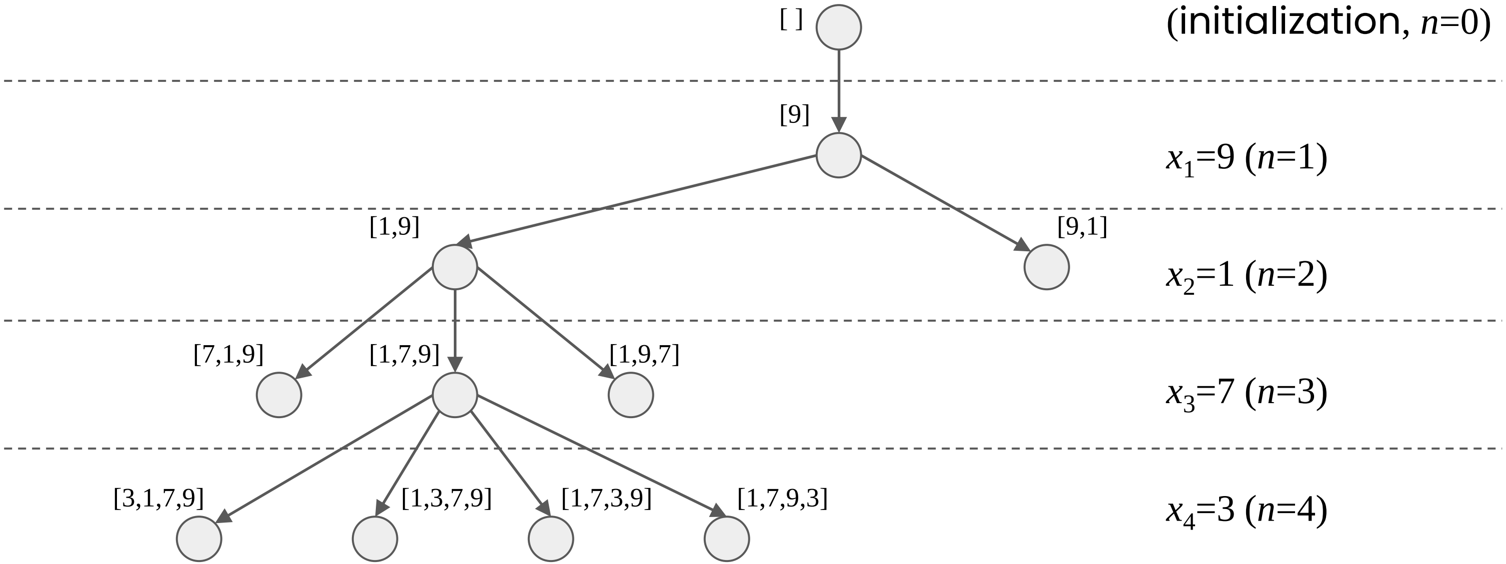
SDP exact: O(N^2) insertion sort



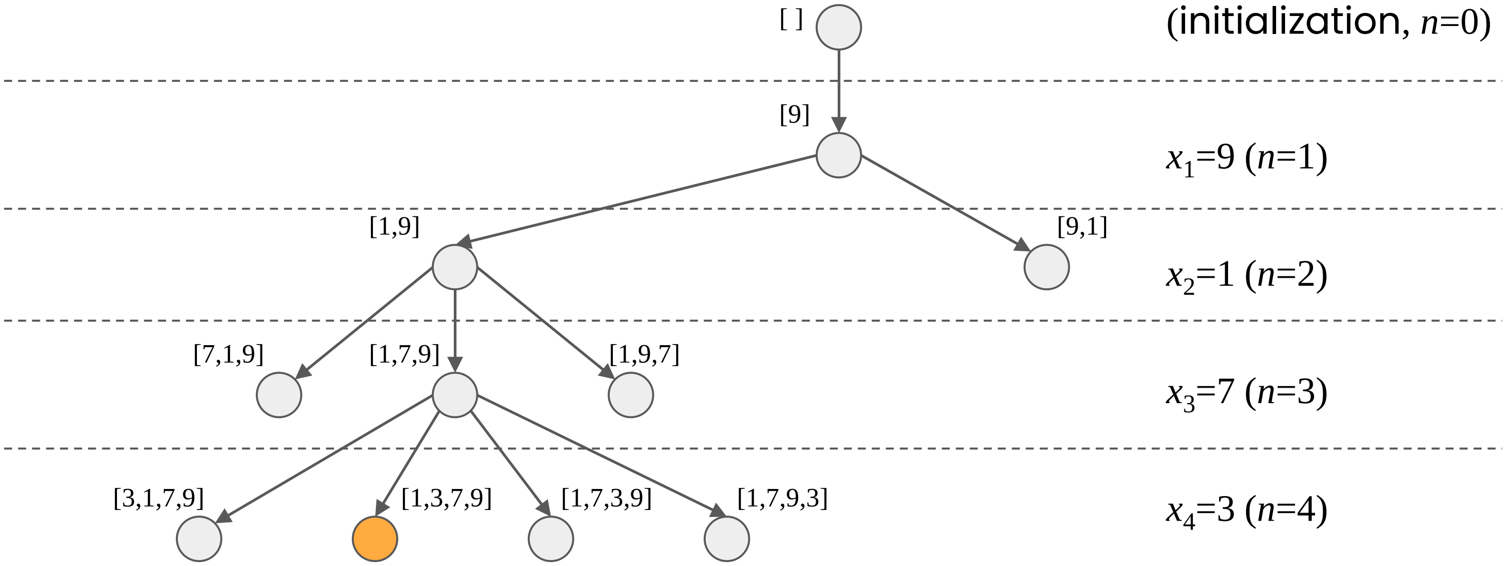
SDP exact: O(N^2) insertion sort



SDP exact: O(N^2) insertion sort



SDP exact: O(N^2) insertion sort



References and further reading

- **MLSP**, Section 2.6
- **CLRS**, Chapter 21