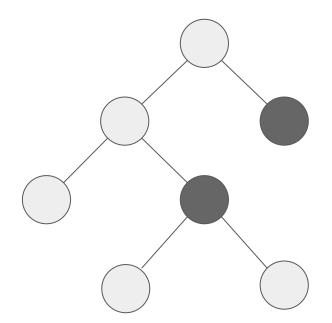
Example of Tree Tilling using Dynamic Programming

Compilers Course MIEIC

The Problem

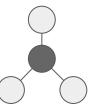
- Given an input tree and a set of subtrees (each one with an associated cost) determine the tree tilling with minimum cost
- Assumptions: the input tree can be fully tilled with the considered set of subtrees
- Connection with the Instruction selection problem:
 - Input tree is a tree-based low-level IR of a code statement (or region of code)
 - Set of subtrees represents the IR tree patterns
 - Cost of each subtree represent the cost of each instruction (e.g., execution clock cycles). If the goal is to find the tree tilling with the minimum number of subtrees, one case use a cost of 1 for each subtree









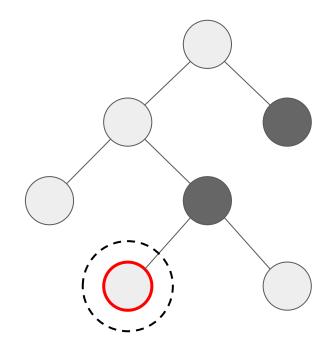


(C) Cost: 4

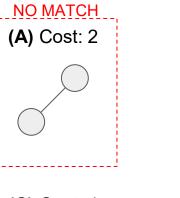


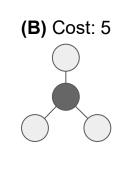
(D) Cost: 3





Sub Trees



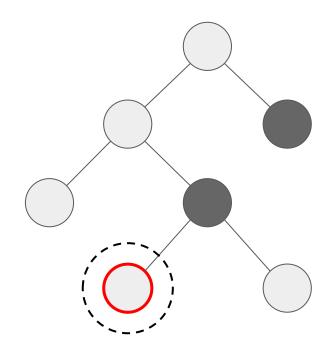


(C) Cost: 4

(D) Cost: 3







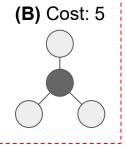






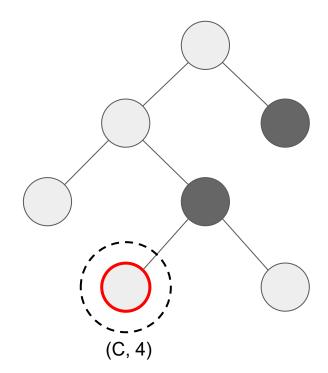


NO MATCH



(D) Cost: 3

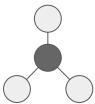




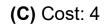


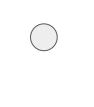






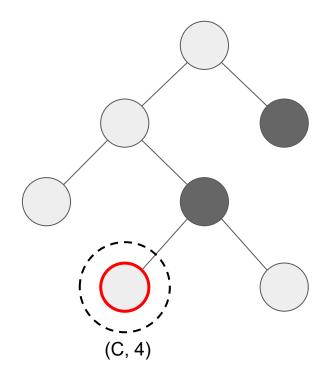
MATCH





(D) Cost: 3





Sub Trees

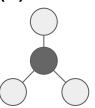








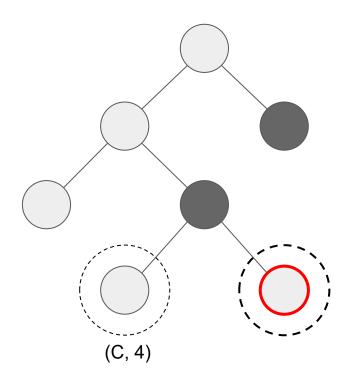
(B) Cost: 5



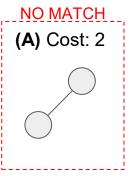
NO MATCH

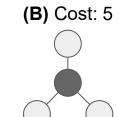






Sub Trees

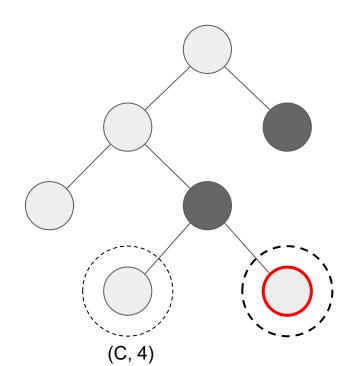




- (C) Cost: 4

(D) Cost: 3





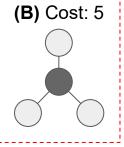






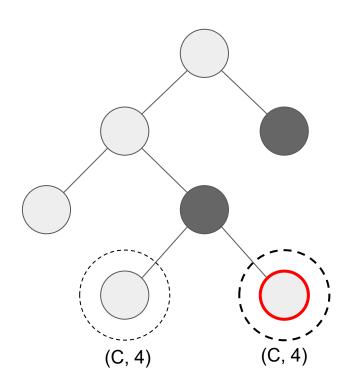


NO MATCH



(D) Cost: 3

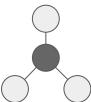








(B) Cost: 5



MATCH

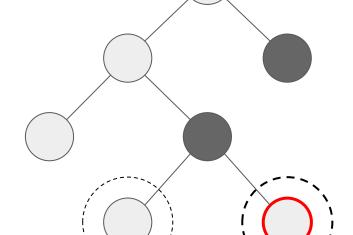




(D) Cost: 3



(C, 4)



(C, 4)

Sub Trees

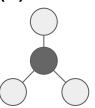




(C) Cost: 4



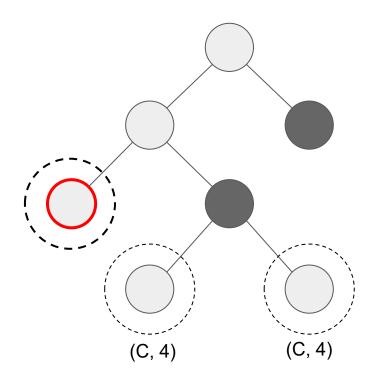
(B) Cost: 5

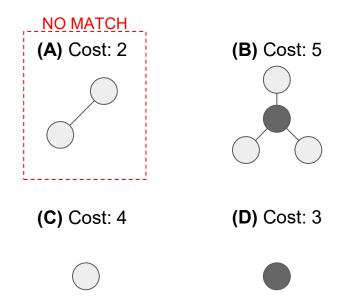


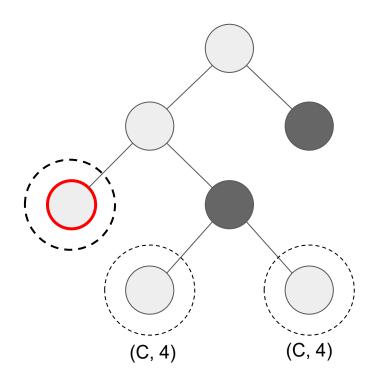
NO MATCH











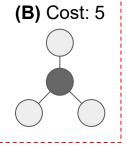






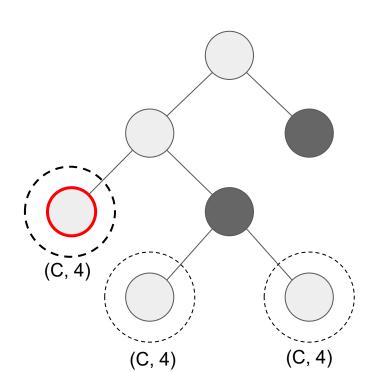


NO MATCH



(D) Cost: 3

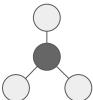






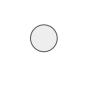






MATCH





(D) Cost: 3



(C, 4)

(C, 4)

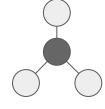
(C, 4)

Sub Trees









(B) Cost: 5

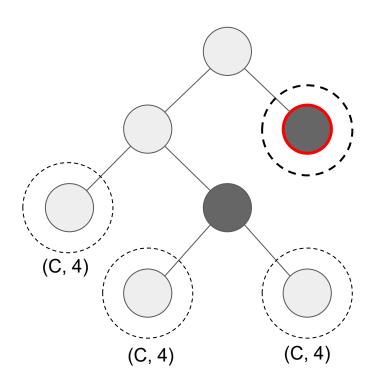
(C) Cost: 4

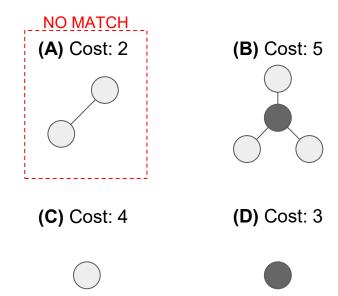


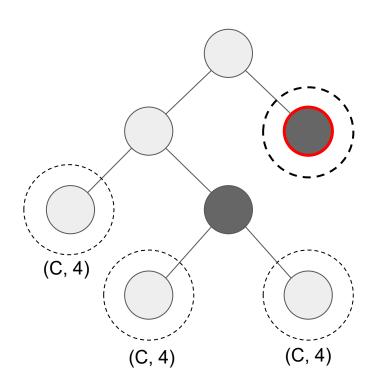
NO MATCH











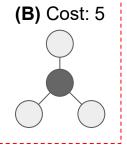








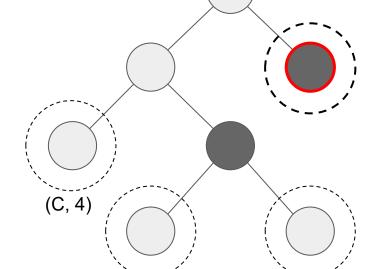
NO MATCH



(D) Cost: 3



(C, 4)

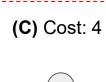


(C, 4)

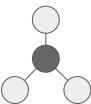
Sub Trees







(B) Cost: 5



NO MATCH



(D) Cost: 3



(C, 4)

(C, 4)

(C, 4)

(D, 3)

Sub Trees

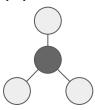
(A) Cost: 2



(C) Cost: 4



(B) Cost: 5



(D) Cost: 3

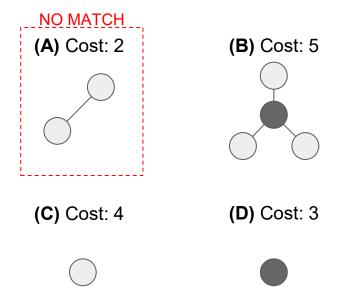
MATCH



(C, 4)

(D, 3) (C, 4)

(C, 4)



(C, 4)

(D, 3) (C, 4)

(C, 4)

Sub Trees

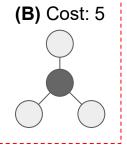




(C) Cost: 4



NO MATCH



(D) Cost: 3



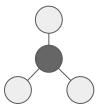
(D, 3)

Sub Trees



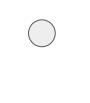






NO MATCH

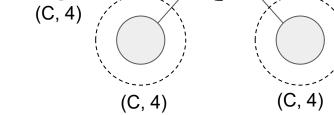


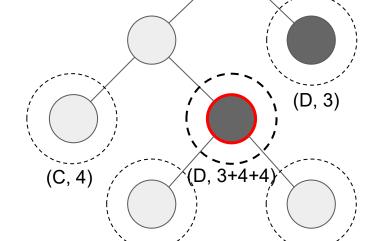


(D) Cost: 3









(C, 4)

(C, 4)

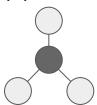








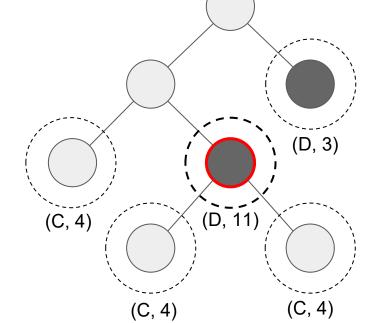
(B) Cost: 5



MATCH







Sub Trees

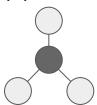




(C) Cost: 4



(B) Cost: 5

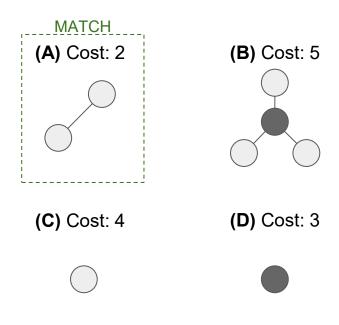


MATCH

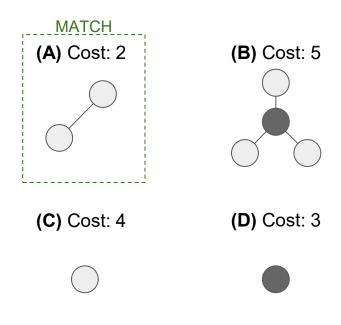


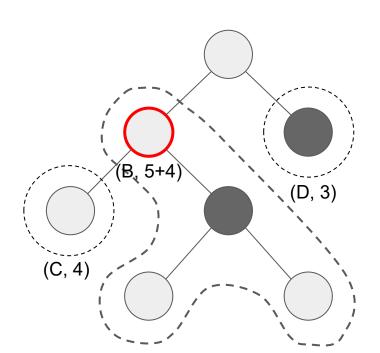


(A, 2+11) (D, 3) (D, 11) (C, 4)(C, 4)

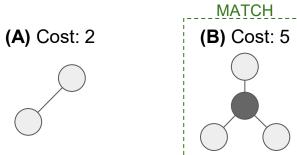


(A, 13) (D, 3) (D, 11) (C, 4)(C, 4)





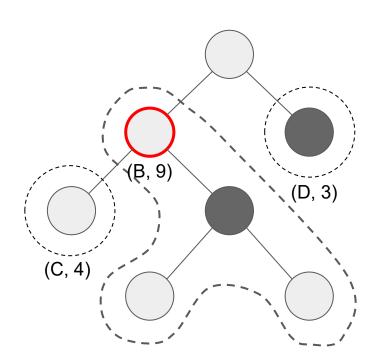
Sub Trees



(C) Cost: 4



(D) Cost: 3



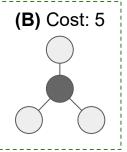






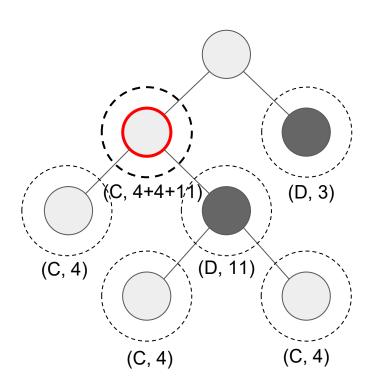


MATCH

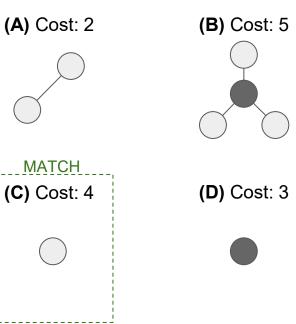


(D) Cost: 3

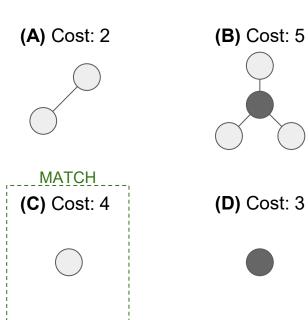


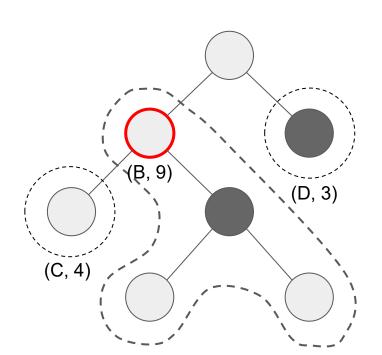






(C, 19),' (D, 3) (C, 4)(D, 11) (C, 4)(C, 4)





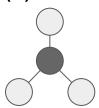








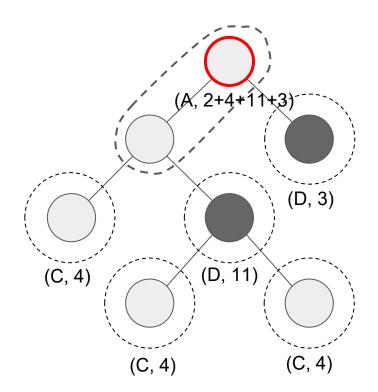
(B) Cost: 5

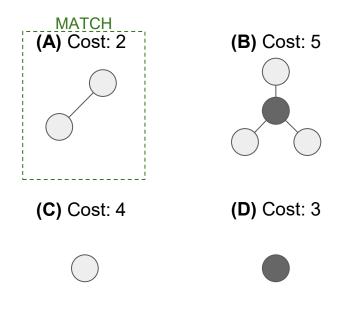


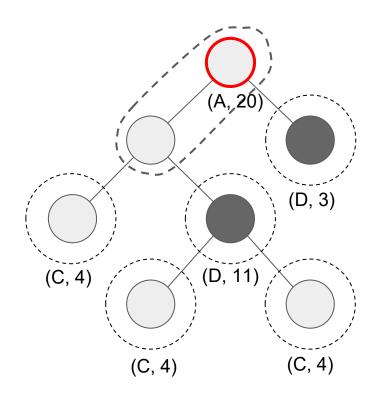
NO MATCH

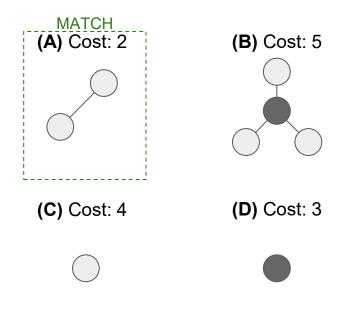


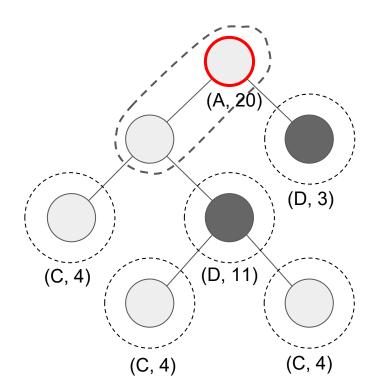








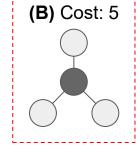




Sub Trees







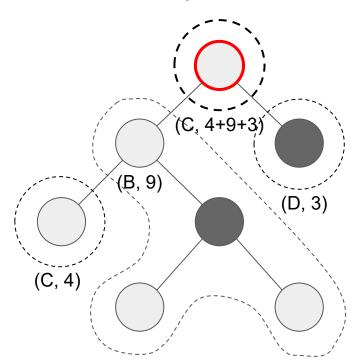
NO MATCH

(C) Cost: 4



(D) Cost: 3

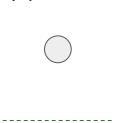




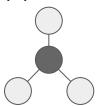






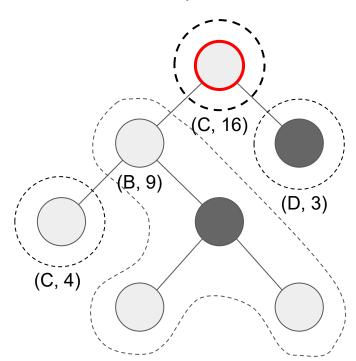


(B) Cost: 5



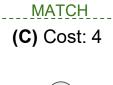
(D) Cost: 3

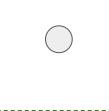




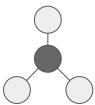






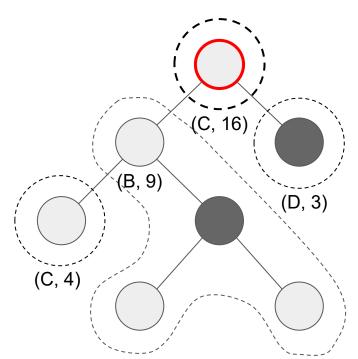


(B) Cost: 5



(D) Cost: 3





Sub Trees

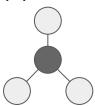




(C) Cost: 4



(B) Cost: 5



NO MATCH



Input Tree (C, <mark>16</mark>) / (B, 9) (D, 3) (C, 4)

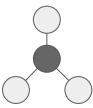
Optimum cost: 16

Sub Trees









(C) Cost: 4



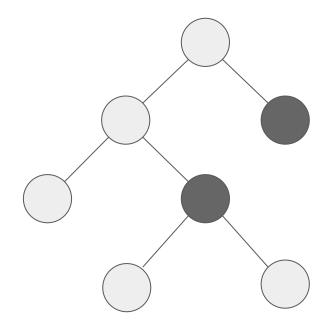
(D) Cost: 3



Questions

- Time complexity for an algorithm implementing tree tilling with dynamic programing?
- Result of applying Maximal Munch to this example=?

Using Maximal Munch

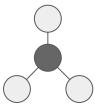


Sub Trees







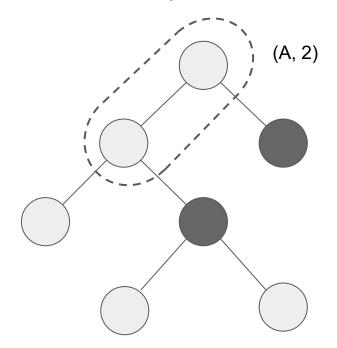


(C) Cost: 4



(D) Cost: 3



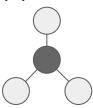


Sub Trees





(B) Cost: 5

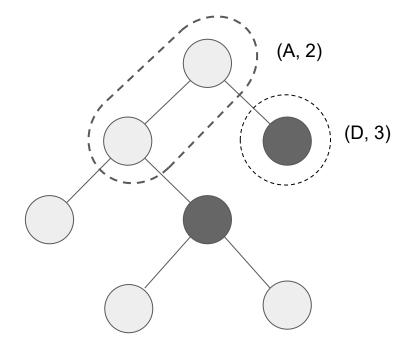


(C) Cost: 4



(D) Cost: 3



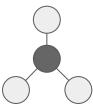


Sub Trees





(B) Cost: 5



(C) Cost: 4





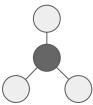
Input Tree (A, 2)(D, 3) (C, 4)

Sub Trees









(C) Cost: 4





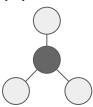
Input Tree (A, 2)(D, 3) (D, 3) (C, 4)

Sub Trees









(C) Cost: 4





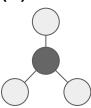
Input Tree (A, 2)(D, 3) (D, 3) (C, 4) (C, 4)

Sub Trees







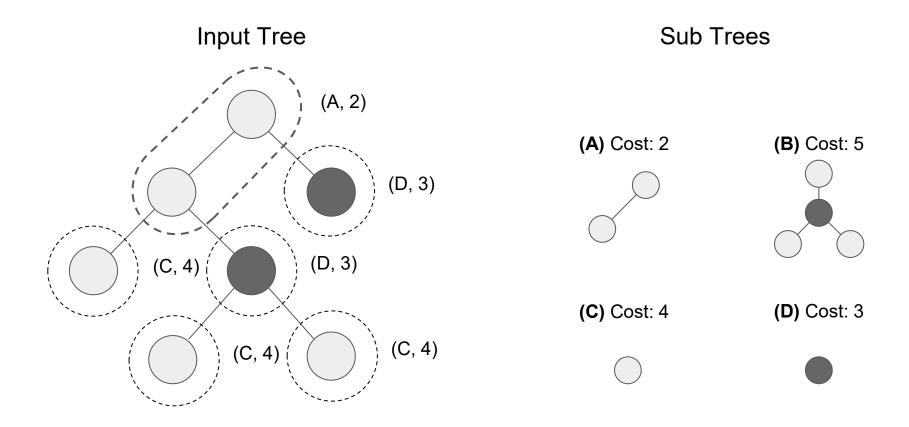


(C) Cost: 4



(D) Cost: 3





Cost: 20

Conclusion

- Dynamic programming is an efficient technique for solving the instruction selection problem when addressing intermediate representations using trees
- It gives the optimum selection, given a fixed cost per instruction, and the goal to minimize the cost based on the sum of all the costs of the selected instructions