Nonserial Decision Diagrams

John Hooker
Carnegie Mellon University

ICS Conference
March 2025

Carnegie Mellon University
Tepper School of Business

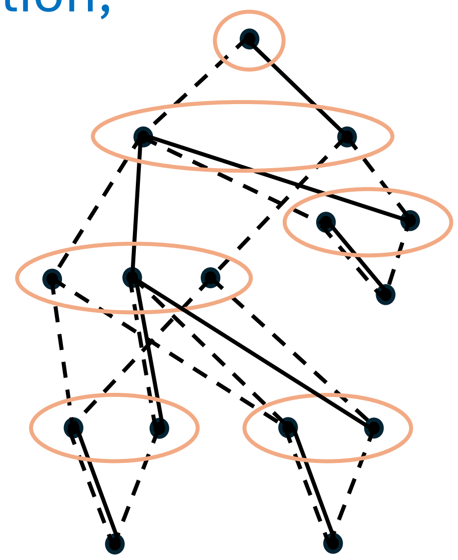
Why Nonserial DDs?

- They exploit structure of problem instances whose variables **partially decouple**.

- They combine **nonserial dynamic programming** ideas with **DD solution technology** – reduction, relaxation, restriction, flow models, etc.

- They can be **dramatically smaller** than serial DDs.

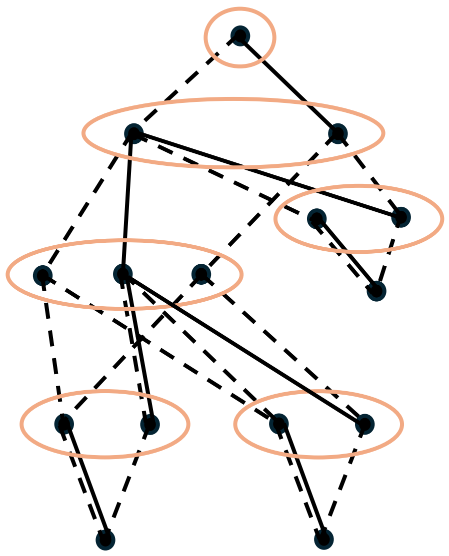
- Reduction in **compilation time** is **even greater**.



Why Nonserial DDs?

When exact DDs are **smaller**....

- **Relaxed DDs** of a given size provide **tighter bounds**.
- **Restricted DDs** of a given size are more likely to yield **feasible solutions**.
- **Flow models** are more likely to be **tractable**.



Example: Set packing

Find a maximum subcollection of sets
in which no two sets have common elements.

{A, C }
{ C,D}
{A,B }
{ C }
{A }
{ B, D}

Example: Set packing

Find a maximum subcollection of sets
in which no two sets have common elements.

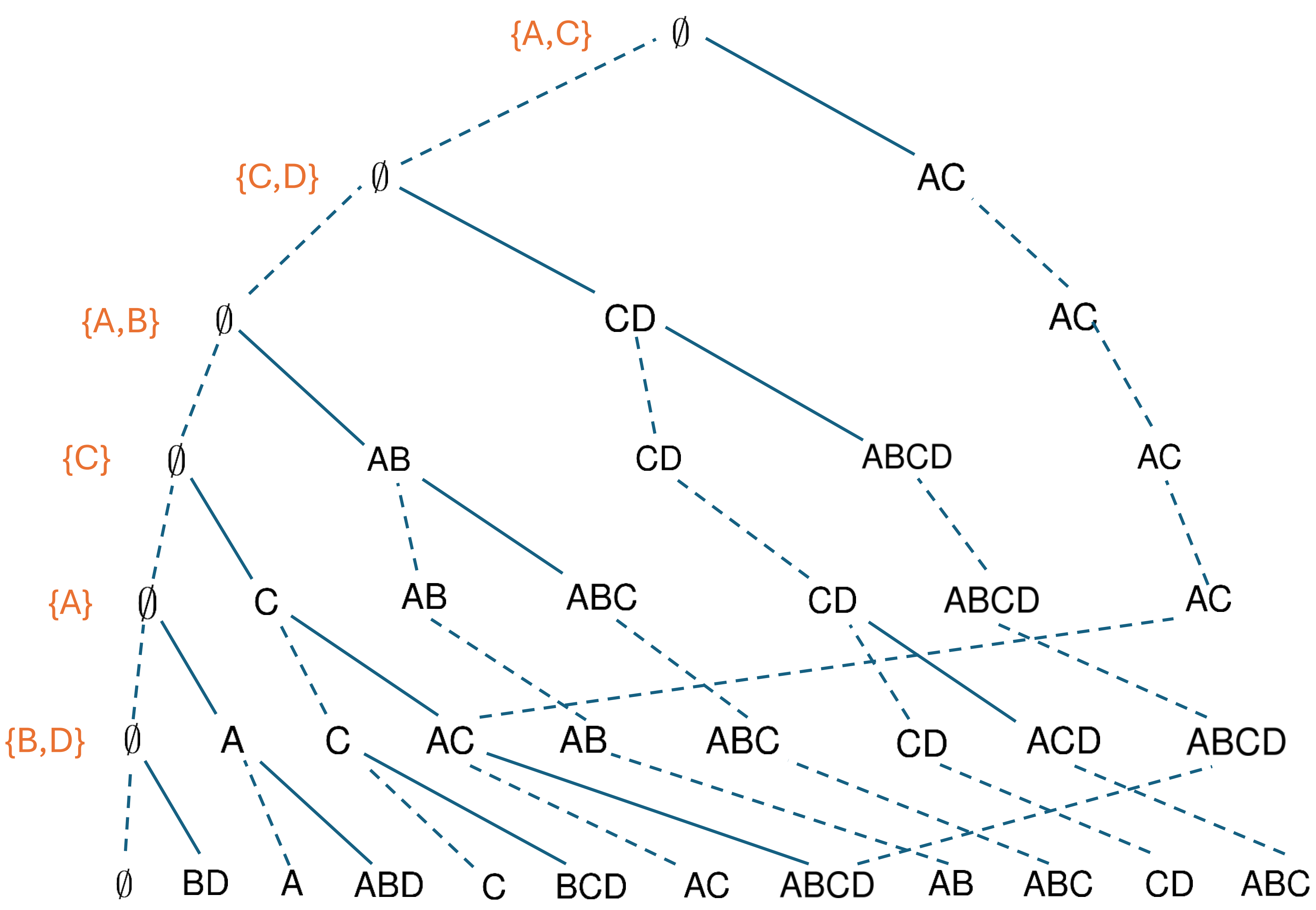
{A, C }
{ C,D}
{A,B }
{ C }
{A }
{ B, D}

solution

Serial DD
for a set packing
problem instance

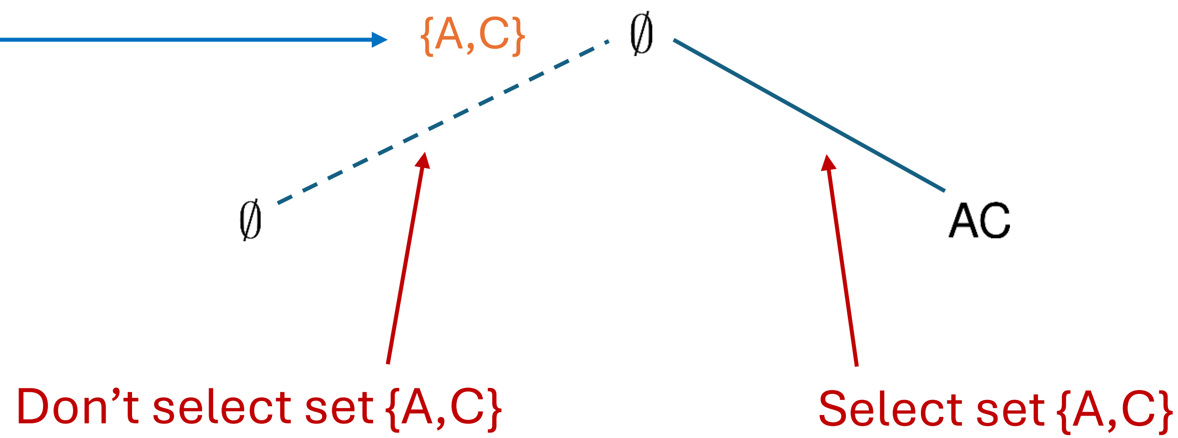
Layers
correspond
to selection
decisions
for each set.

Variables
indicate the
decisions.

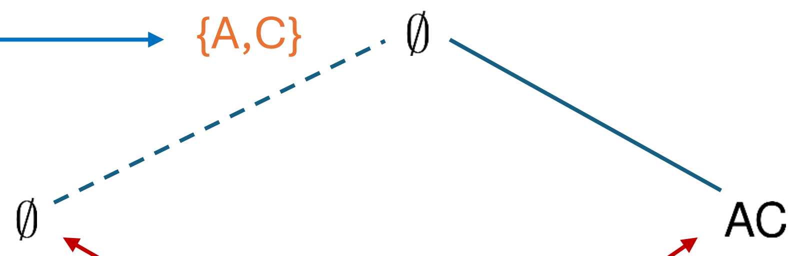


Carnegie Mellon University
Tepper School of Business

Decide whether
to select set {A,C}

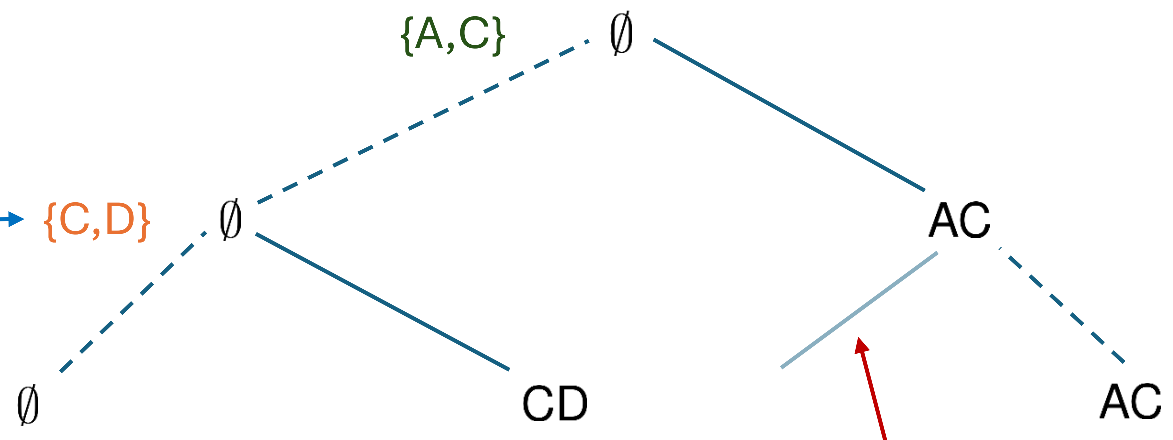


Decide whether to select set {A,C}



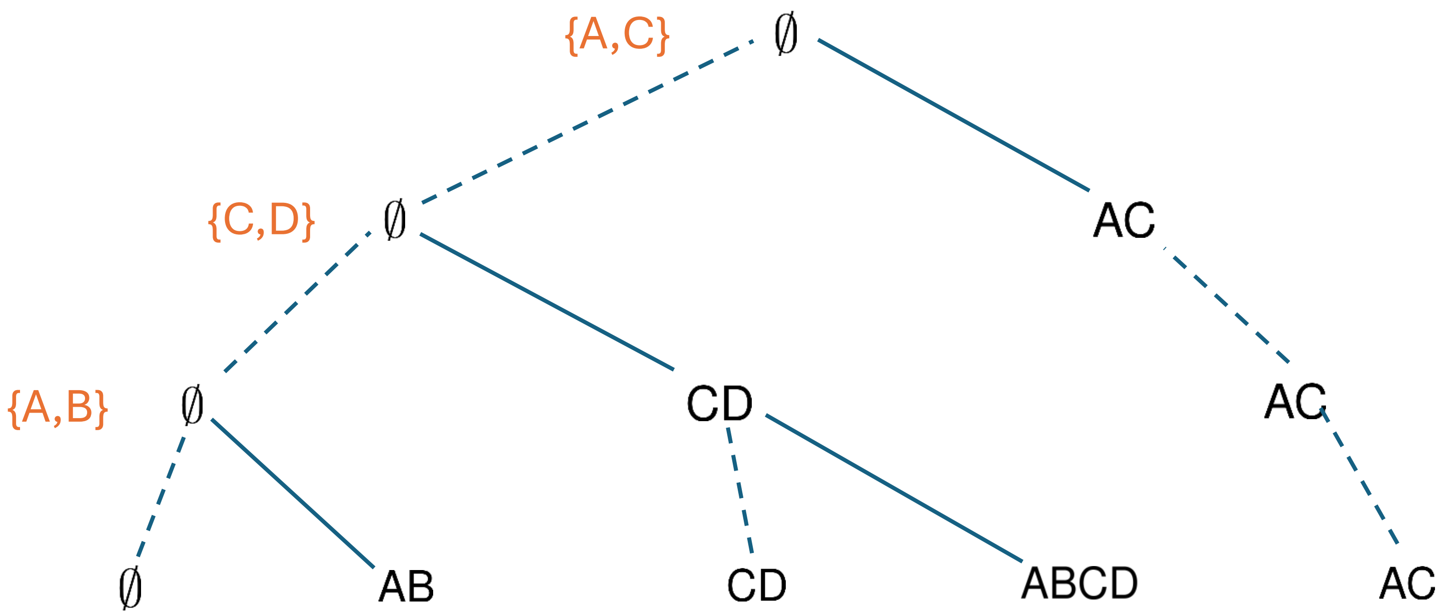
State consists of elements in sets so far selected

Decide whether to select set {C,D}



Cannot select {C,D} because C is already in the state

Serial DD
for a set packing
problem instance

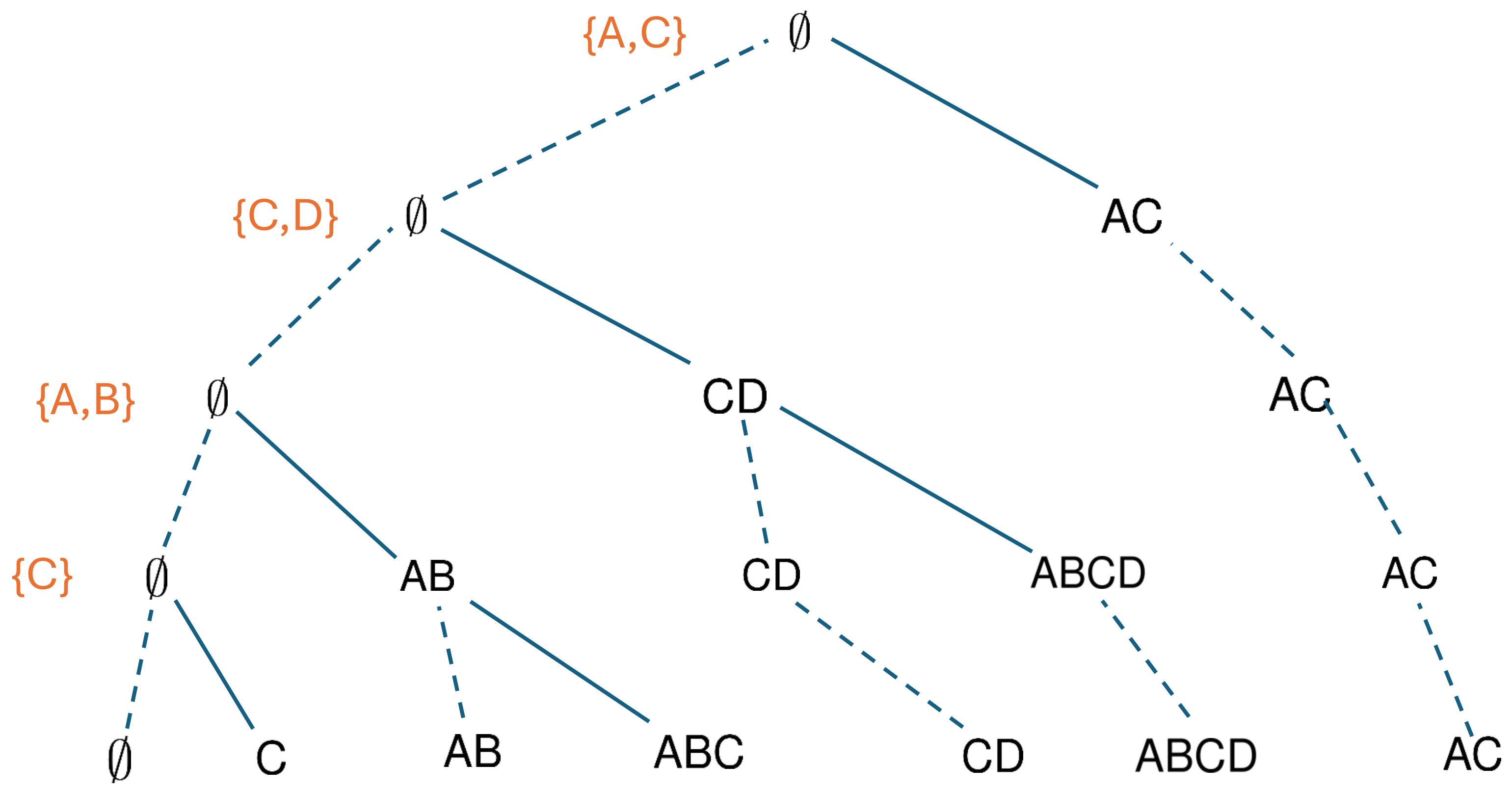


Decide whether
to select set {A,B}

Serial DD

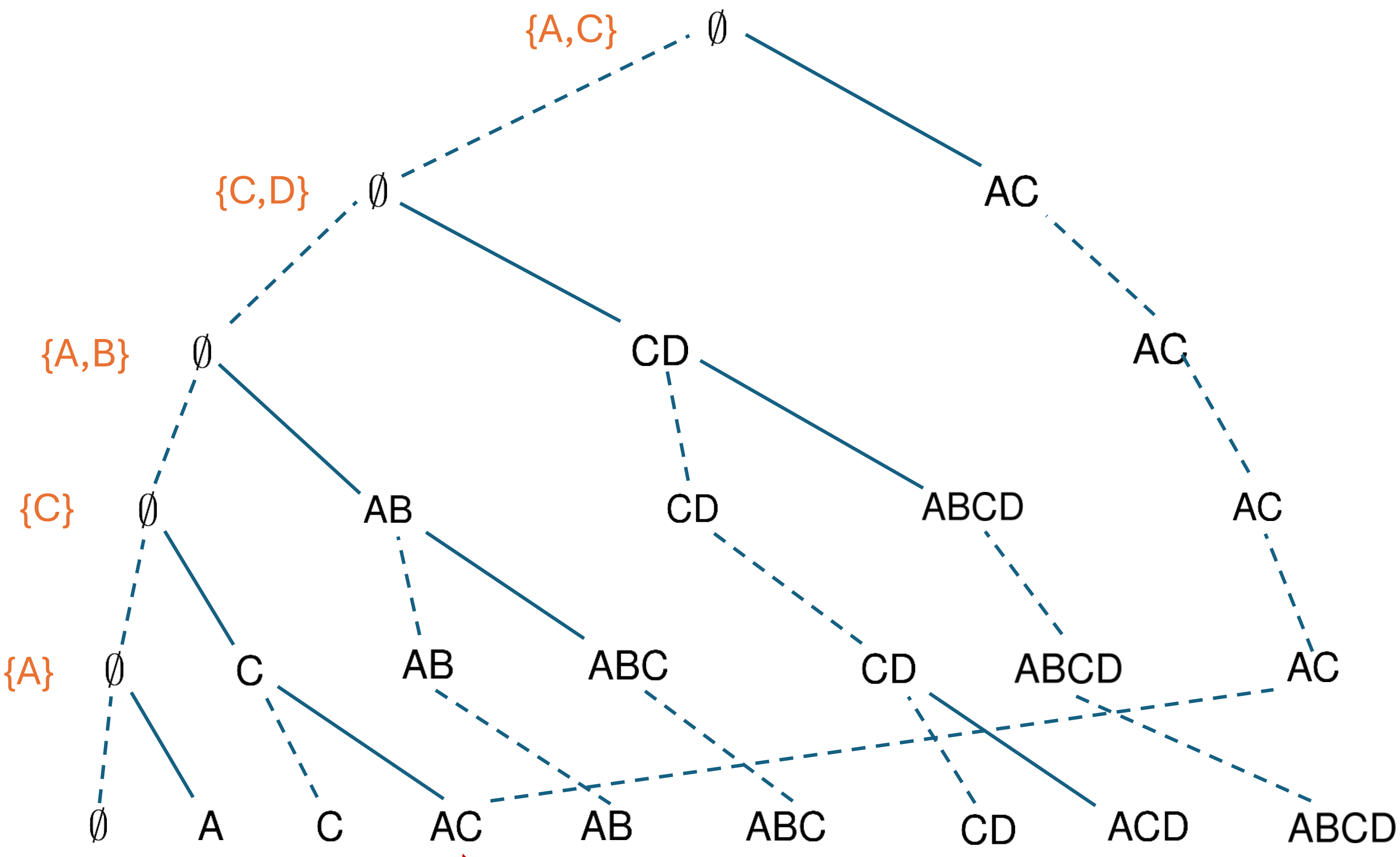
for a set packing problem instance

Decide whether to select set {C}



Serial DD
for a set packing
problem instance

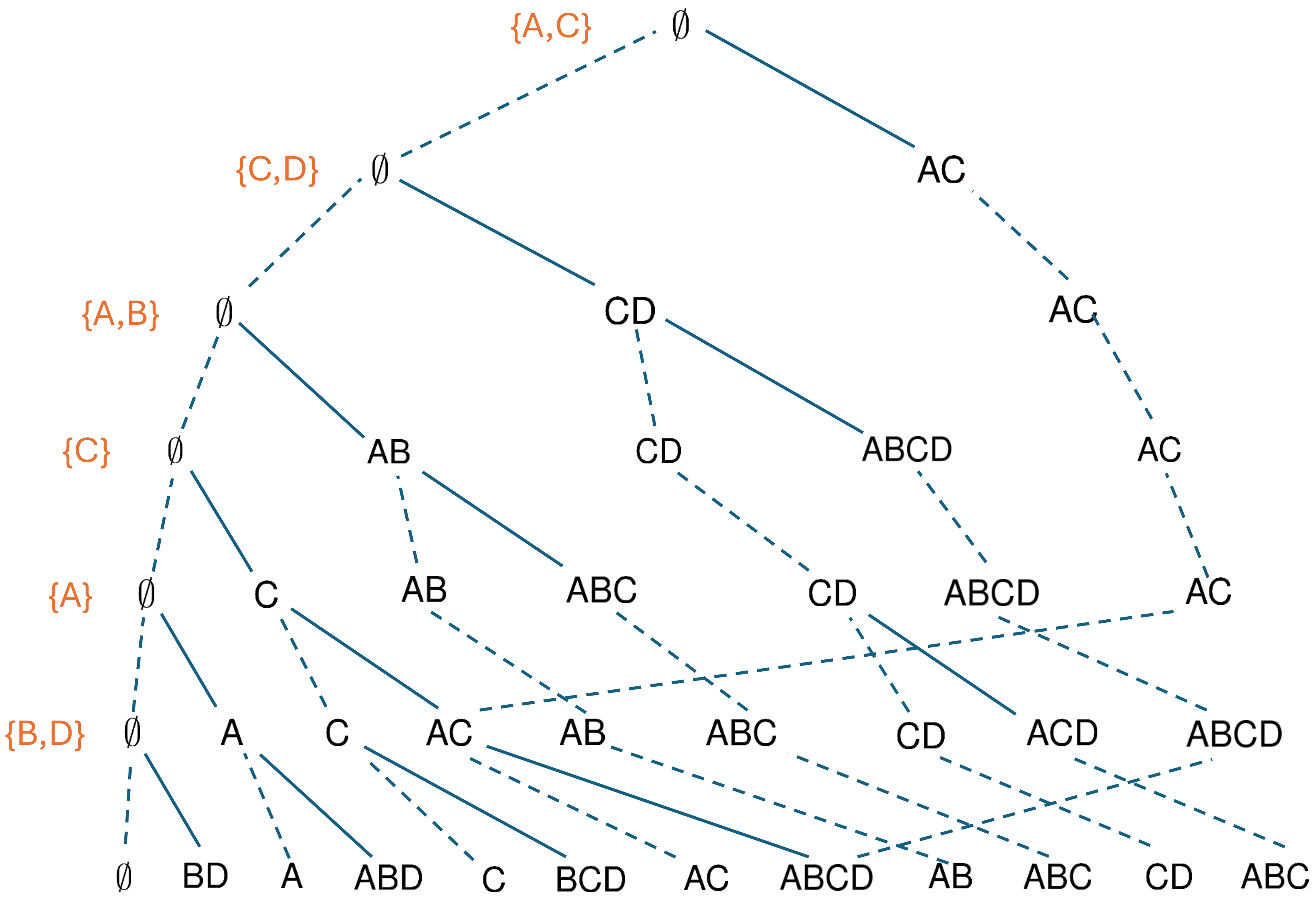
Decide whether
to select set {A}



DD is not a tree because branches can terminate in
the same state. This normally happens quite often.

Serial DD
for a set packing
problem instance

Decide whether
to select set {B,D}

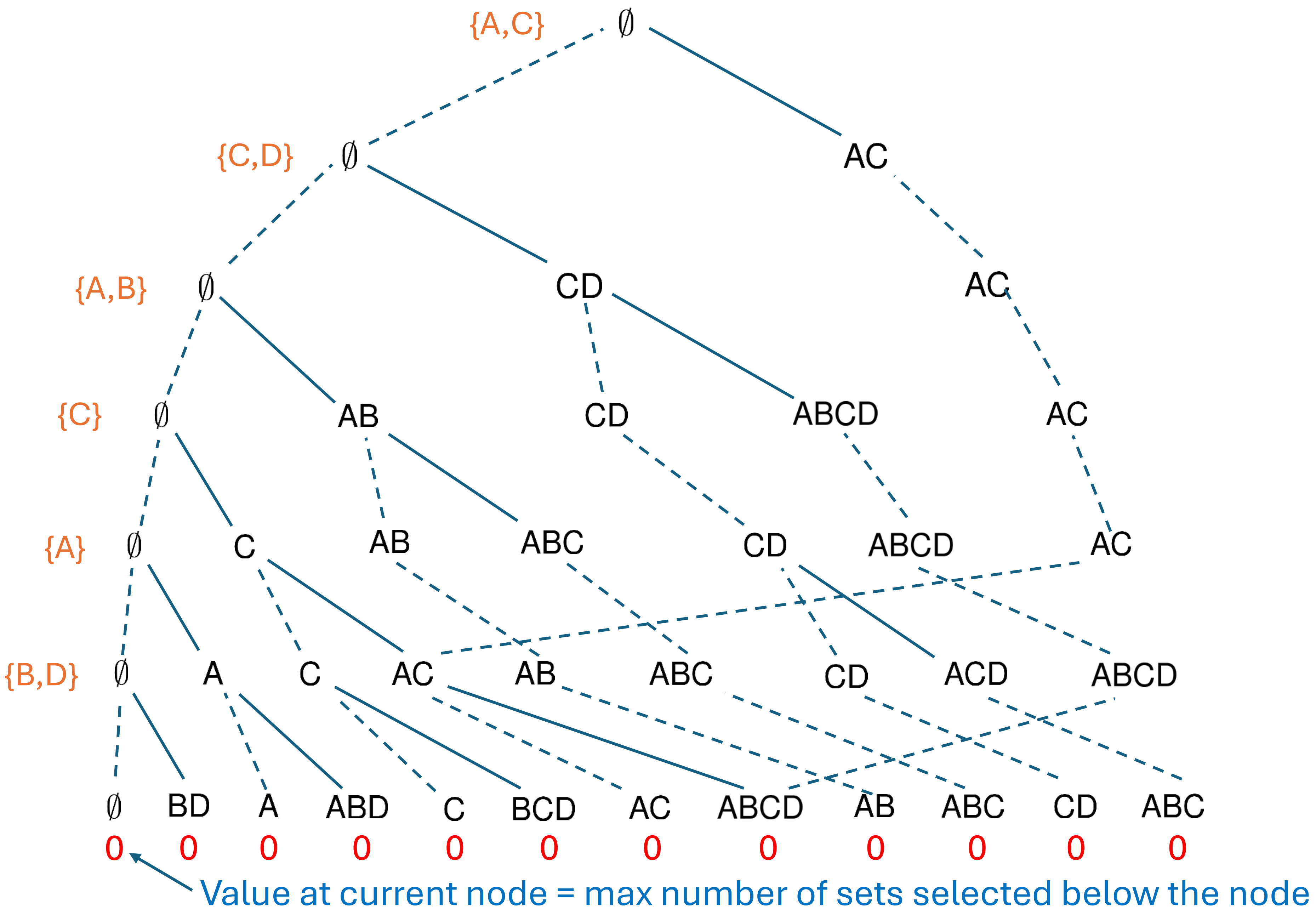


DD has 39 nodes

Serial DD

for a set packing problem instance

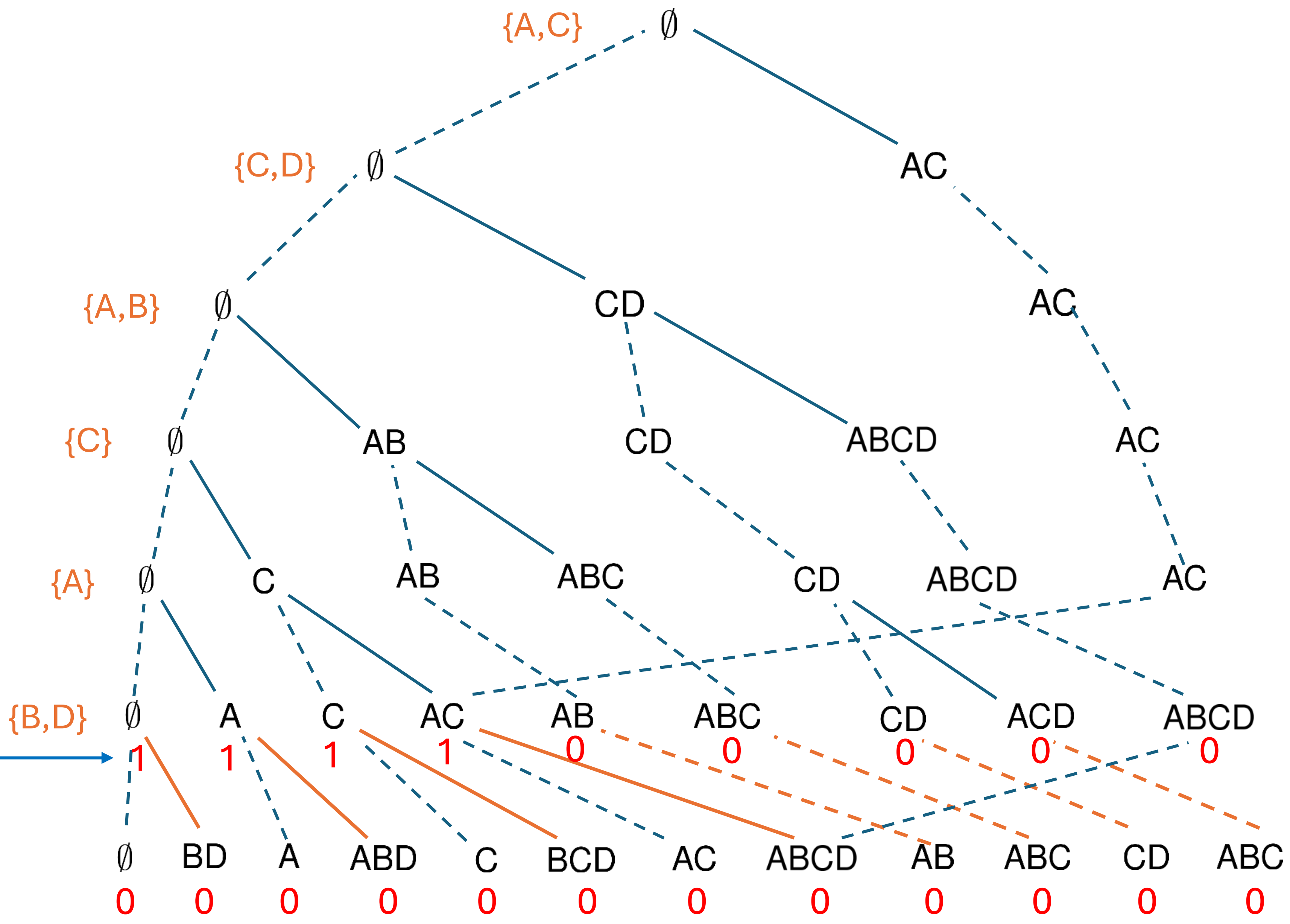
Now find an optimal solution recursively, using a backward pass, as in dynamic programming.



Carnegie Mellon University
Tepper School of Business

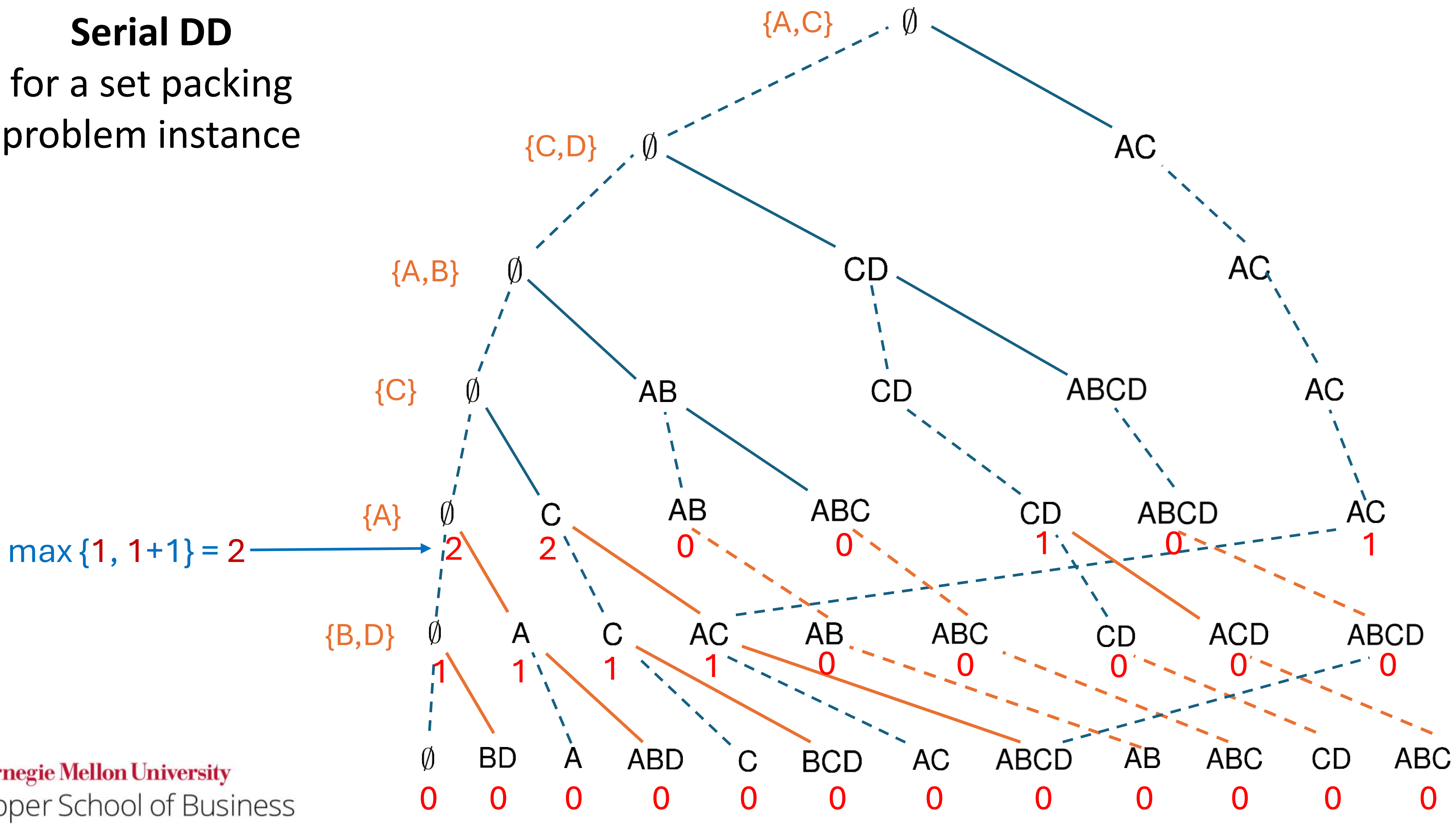
Serial DD
for a set packing problem instance

max {0, 0+1} = 1
Mark optimal decision with orange arc

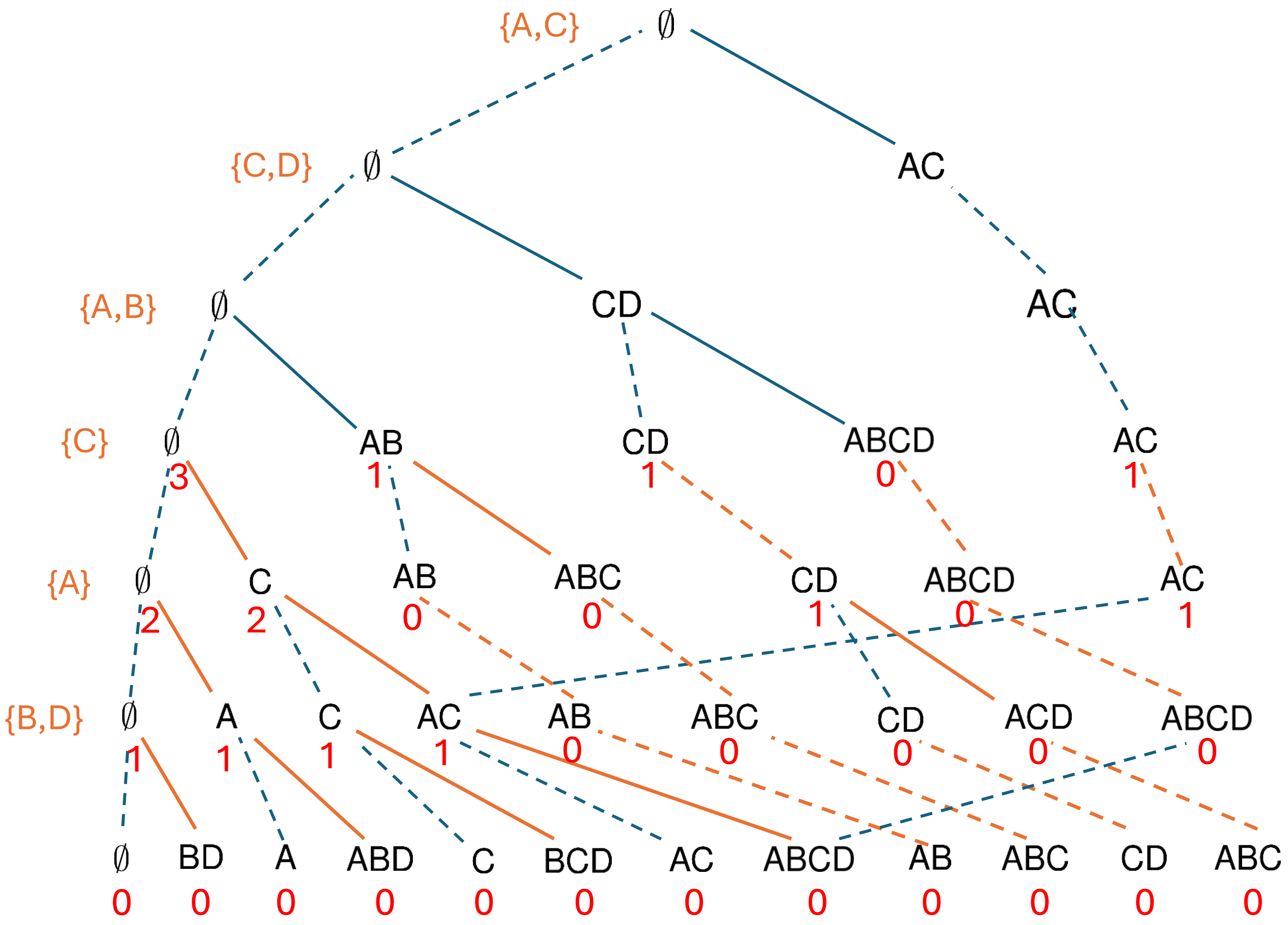


Carnegie Mellon University
Tepper School of Business

Serial DD
for a set packing problem instance



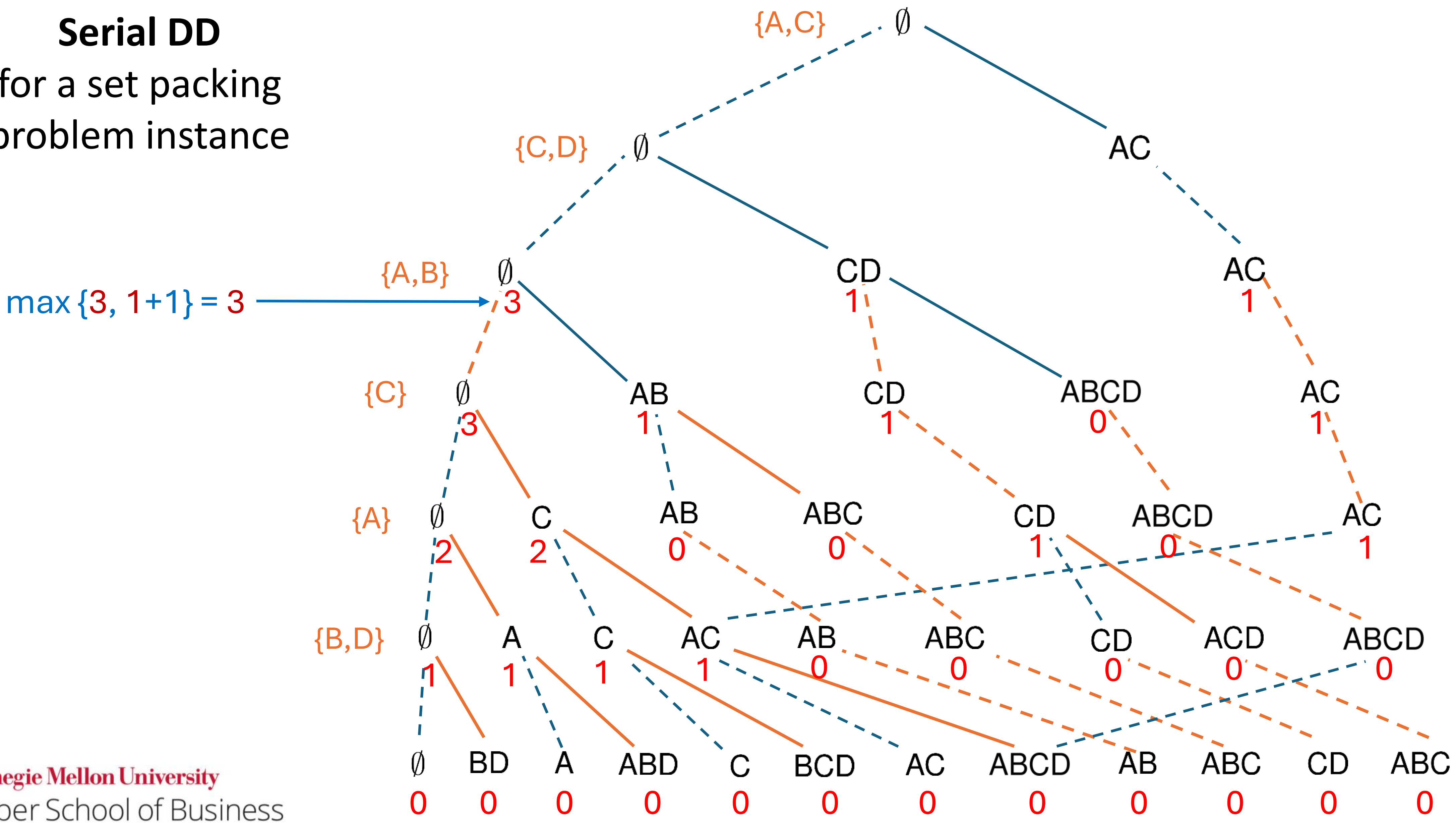
Serial DD
for a set packing
problem instance



Carnegie Mellon University
Tepper School of Business

Serial DD

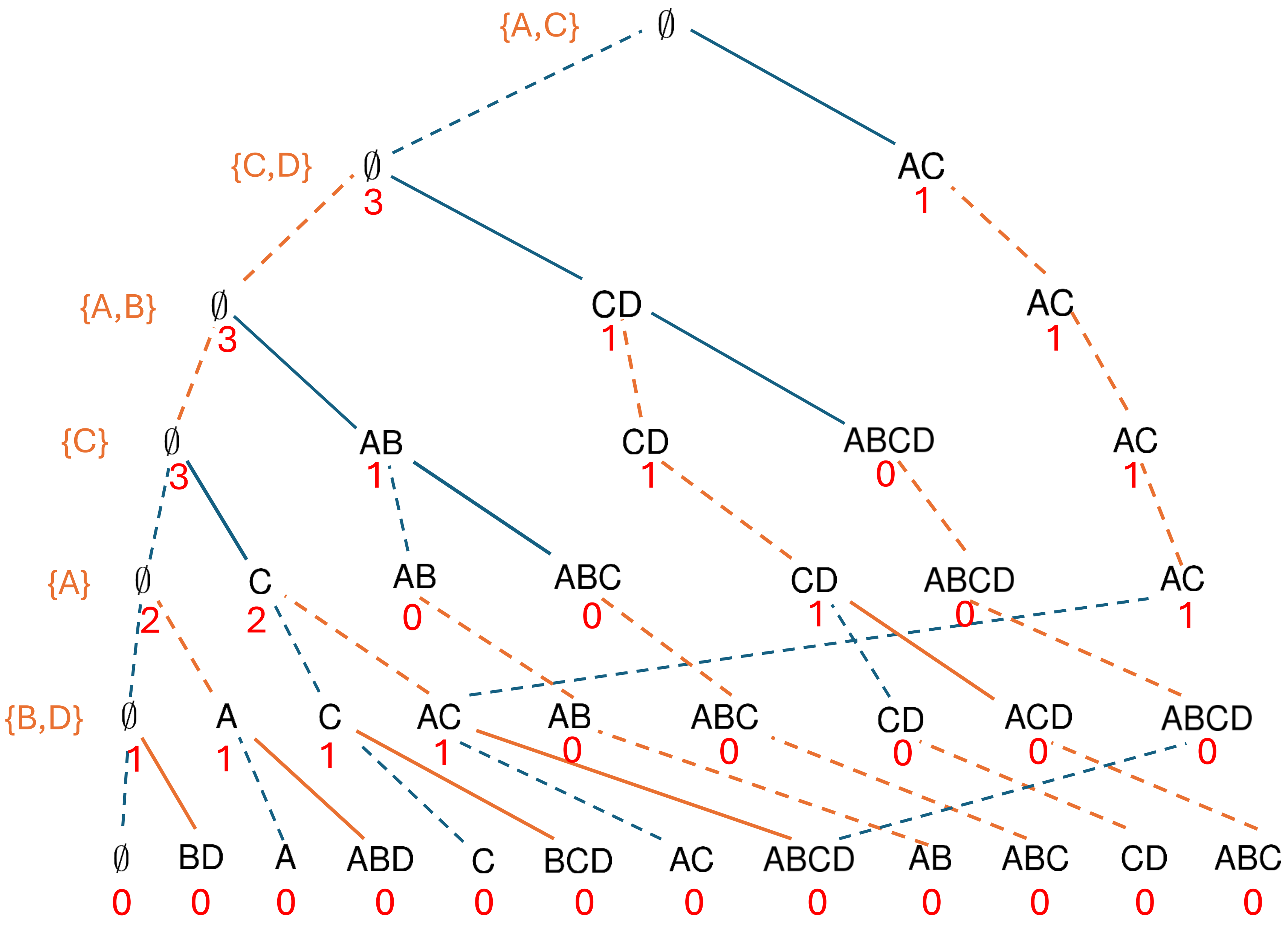
for a set packing problem instance



Carnegie Mellon University
Tepper School of Business

Serial DD

for a set packing problem instance

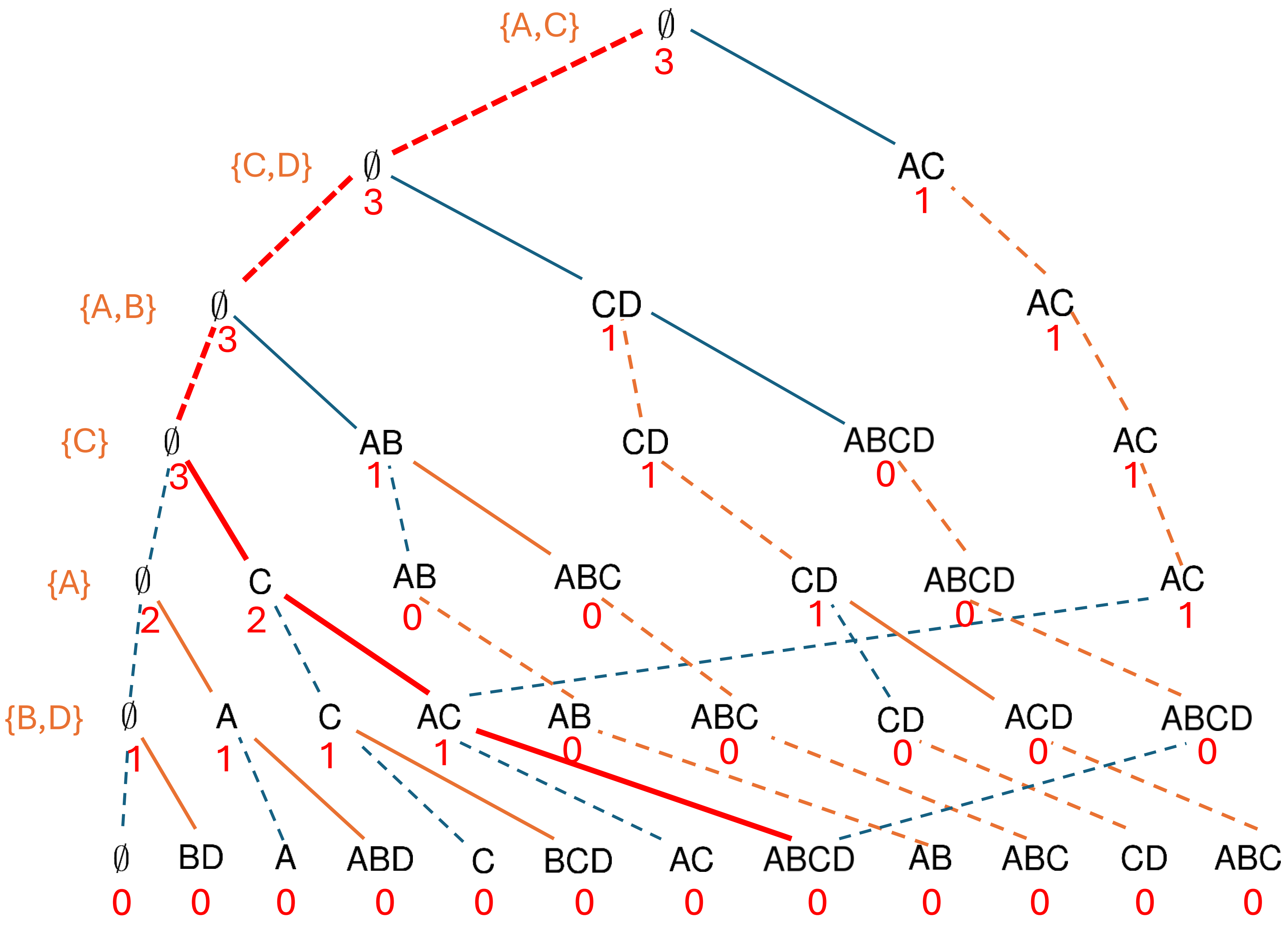


Carnegie Mellon University
Tepper School of Business

Serial DD

for a set packing problem instance

Trace optimal choices top-down to find optimal solution {C} + {A} + {B,D}

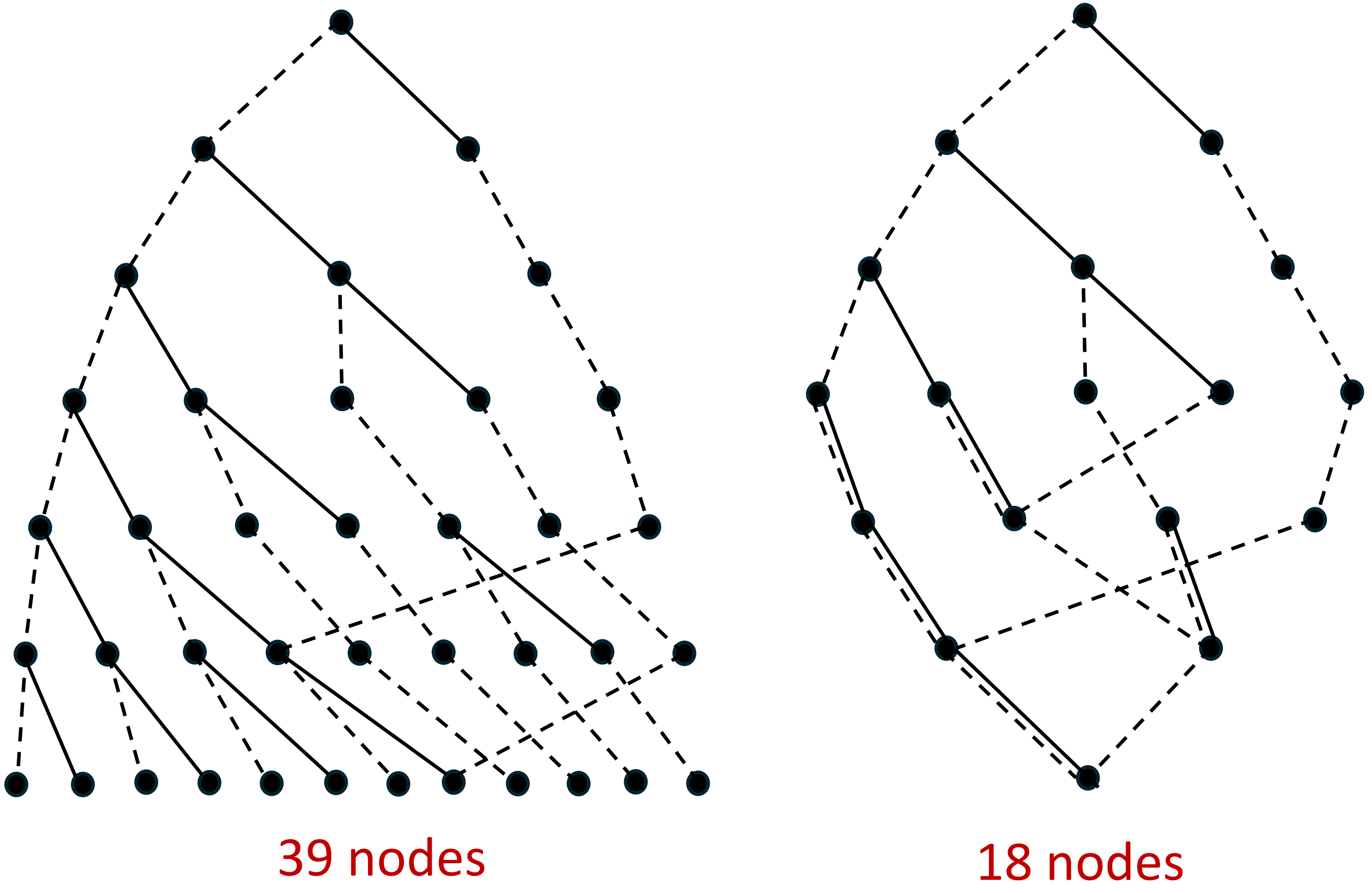


Original and **reduced** serial DDs

For a given variable ordering, a **unique reduced DD** represents a given set of feasible solutions.

Find an **optimal solution** in the reduced DD the same way as before.

There is no need for dynamic programming states.



How does a **DD** differ from a dynamic programming **state transition graph**?

- DD nodes need not be associated with **states**.
- The **reduced DD** can be much **smaller** than the state transition graph.
- Much smaller **relaxed DDs** (obtained by node splitting or merger during top-down compilation) provide **bounds** without solving an LP relaxation.
- Much smaller **restricted DDs** (obtained by deleting arcs during compilation) provide a **primal heuristic**.

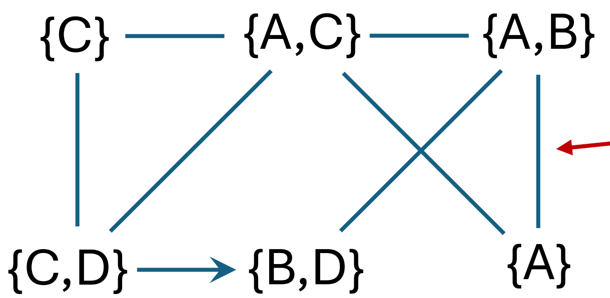
Nonserial Decision Diagrams

- They exploit structure of problem instances with **small treewidth**.
- **Treewidth** (with respect to an ordering) = **max in-degree** of nodes in the **induced** dependency graph.
- **Complexity** of a problem **instance** is at worst exponential in its minimum **treewidth** over all orderings.
- Instances with **small treewidth** generate **much smaller nonserial DDs** and are **much easier to solve**.

Example: Set packing

{A,C}
{C,D}
{A,B}
{C}
{A}
{B,D}

First, build **dependency graph** that shows variable coupling.
Here, 0-1 variables indicate whether each set is included in packing.

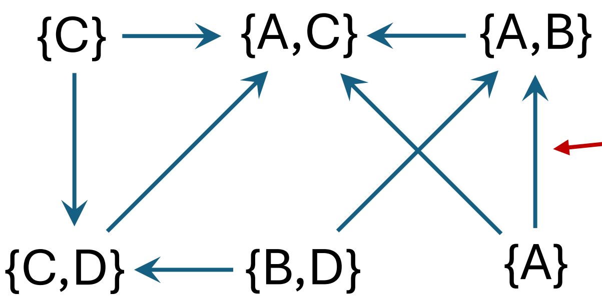


Arc indicates one or more elements in common

Example: Set packing

{A,C}
{C,D}
{A,B}
{C}
{A}
{B,D}

First, build **dependency graph** that shows variable coupling.
Here, 0-1 variables indicate whether each set is included in packing.



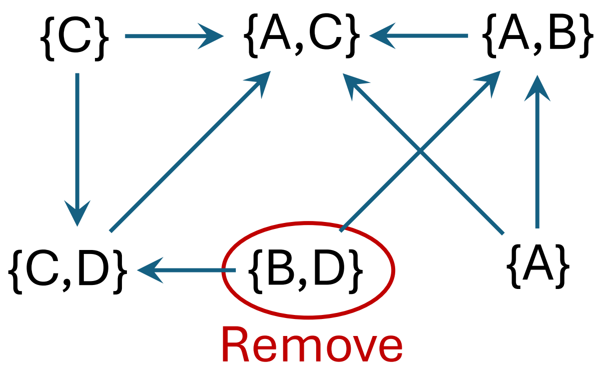
Arc indicates one or more elements in common

We generally don't know the min-treewidth ordering.
As a heuristic, we use a **min-degree ordering**.

Example: Set packing

{A,C}
{C,D}
{A,B}
{C}
{A}
{B,D}

Now, build **induced** dependency graph by removing nodes in order, adding arcs to connect all neighbors.



{B,D}

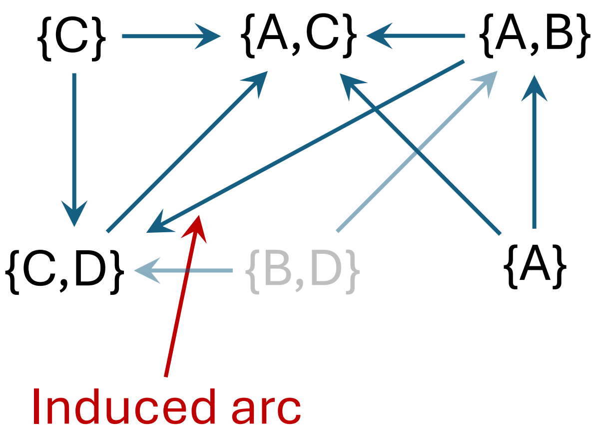
Build tree of layers for nonserial DD

Carnegie Mellon University
Tepper School of Business

Example: Set packing

Now, build **induced** dependency graph by removing nodes in order, adding arcs to connect all neighbors.

{A,C}
{C,D}
{A,B}
{C}
{A}
{B,D}



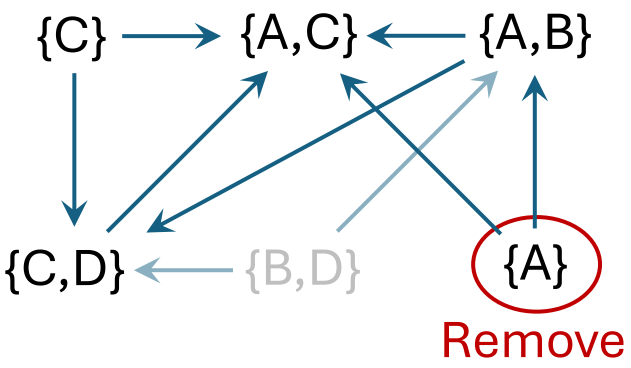
Build tree of layers for nonserial DD

{B,D}

Example: Set packing

{A,C}
{C,D}
{A,B}
{C}
{A}
{B,D}

Now, build **induced** dependency graph by removing nodes in order, adding arcs to connect all neighbors.



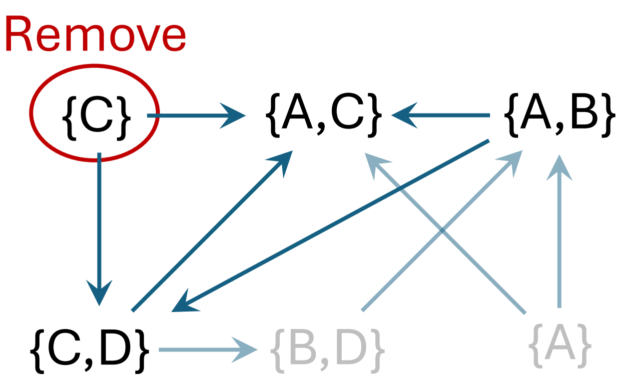
{B,D} {A}

Build tree of levels for nonserial DD

Example: Set packing

{A,C}
{C,D}
{A,B}
{C}
{A}
{B,D}

Now, build **induced** dependency graph by removing nodes in order, adding arcs to connect all neighbors.



{C}

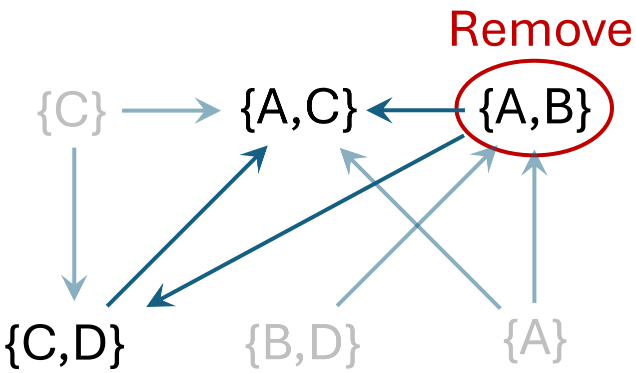
{B,D} {A}

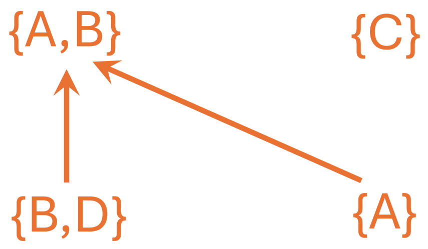
Build tree of levels for nonserial DD

Example: Set packing

{A,C}
{C,D}
{A,B}
{C}
{A}
{B,D}

Now, build **induced** dependency graph by removing nodes in order, adding arcs to connect all neighbors.



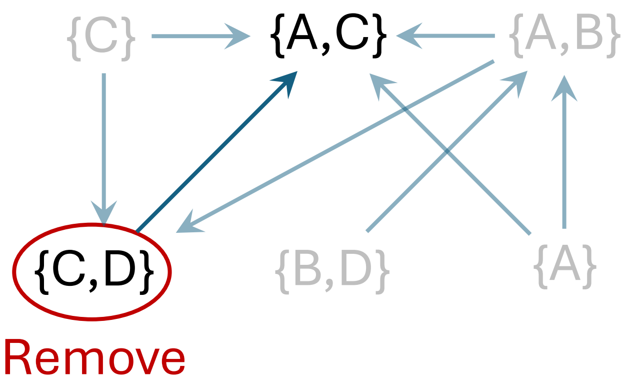


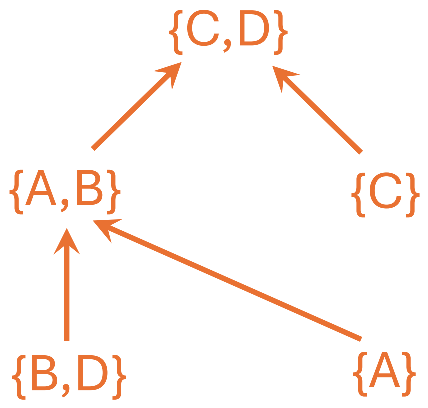
Build tree of levels
for nonserial DD

Example: Set packing

{A,C}
{C,D}
{A,B}
{C}
{A}
{B,D}

Now, build **induced** dependency graph by removing nodes in order, adding arcs to connect all neighbors.



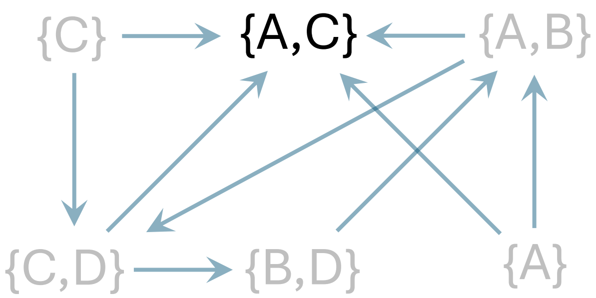


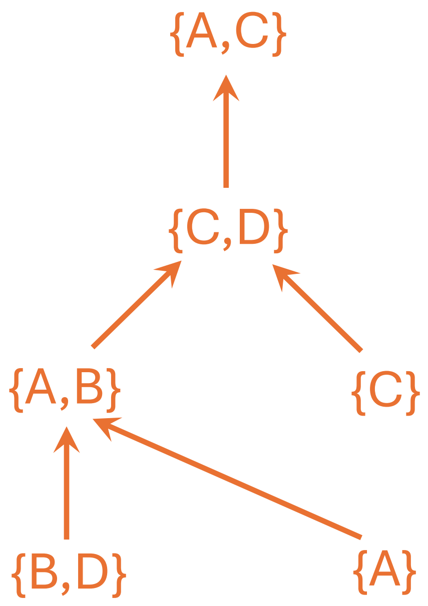
Build tree of levels
for nonserial DD

Example: Set packing

{A,C}
{C,D}
{A,B}
{C}
{A}
{B,D}

Now, build **induced** dependency graph by removing nodes in order, adding arcs to connect all neighbors.

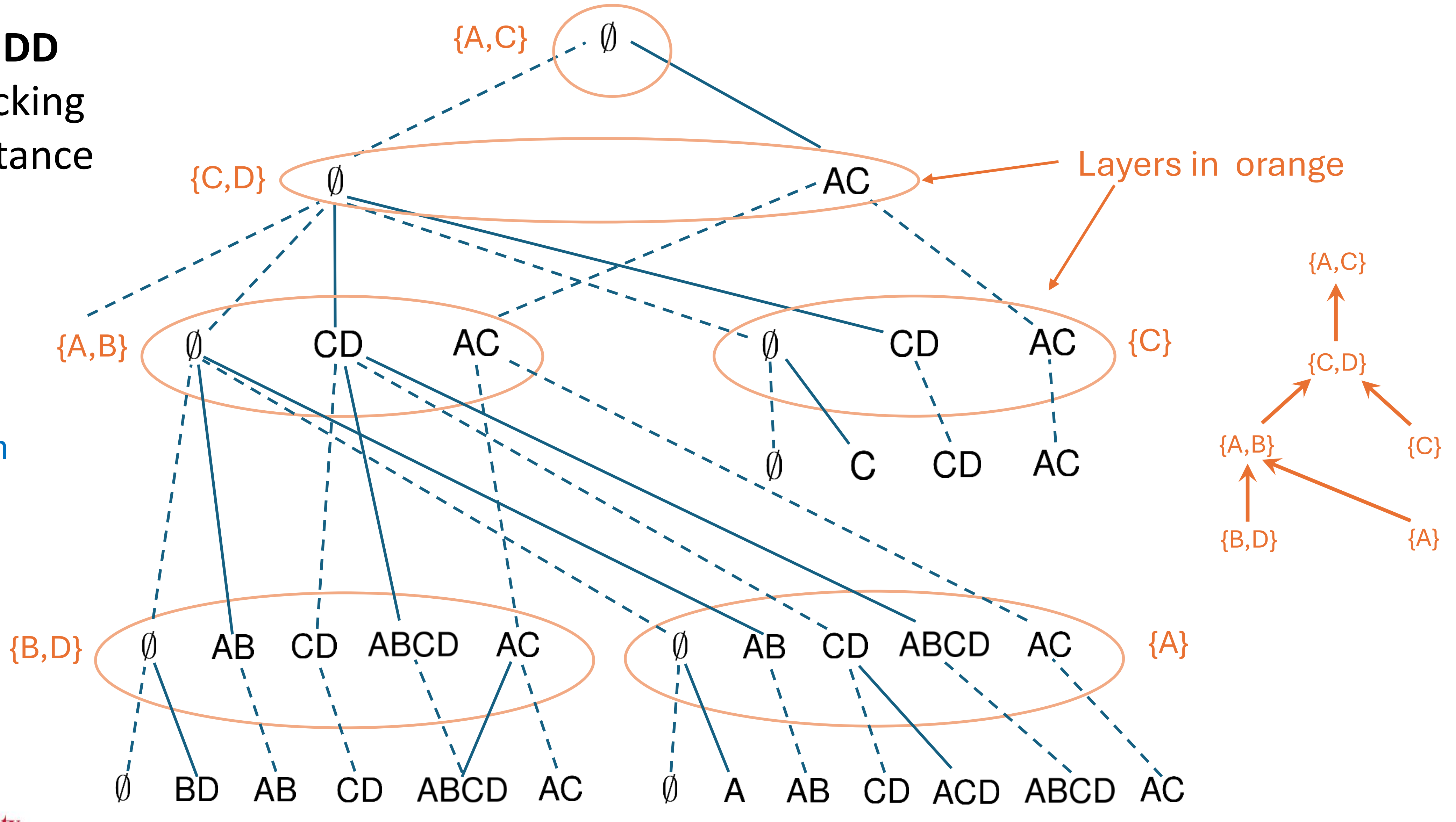




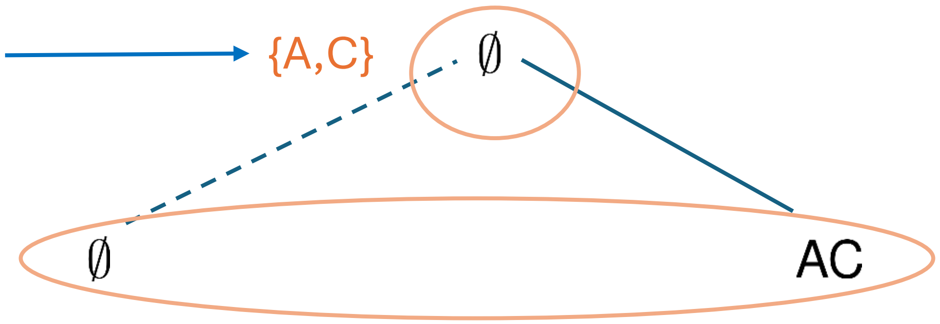
Treewidth =
max in-degree = 2

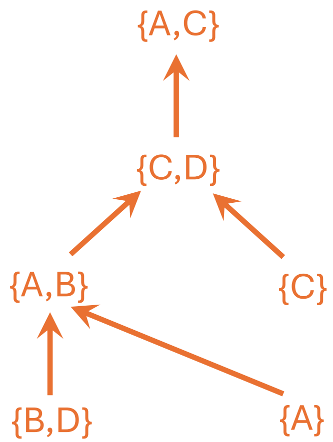
Nonserial DD
for a set packing problem instance

Layers form a **tree** rather than an ordered sequence

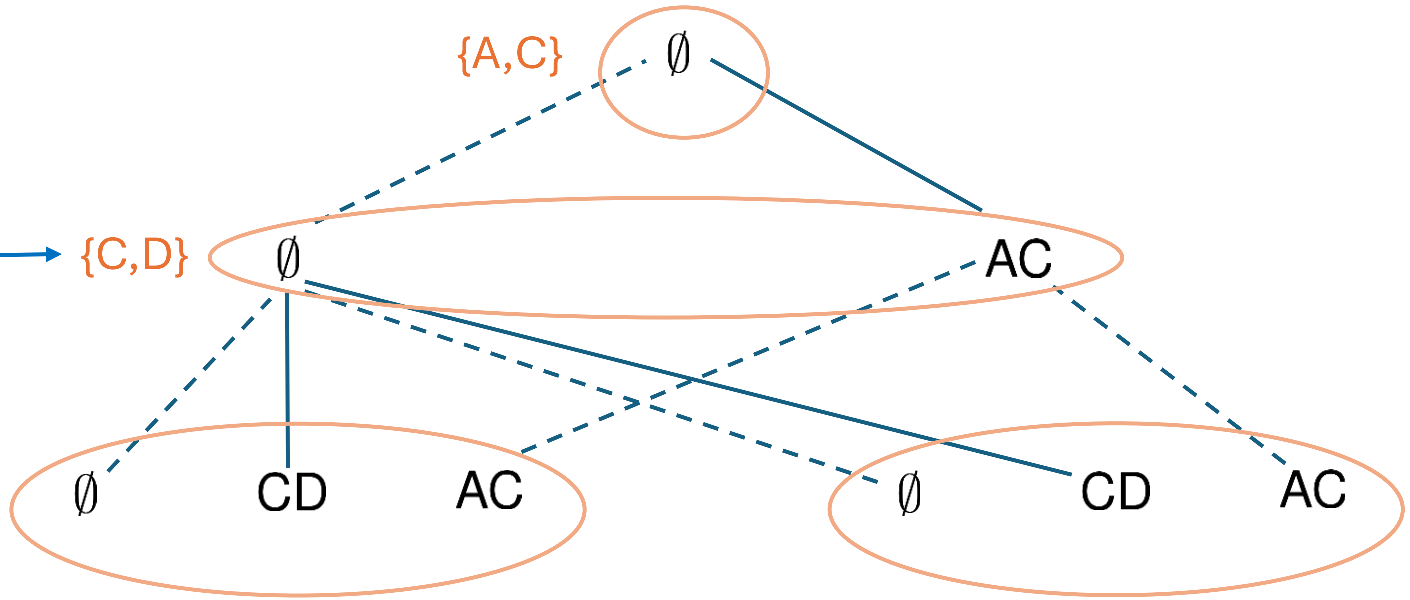


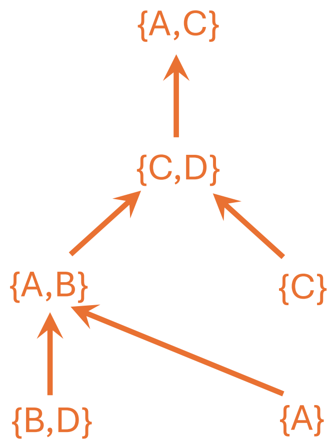
Decide whether
to select set {A,C}

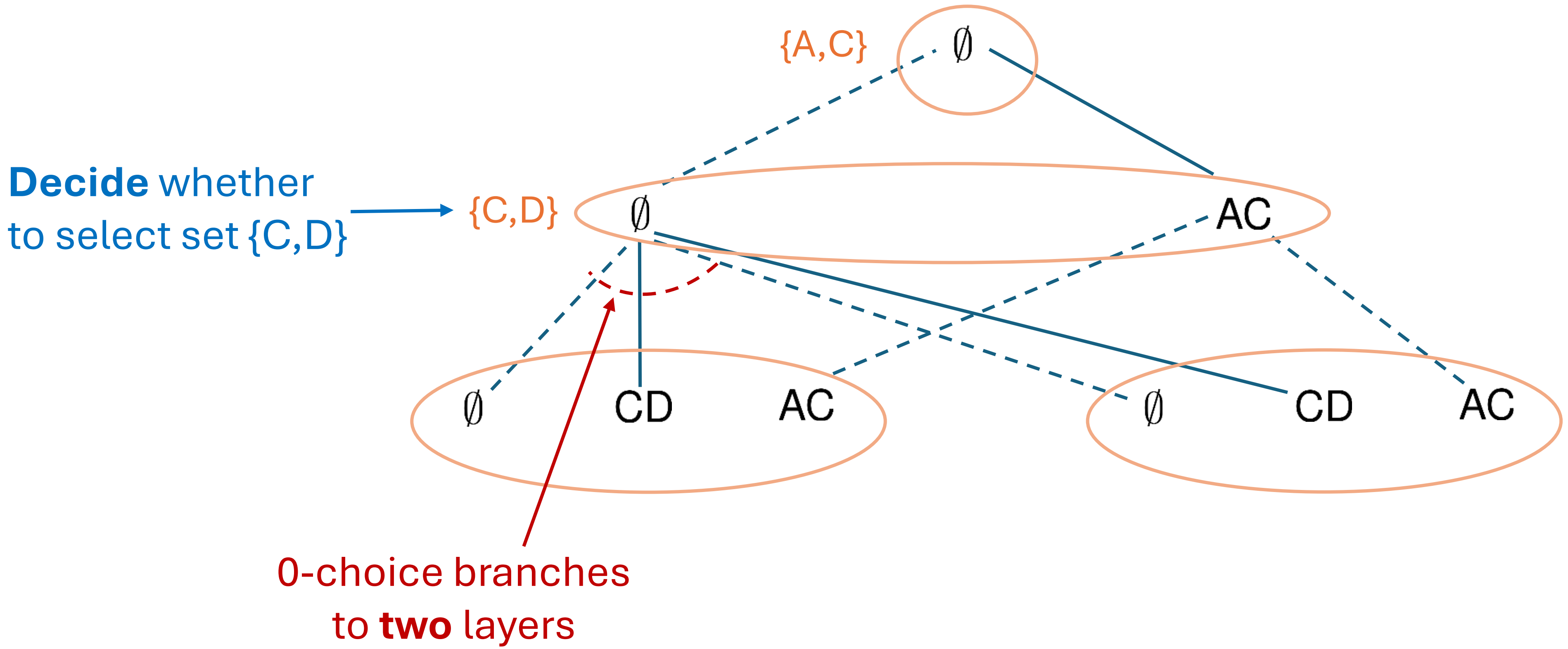




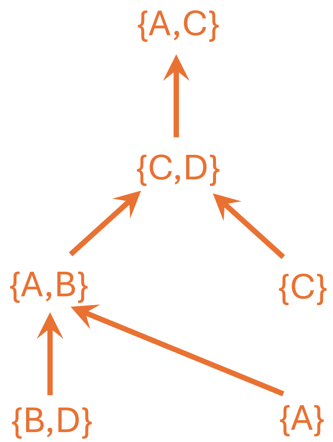
Decide whether to select set {C,D}



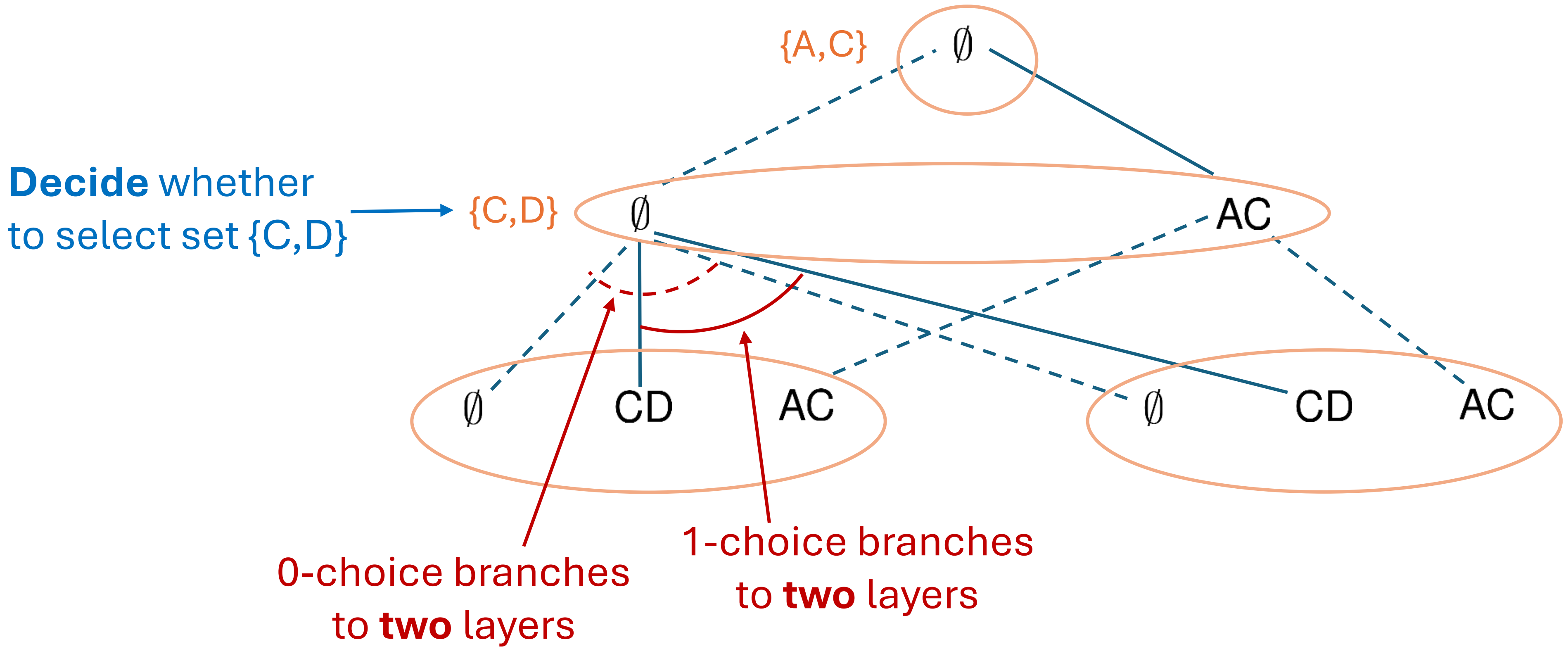




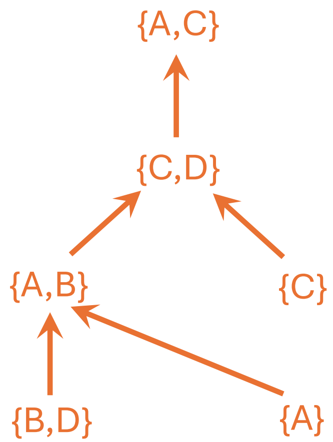
{A,C}
∅
Decide whether
to select set {C,D}
{C,D}
∅
AC
∅
CD
AC
∅
CD
AC
0-choice branches
to two layers



{A,C}
{C,D}
{A,B}
{C}
{B,D}
{A}

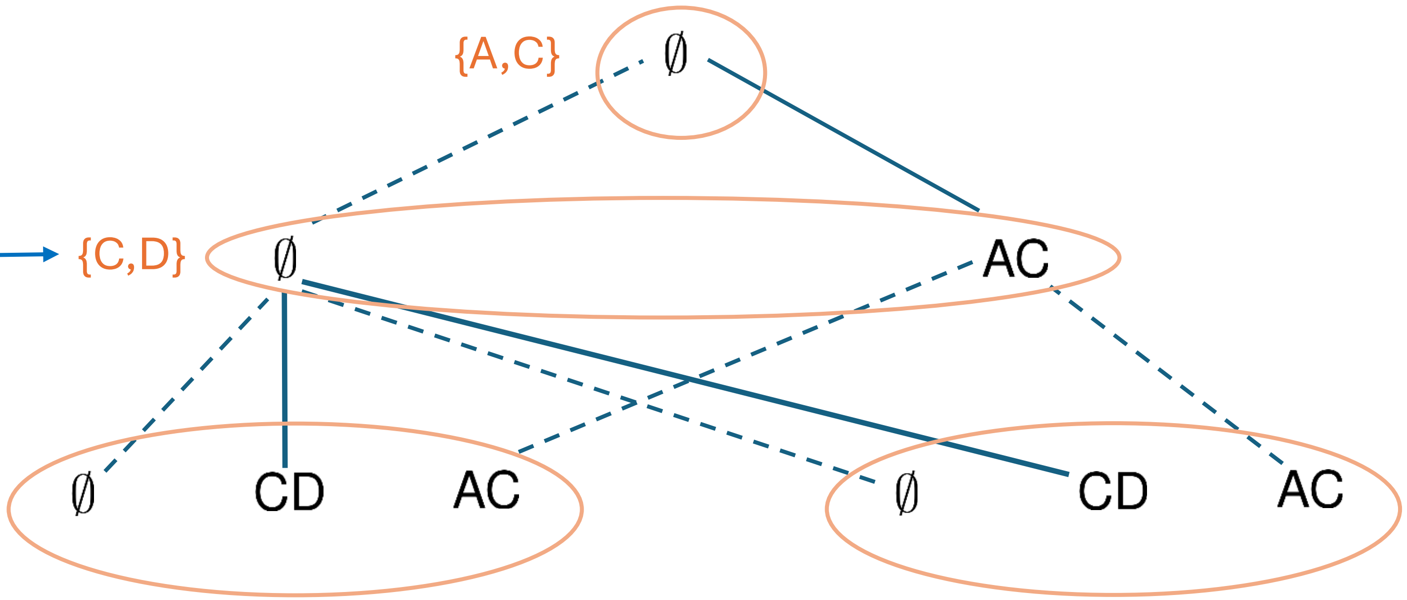


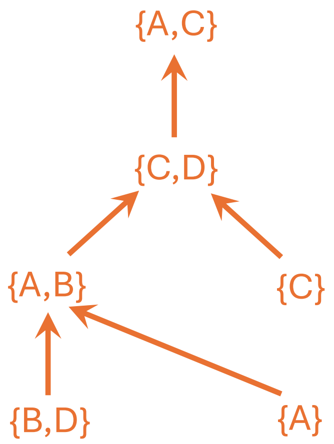
Decide whether
to select set {C,D}
{A,C}
∅
{C,D}
∅
AC
∅
CD
AC
∅
CD
AC
0-choice branches
to **two** layers
1-choice branches
to **two** layers



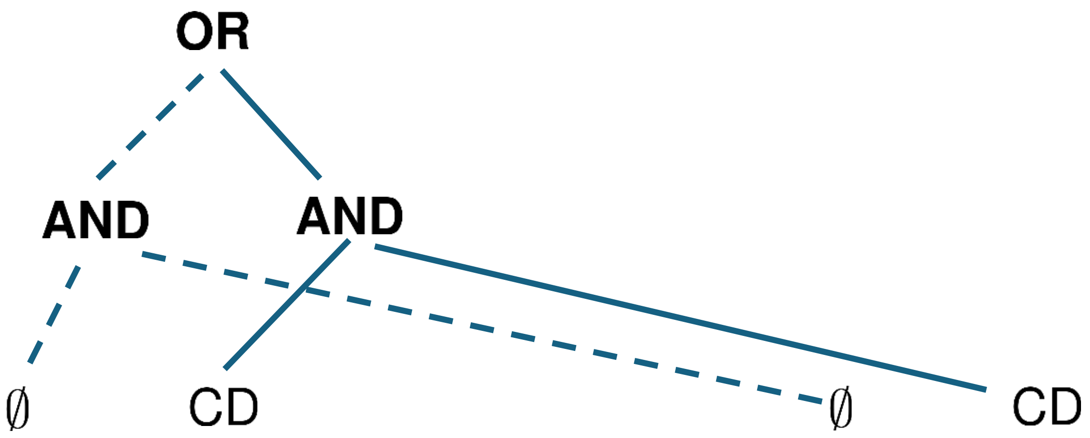
{A,C}
{C,D}
{A,B}
{C}
{B,D}
{A}

Decide whether to select set {C,D}

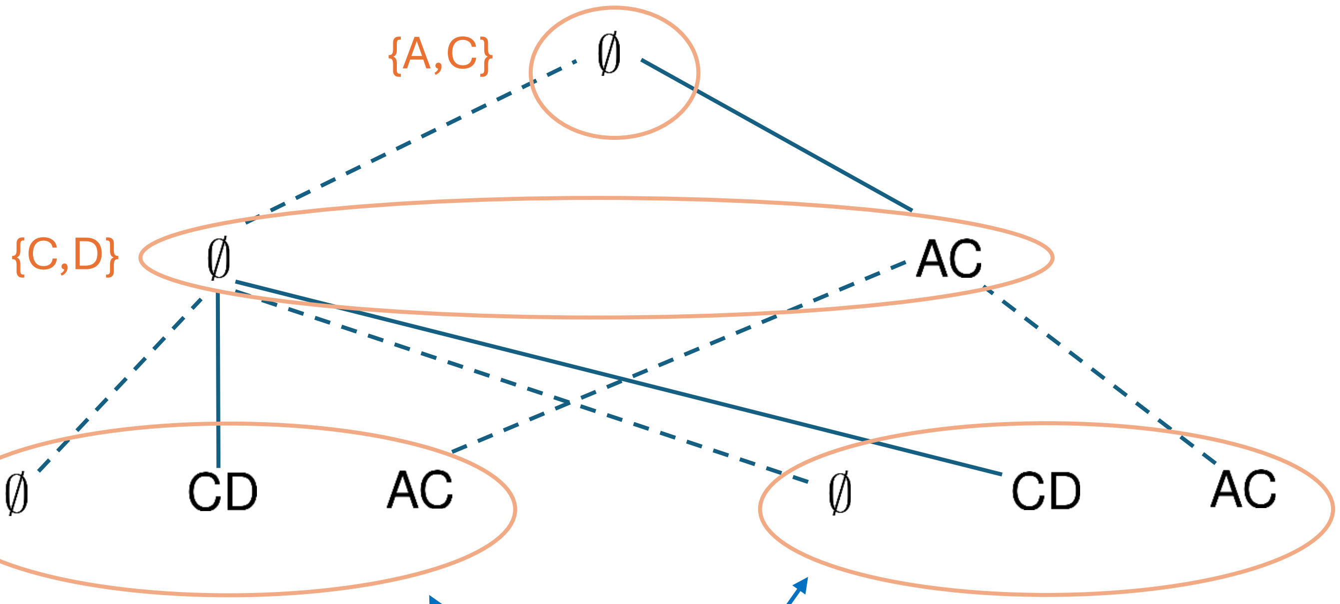


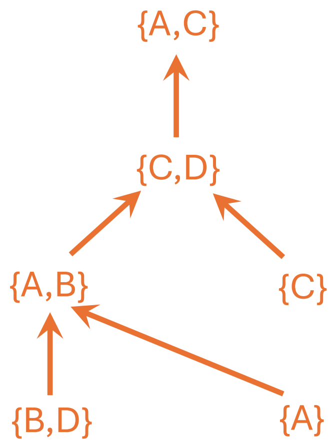


Can be viewed as **and-or** DD



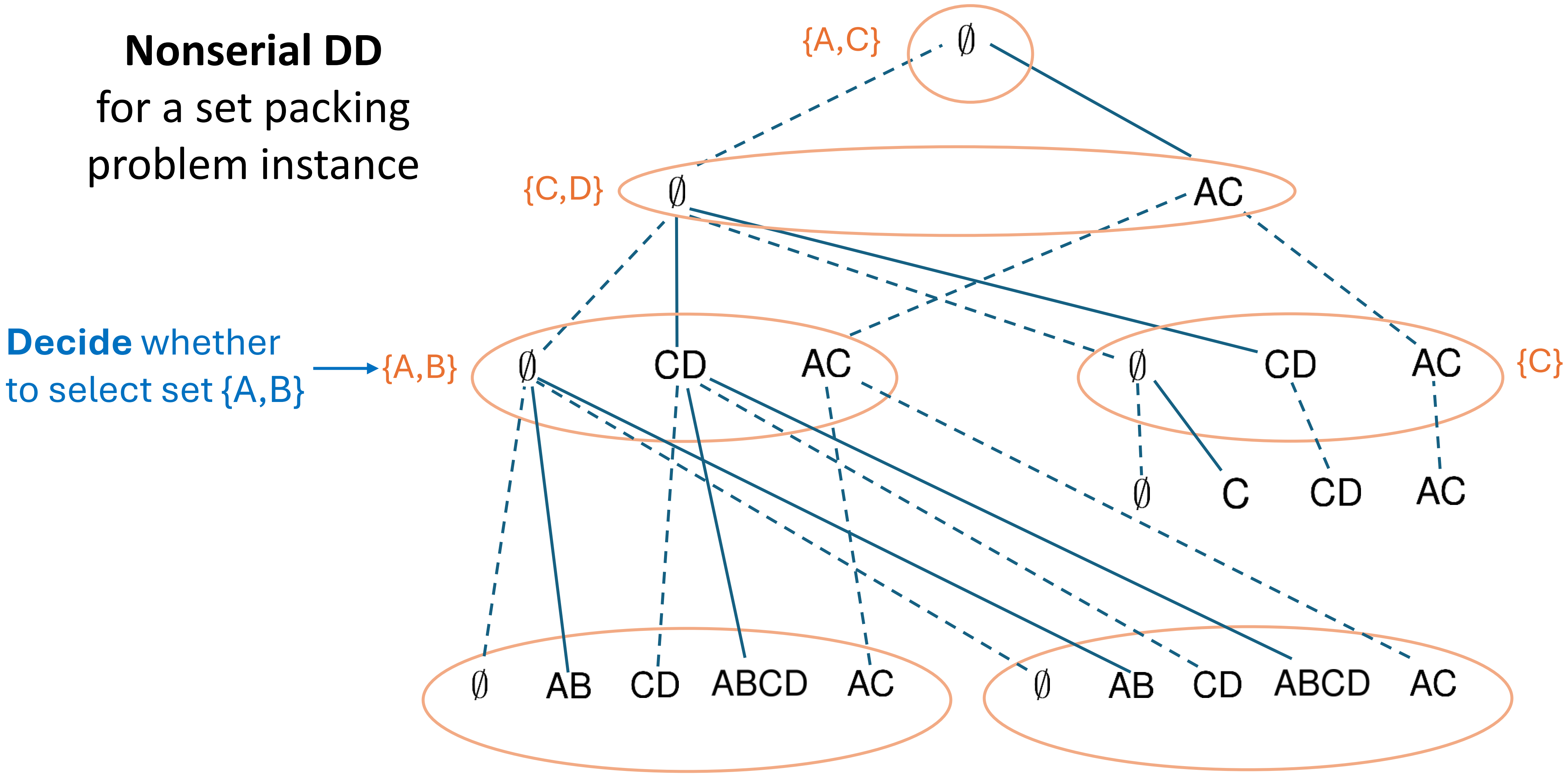
Decide whether to select set {C,D}

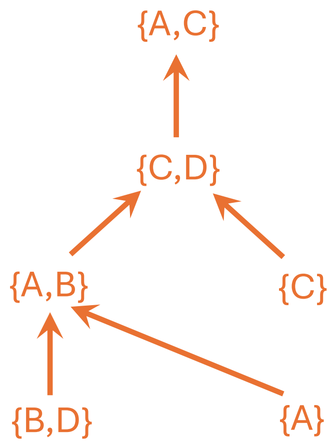




Duplication of states creates some overhead, but this will be offset by smaller width of layers.

Nonserial DD
for a set packing
problem instance

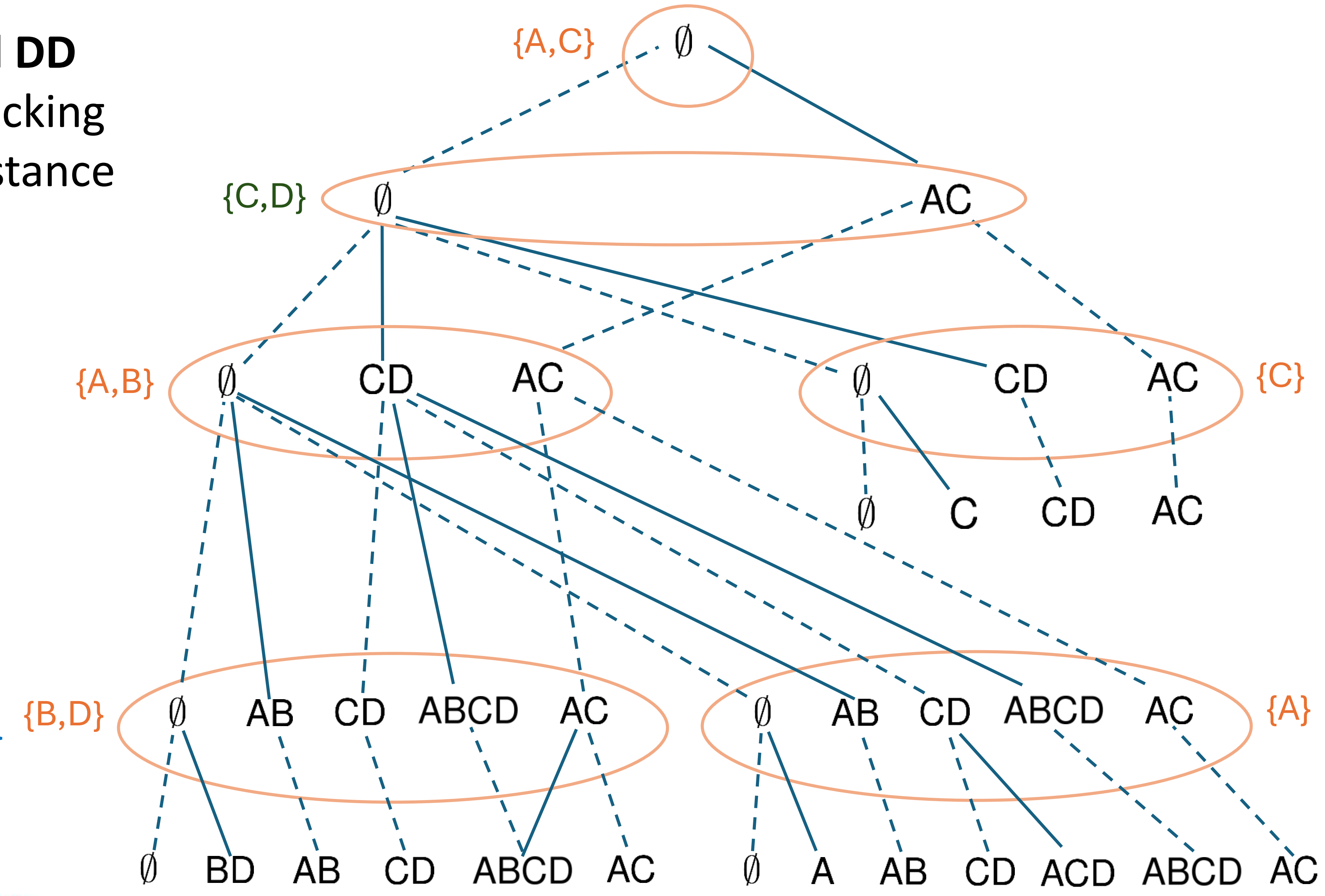


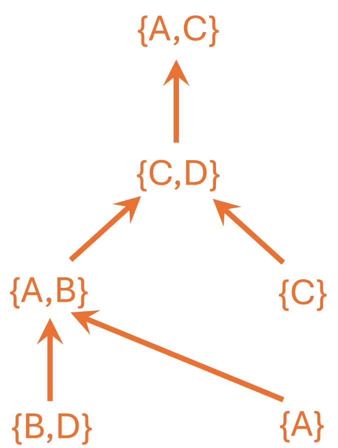


Decide whether
to select set {C}

Nonserial DD

for a set packing problem instance





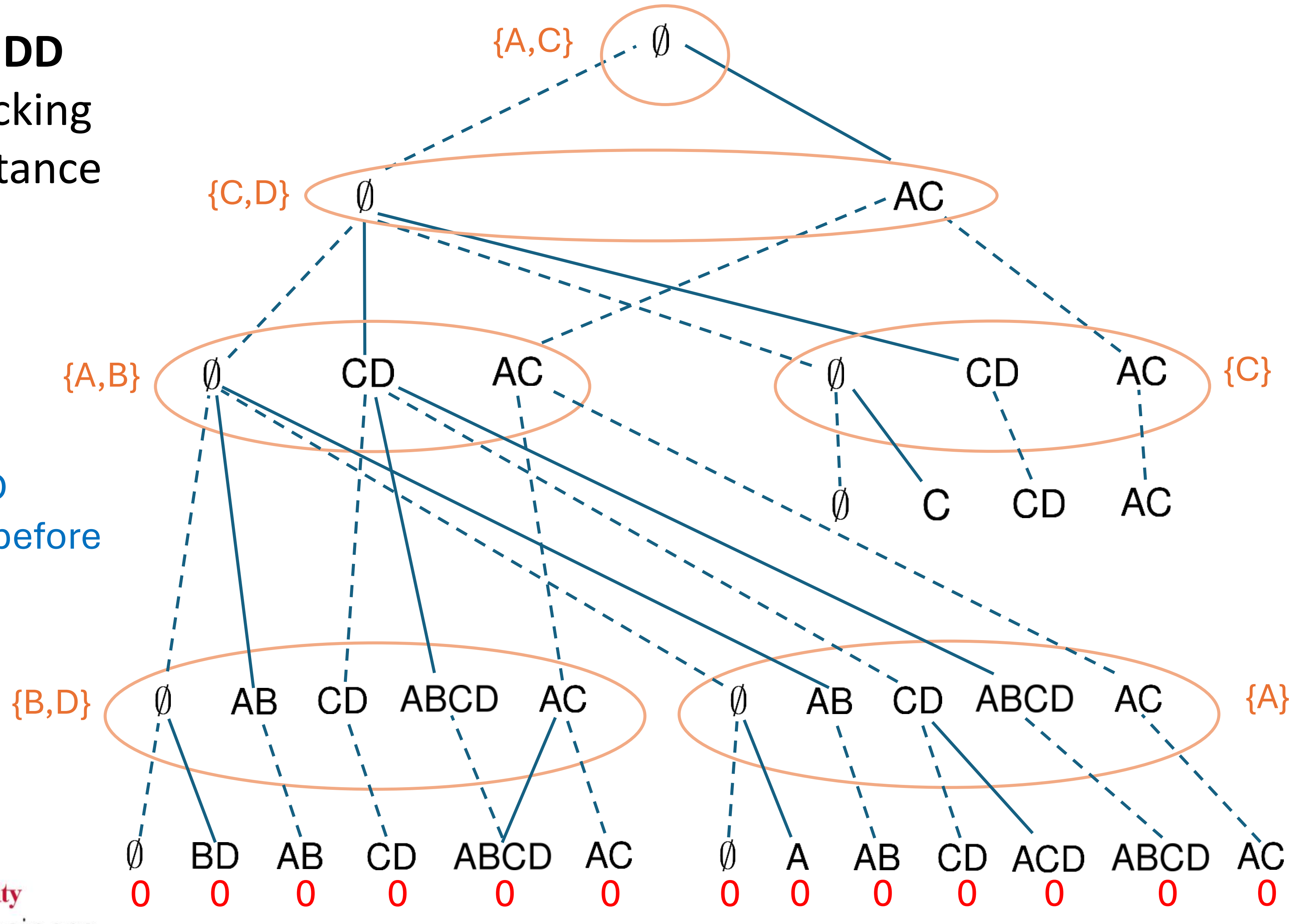
Decide whether to select set {B,D}

Decide whether to select set {A}

DD has 36 nodes

Carnegie Mellon University
Tepper School of Business

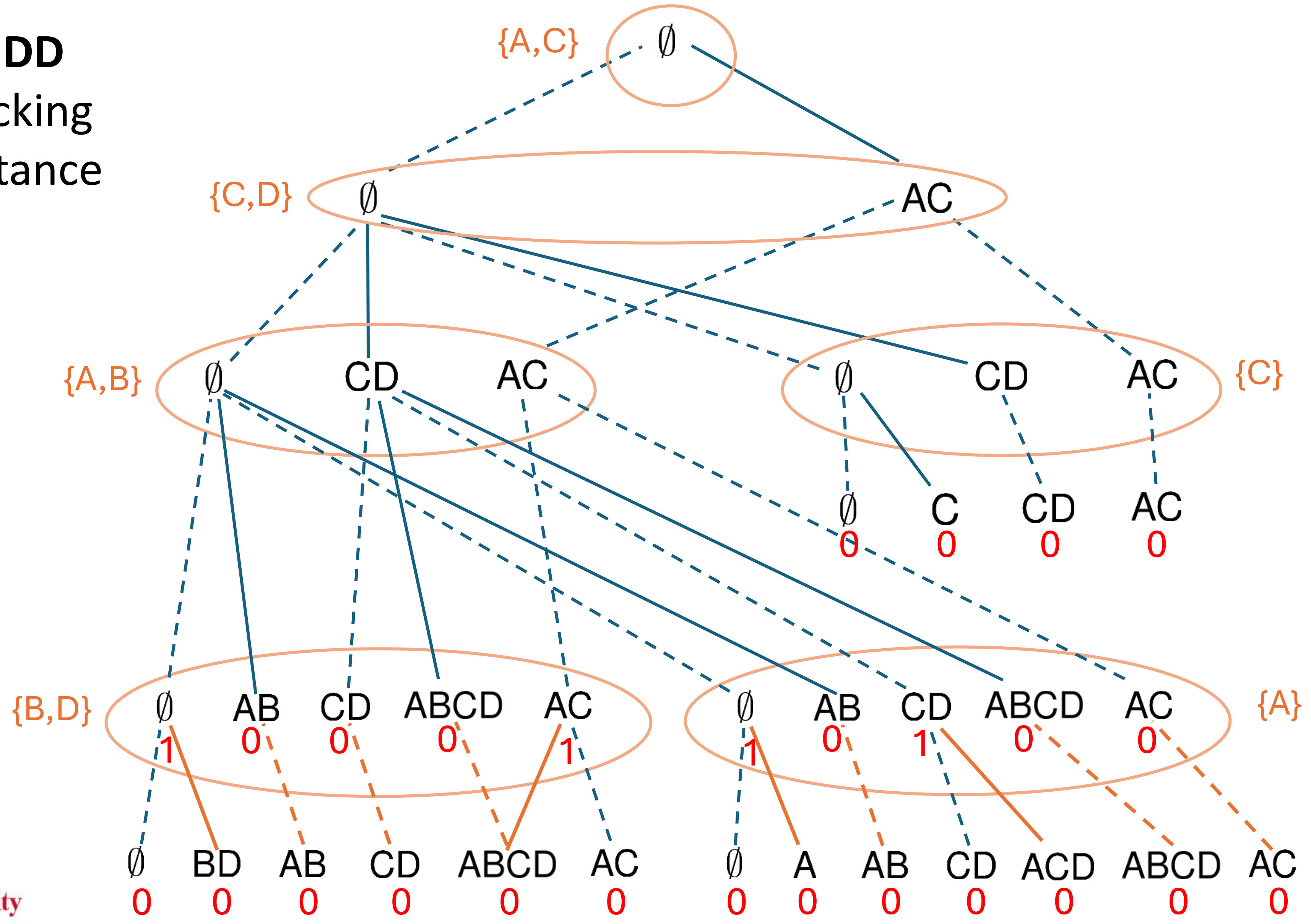
Nonserial DD
for a set packing
problem instance



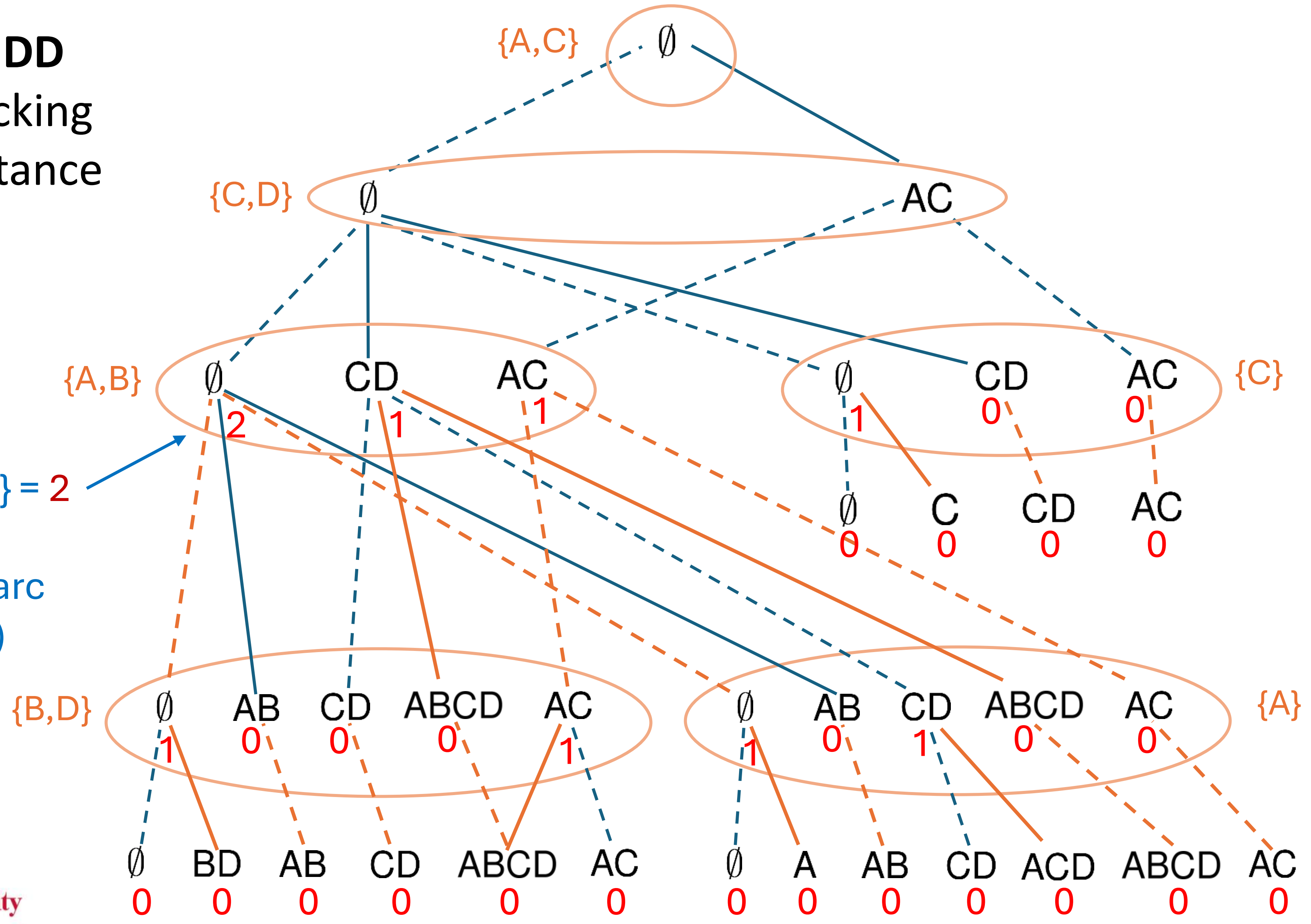
Evaluate the DD
bottom-up as before

Nonserial DD

for a set packing problem instance



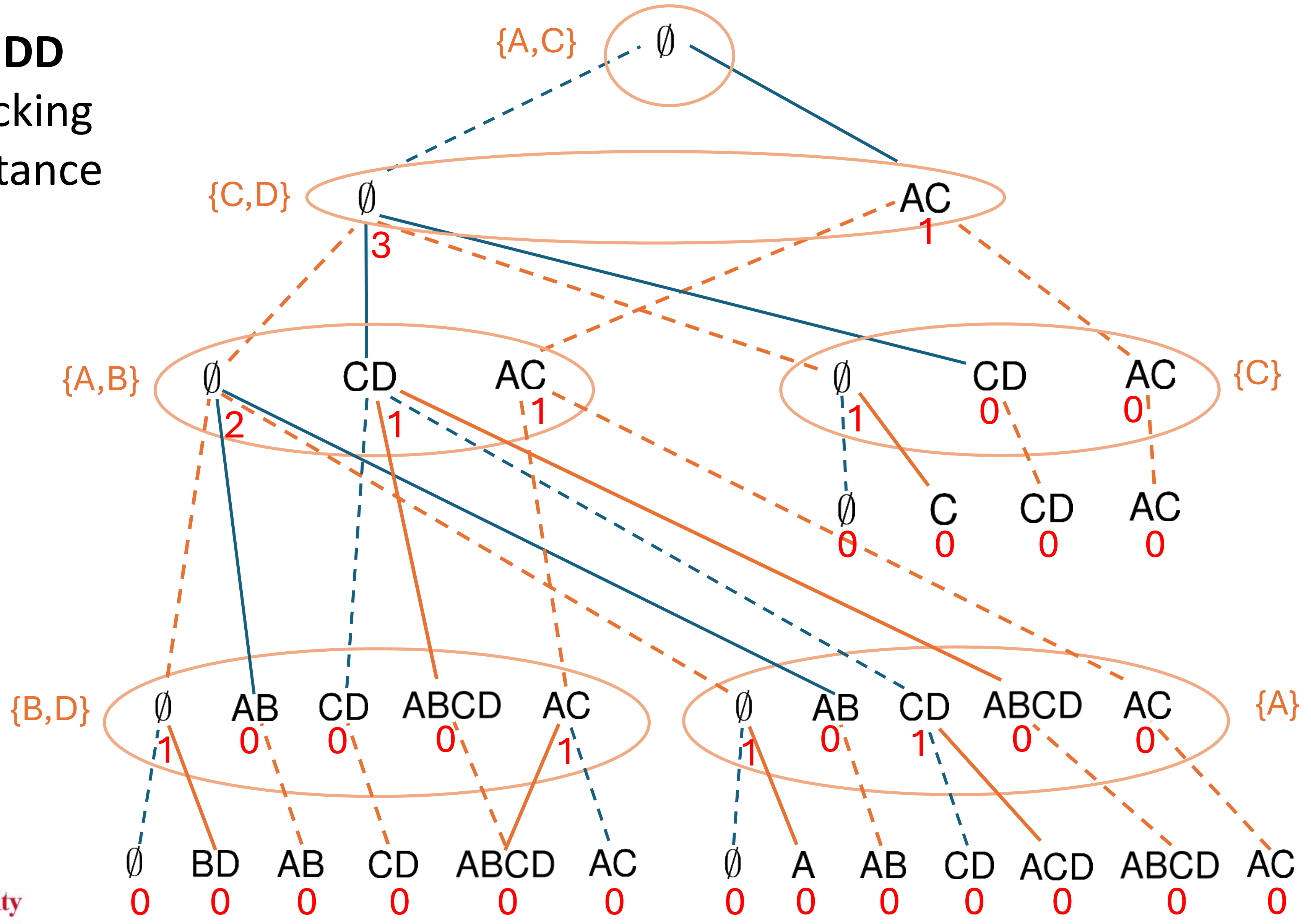
Nonserial DD
for a set packing problem instance



max {1+1, 0+0+1} = 2
Outgoing 1-arcs counted as one arc (as in and-or DD)

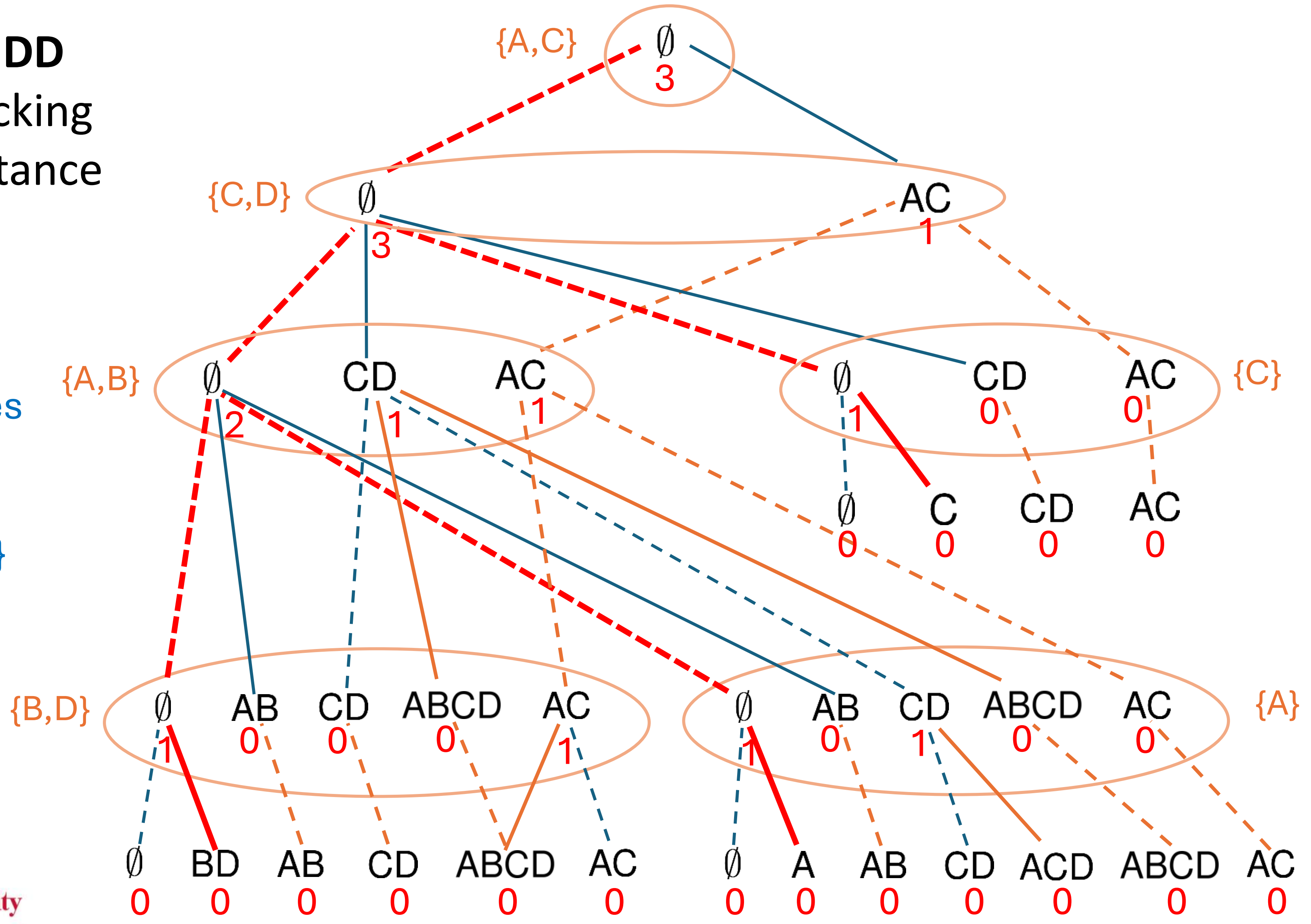
Nonserial DD

for a set packing problem instance



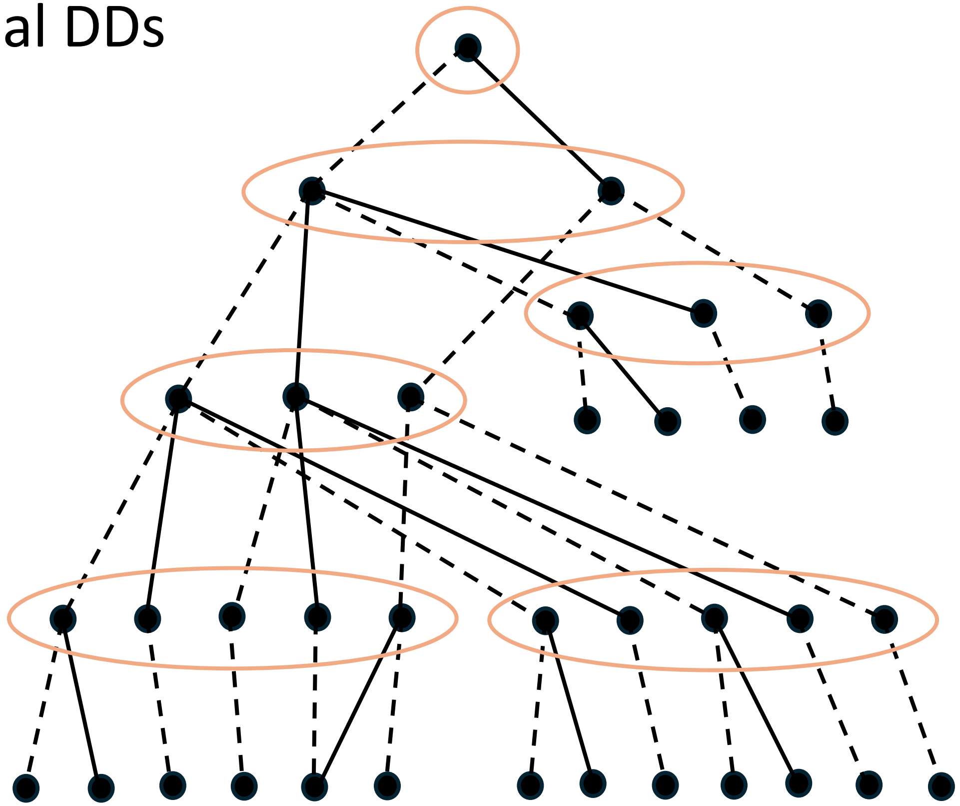
Carnegie Mellon University
Tepper School of Business

Nonserial DD
for a set packing
problem instance

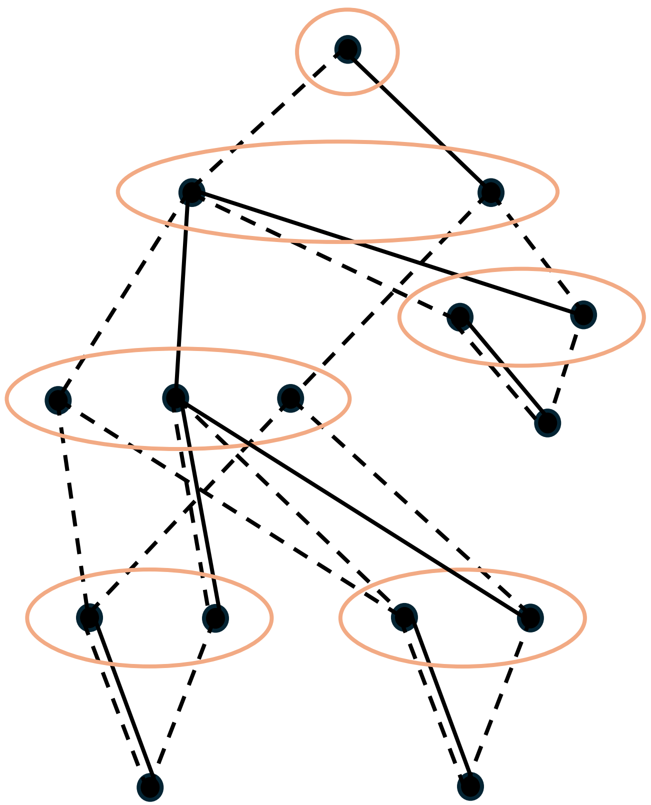


Trace **tree** of
optimal choices
to find optimal
solution
{C} + {A} + {B,D}

Original and **reduced** nonserial DDs



36 nodes



15 nodes

Computational Experiments

- **Compare size** of non-reduced **serial** and **nonserial DDs** for randomly generated **set packing** instances of various treewidths.
- Use **min-degree ordering** for serial and nonserial DDs, as it benefits both.
- Let each element occur in a given set with **probability *p***.
- **Discard** random instances with a **disconnected** dependency graph.
- Use smaller **values of *p*** to get smaller **treewidths**.

Serial and **nonserial**
DD size vs treewidth

Each instance is represented by **two data points**.

Instances with **many elements per set** are **easier to solve** due to fewer feasible solutions.

Average 2.4-6 elements/set

Serial and **nonserial**
DD size vs treewidth

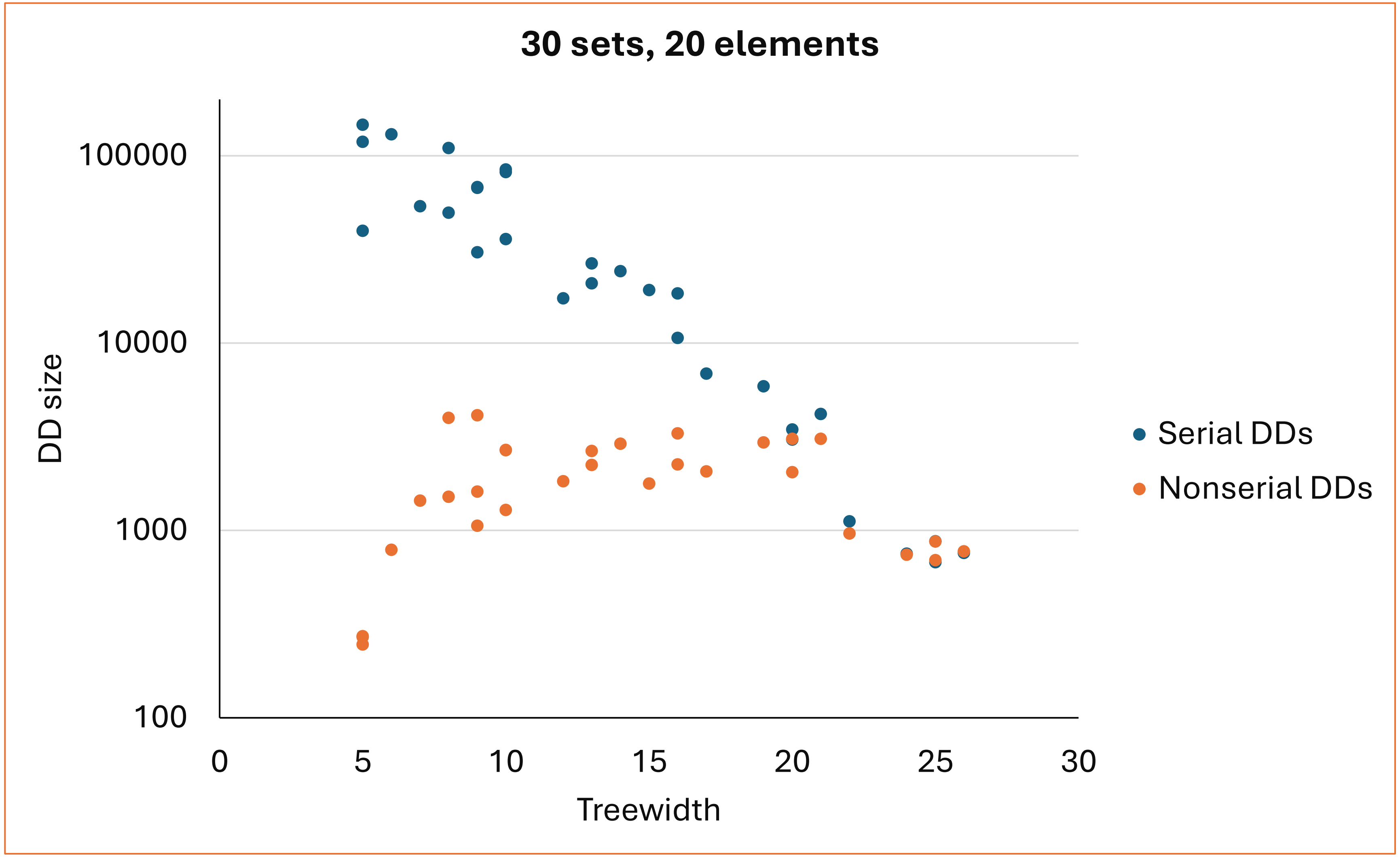
Smaller bandwidths result in **much larger serial DDs** (instances are harder).

Nonserial DD size is fairly **constant.**

Nonserial DD's exploitation of small bandwidth **offsets** greater difficulty of the instance.

Average 2.4-6 elements/set

Serial and **nonserial**
DD size vs treewidth



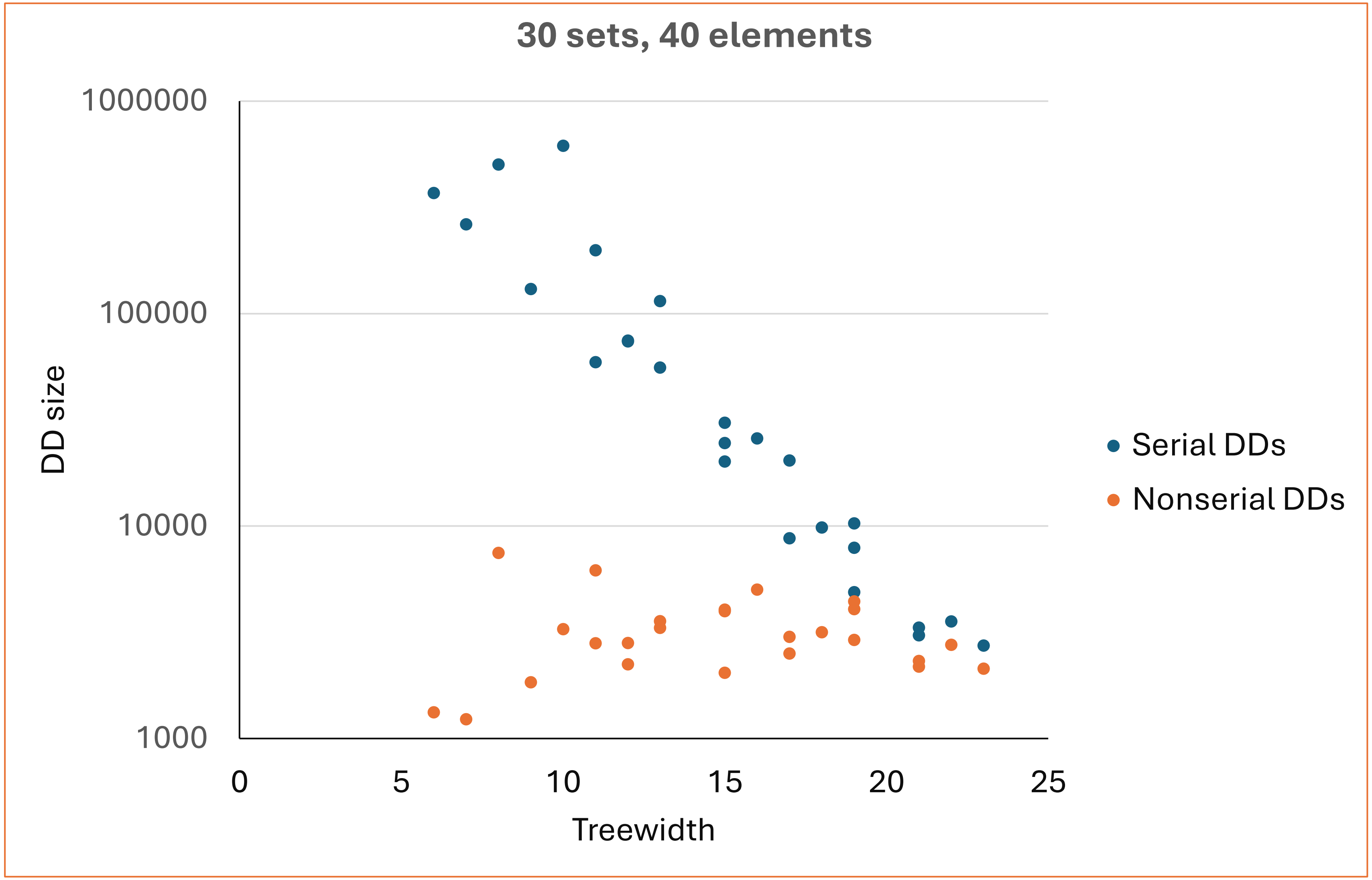
Similar pattern,
except for inverted-U shape of nonserial data points

Average 1.6-6 elements/set

Serial and **nonserial**

DD size vs treewidth

Larger DDs, but otherwise **similar pattern**



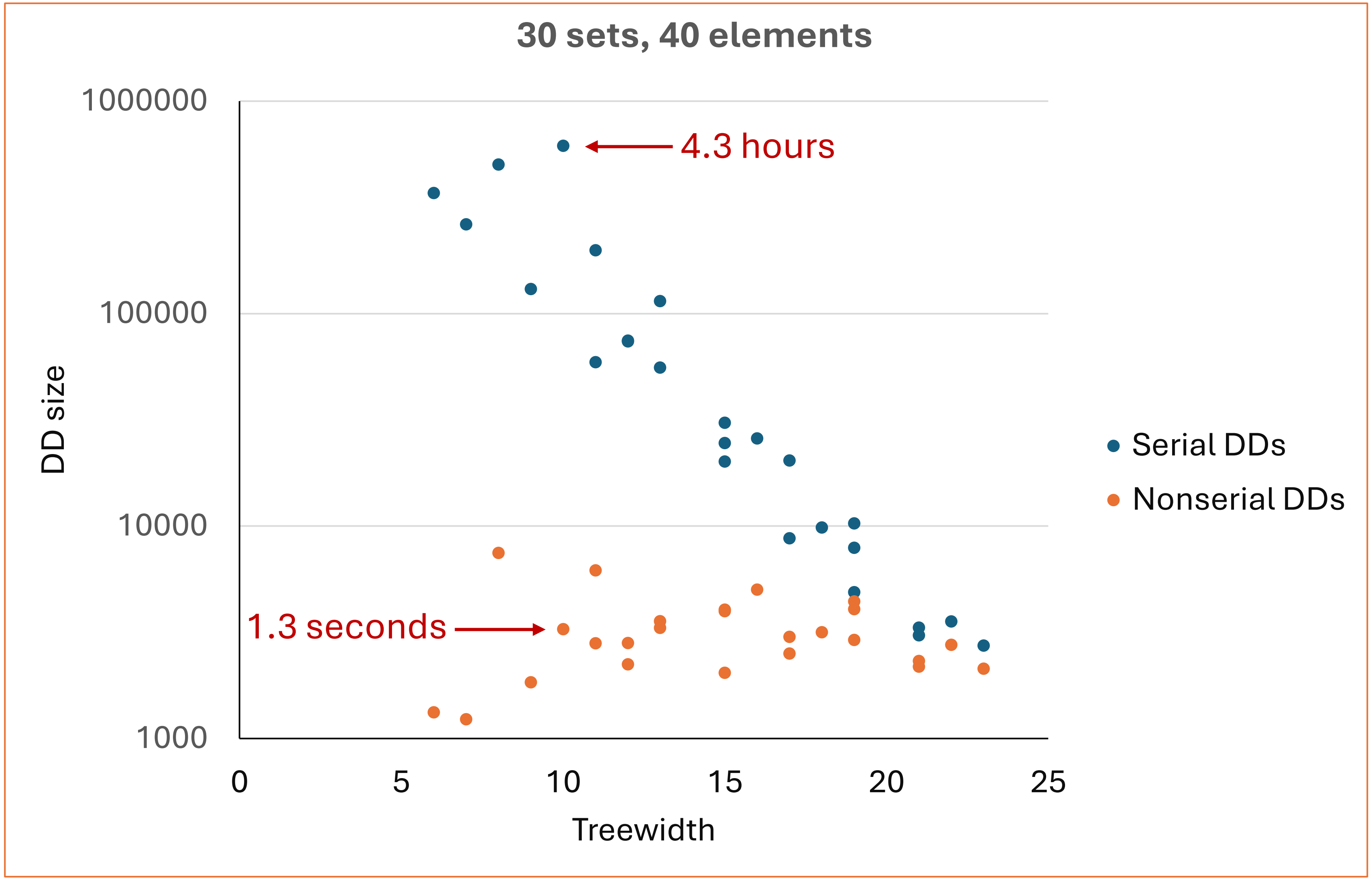
Average 2.4-6 elements/set

Serial and **nonserial**
DD size vs treewidth

Difference in **compile time** is even more dramatic than DD size.

Compile time is roughly **quadratic** in max **layer** size.

Serial DD layers are much **larger**.

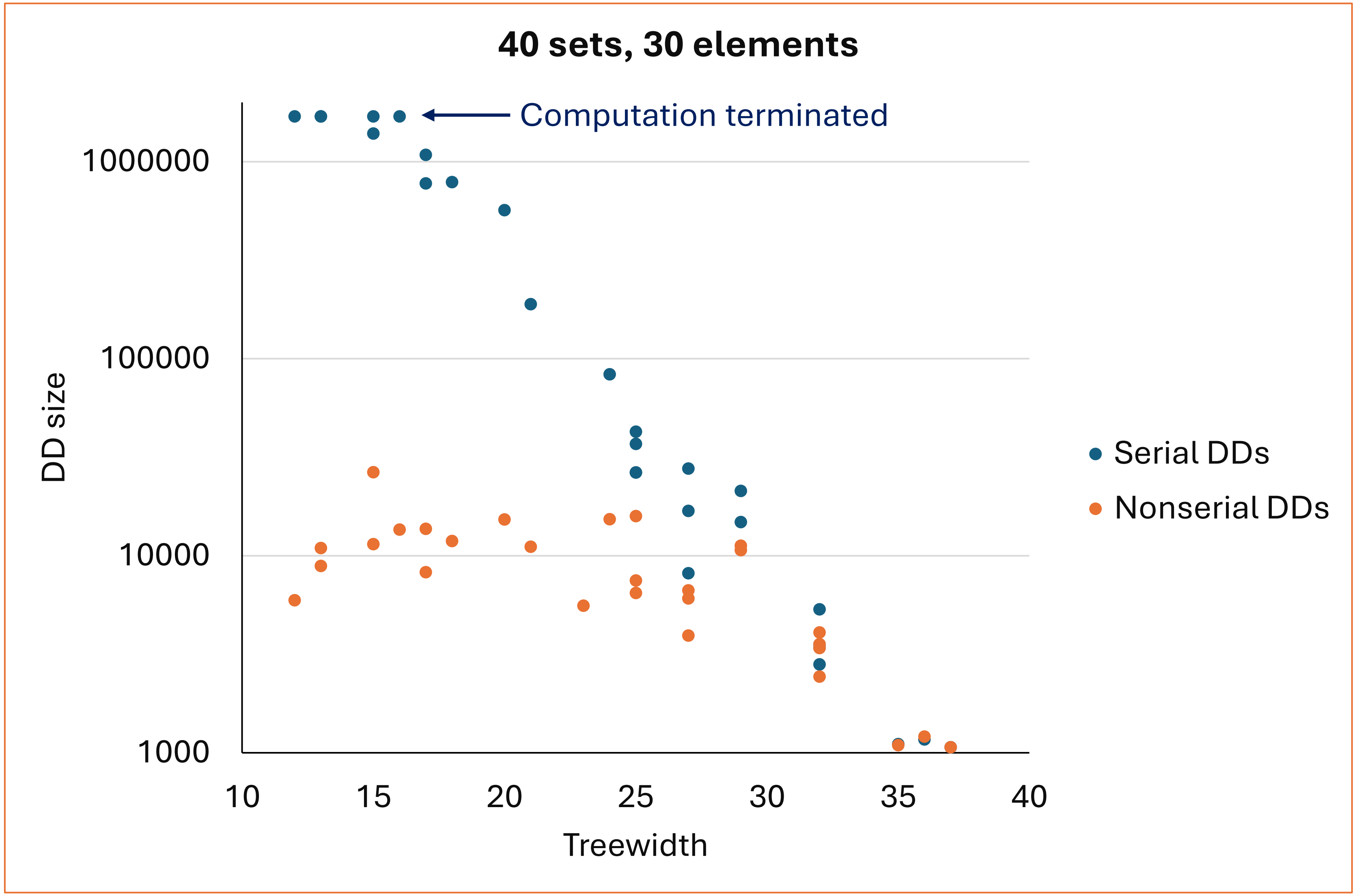


Average 2.4-6 elements/set

Serial and **nonserial**
DD size vs treewidth

Some **serial** DDs are too large to build.

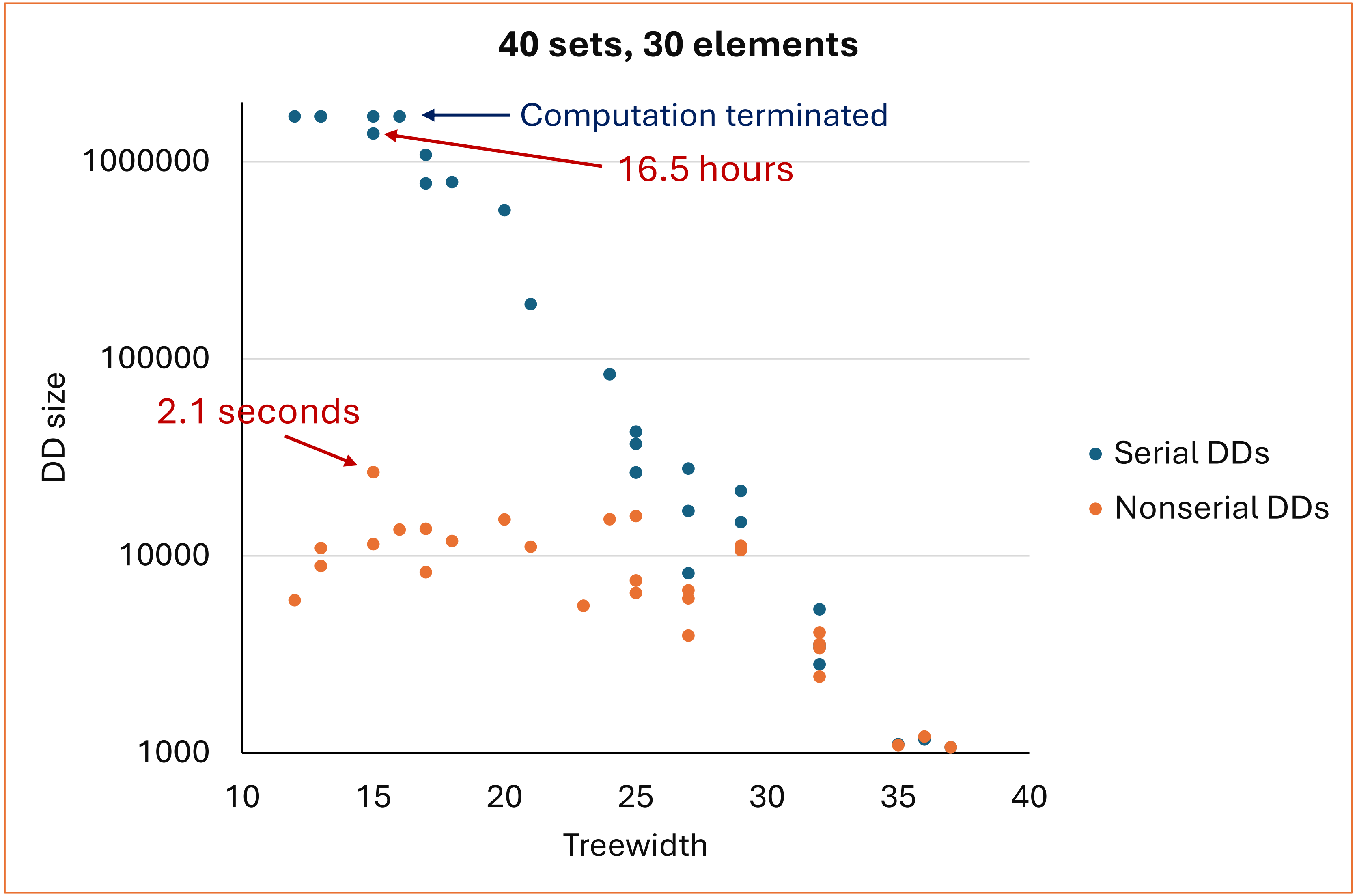
Nonserial DD size again levels off with smaller treewidths



Average 2.4-9 elements/set

Serial and **nonserial**
DD size vs treewidth

Compile time advantage of nonserial DD is again even greater than **size** advantage.



Average 2.4-9 elements/set

Conclusion…

For set packing problems, **nonserial DDs** are **very helpful** when you **need them,** and are **not helpful** when you **don’t need them.**

Future research…

Examine **other problem classes**.

Conjectures

- We should **always use nonserial DDs** in DD applications.
- There is **no computational penalty** for doing so.
- There are **enormous computational benefits** when treewidth is limited.
- All DD technologies easily **generalize** to the nonserial case (reduction, relaxation, restriction, flow models)