# Contents

* Backtracking Algorithm with example of it working
* Example showing that the counter of the binary causal pairs is accurate
* Code for the file that reads groups from the message file and makes a trace

## Backtracking Algorithm:

Backtracking Algorithm that reads in traces, and gets all the causal pairs

Ran and tested on all the **synthetic traces**. Generates Files called common subsequences.

The code is in github, pairs.py

As it runs, I had it print progress in the console, to verify it’s correct

A computer screen with white text

Description automatically generated

### Example:

Getting pairs for: synthetic\_traces/traces/trace-large-20\trace-large-20-audio-membus.txt

[5 5 48 36 49 36 39 41 39 41 48 49 48 49 39 48 49 41 5 36 5 36 48 49 39 41 39 5 36 41 48 49 48 49 39 41 5 48 36 49 39 41 58 59 48 49 48 49 5 36 39 5 41 36 5 36 39 41 48 49 39 41 5 36 5 36 39 39 41 41 5 36 58 59 5 36 39 41 5 36 48 49 48 49 5 36 48 49 5 36 39 41 48 49 5 36 48 49 39 48 49 41 39 41 5 36 48 49 5 36 5 36 39 41 39 41 5 36 5 36 5 36 39 41 48 49 5 36 39 48 49 41 39 41 39 48 49 41 39 41 48 49 48 49 48 49 39 39 41 41 39 39 41 41 48 49 39 48 49 41 48 49 48 49 48 49 5 36 48 49 48 49 5 5 36 36 5 36 39 41 5 36 5 36 48 49 39 41 39 41 5 36 39 41 48 49 48 49 5 36 48 49 39 41 5 36 48 49 39 39 41 41 48 49 39 41 48 49 39 41 39 41 39 41 5 36 39 41 5 36 39 5 41 36 48 49 5 36 48 48 49 49 5 36 48 49 5 36 39 39 41 48 49 41 5 48 49 36 39 41 48 49 48 49 39 41 5 48 49 48 49 36 39 5 36 48 41 49 58 59 5 36 5 48 49 36 39 41 39 41 48 49 5 5 36 36 58 59 39 41 39 41 39 41 39 41 48 49 39 41 39 39 41 41 39 48 49 41 5 36 39 41 39 41 5 36 48 48 49 49 48 49 48 49 58 59 39 41 39 41 39 41 39 41 5 36 5 36 48 49 5 36 48 49 39 41 48 49 58 59 5 39 36 41 5 36 58 59 5 36 48 49 48 49 5 36 48 49 39 48 49 41 48 49 48 49 39 41 48 49 5 5 36 36 5 36 5 36 5 36 39 41 48 49 5 36 48 49 39 41 5 48 49 36 5 36 5 36 39 41 5 36 48 49 39 41 5 36 48 49 39 39 41 48 49 41 48 49 48 49 48 49 39 39 41 41 58 59 58 59 5 39 41 36 5 36 48 49 5 36 48 49 39 41 39 58 59 41 5 36 48 49 48 49 48 49 5 36 58 59 48 49 5 36 48 49 39 39 39 41 48 49 41 41 48 49 39 5 36 41 48 49 5 36 5 48 49 36 5 36 48 49 5 48 49 36 5 36 39 41 48 49 48 49 48 49 5 36 39 5 41 36 5 48 49 36 48 49 48 49 48 49 39 41 48 49 5 5 48 49 36 36 5 36 48 49 5 36 5 48 49 36 39 41 48 49 5 58 36 59 5 36 58 59 48 49 58 59 5 36 48 49 5 39 36 41 5 36 48 49 58 59 39 5 41 48 49 36 5 36 39 41 5 39 41 36 39 58 59 48 41 49 48 49 48 49 39 41 48 49 48 49 39 48 49 41 39 41 39 41 5 5 36 36 48 49 5 36 39 41 39 41 5 36 48 49 39 39 5 41 41 36 39 41 39 41 39 41 39 39 41 41 39 48 41 48 49 49 48 49 39 48 49 41 48 49 39 5 41 36 39 41 5 36 5 36 5 36 48 49 39 41 5 36 48 49 39 41 48 49 39 48 41 49 48 49]

Trying initial node: 5

Checking end node: 48

Found causal pair: (5, 48)

Remaining trace after removal: [36, 49, 36, 39, 41, 39, 41, 49, 49, 39, 49, 41, 36, 36, 49, 39, 41, 39, 36, 41, 49, 49, 39, 41, 36, 49, 39, 41, 58, 59, 49, 49, 36, 39, 41, 36, 36, 39, 41, 49, 39, 41, 36, 36, 39, 39, 41, 41, 36, 58, 59, 36, 39, 41, 36, 49, 49, 36, 49, 36, 39, 41, 49, 36, 49, 39, 49, 41, 39, 41, 36, 49, 36, 36, 39, 41, 39, 41, 36, 36, 36, 39, 41, 49, 36, 39, 49, 41, 39, 41, 39, 49, 41, 39, 41, 49, 49, 49, 39, 39, 41, 41, 39, 39, 41, 41, 49, 39, 49, 41, 49, 49, 49, 36, 49, 49, 36, 36, 36, 39, 41, 36, 36, 49, 39, 41, 39, 41, 36, 39, 41, 49, 49, 36, 49, 39, 41, 36, 49, 39, 39, 41, 41, 49, 39, 41, 49, 39, 41, 39, 41, 39, 41, 36, 39, 41, 36, 39, 41, 36, 49, 36, 49, 49, 36, 49, 36, 39, 39, 41, 49, 41, 49, 36, 39, 41, 49, 49, 39, 41, 49, 49, 36, 39, 36, 41, 49, 58, 59, 36, 49, 36, 39, 41, 39, 41, 49, 36, 36, 58, 59, 39, 41, 39, 41, 39, 41, 39, 41, 49, 39, 41, 39, 39, 41, 41, 39, 49, 41, 36, 39, 41, 39, 41, 36, 49, 49, 49, 49, 58, 59, 39, 41, 39, 41, 39, 41, 39, 41, 36, 36, 49, 36, 49, 39, 41, 49, 58, 59, 39, 36, 41, 36, 58, 59, 36, 49, 49, 36, 49, 39, 49, 41, 49, 49, 39, 41, 49, 36, 36, 36, 36, 36, 39, 41, 49, 36, 49, 39, 41, 49, 36, 36, 36, 39, 41, 36, 49, 39, 41, 36, 49, 39, 39, 41, 49, 41, 49, 49, 49, 39, 39, 41, 41, 58, 59, 58, 59, 39, 41, 36, 36, 49, 36, 49, 39, 41, 39, 58, 59, 41, 36, 49, 49, 49, 36, 58, 59, 49, 36, 49, 39, 39, 39, 41, 49, 41, 41, 49, 39, 36, 41, 49, 36, 49, 36, 36, 49, 49, 36, 36, 39, 41, 49, 49, 49, 36, 39, 41, 36, 49, 36, 49, 49, 49, 39, 41, 49, 49, 36, 36, 36, 49, 36, 49, 36, 39, 41, 49, 58, 36, 59, 36, 58, 59, 49, 58, 59, 36, 49, 39, 36, 41, 36, 49, 58, 59, 39, 41, 49, 36, 36, 39, 41, 39, 41, 36, 39, 58, 59, 41, 49, 49, 49, 39, 41, 49, 49, 39, 49, 41, 39, 41, 39, 41, 36, 36, 49, 36, 39, 41, 39, 41, 36, 49, 39, 39, 41, 41, 36, 39, 41, 39, 41, 39, 41, 39, 39, 41, 41, 39, 41, 49, 49, 49, 39, 49, 41, 49, 39, 41, 36, 39, 41, 36, 36, 36, 49, 39, 41, 36, 49, 39, 41, 49, 39, 41, 49, 49]

Trying initial node: 36

Checking end node: 49 Fail. 49 is causal to 36, but 36 isn’t causal to 49. 36 is a terminating node

Checking end node: 39 Fail

Checking end node: 41 Fail

Checking end node: 58 Fail

Checking end node: 59 Fail

No pairs were found for 36.

Note: can optimize during backtracking, automatic backtrack if the initial node is in the list of terminating nodes from the message file

Backtracking to previous initial node.

Backtracking...try with another pair since 5 48 returned false, try another pair.

Checking end node: 36

Found causal pair: (5, 36)

Remaining trace after removal: [48, 49, 39, 41, 39, 41, 48, 49, 48, 49, 39, 48, 49, 41, 48, 49, 39, 41, 39, 41, 48, 49, 48, 49, 39, 41, 48, 49, 39, 41, 58, 59, 48, 49, 48, 49, 39, 41, 39, 41, 48, 49, 39, 41, 39, 39, 41, 41, 58, 59, 39, 41, 48, 49, 48, 49, 48, 49, 39, 41, 48, 49, 48, 49, 39, 48, 49, 41, 39, 41, 48, 49, 39, 41, 39, 41, 39, 41, 48, 49, 39, 48, 49, 41, 39, 41, 39, 48, 49, 41, 39, 41, 48, 49, 48, 49, 48, 49, 39, 39, 41, 41, 39, 39, 41, 41, 48, 49, 39, 48, 49, 41, 48, 49, 48, 49, 48, 49, 48, 49, 48, 49, 39, 41, 48, 49, 39, 41, 39, 41, 39, 41, 48, 49, 48, 49, 48, 49, 39, 41, 48, 49, 39, 39, 41, 41, 48, 49, 39, 41, 48, 49, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 48, 49, 48, 48, 49, 49, 48, 49, 39, 39, 41, 48, 49, 41, 48, 49, 39, 41, 48, 49, 48, 49, 39, 41, 48, 49, 48, 49, 39, 48, 41, 49, 58, 59, 48, 49, 39, 41, 39, 41, 48, 49, 58, 59, 39, 41, 39, 41, 39, 41, 39, 41, 48, 49, 39, 41, 39, 39, 41, 41, 39, 48, 49, 41, 39, 41, 39, 41, 48, 48, 49, 49, 48, 49, 48, 49, 58, 59, 39, 41, 39, 41, 39, 41, 39, 41, 48, 49, 48, 49, 39, 41, 48, 49, 58, 59, 39, 41, 58, 59, 48, 49, 48, 49, 48, 49, 39, 48, 49, 41, 48, 49, 48, 49, 39, 41, 48, 49, 39, 41, 48, 49, 48, 49, 39, 41, 48, 49, 39, 41, 48, 49, 39, 41, 48, 49, 39, 39, 41, 48, 49, 41, 48, 49, 48, 49, 48, 49, 39, 39, 41, 41, 58, 59, 58, 59, 39, 41, 48, 49, 48, 49, 39, 41, 39, 58, 59, 41, 48, 49, 48, 49, 48, 49, 58, 59, 48, 49, 48, 49, 39, 39, 39, 41, 48, 49, 41, 41, 48, 49, 39, 41, 48, 49, 48, 49, 48, 49, 48, 49, 39, 41, 48, 49, 48, 49, 48, 49, 39, 41, 48, 49, 48, 49, 48, 49, 48, 49, 39, 41, 48, 49, 48, 49, 48, 49, 48, 49, 39, 41, 48, 49, 58, 59, 58, 59, 48, 49, 58, 59, 48, 49, 39, 41, 48, 49, 58, 59, 39, 41, 48, 49, 39, 41, 39, 41, 39, 58, 59, 48, 41, 49, 48, 49, 48, 49, 39, 41, 48, 49, 48, 49, 39, 48, 49, 41, 39, 41, 39, 41, 48, 49, 39, 41, 39, 41, 48, 49, 39, 39, 41, 41, 39, 41, 39, 41, 39, 41, 39, 39, 41, 41, 39, 48, 41, 48, 49, 49, 48, 49, 39, 48, 49, 41, 48, 49, 39, 41, 39, 41, 48, 49, 39, 41, 48, 49, 39, 41, 48, 49, 39, 48, 41, 49, 48, 49]

Trying initial node: 48

Checking end node: 49

Found causal pair: (48, 49)

Remaining trace after removal: [39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 58, 59, 39, 41, 39, 41, 39, 41, 39, 39, 41, 41, 58, 59, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 39, 41, 41, 39, 39, 41, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 39, 41, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 39, 41, 41, 39, 41, 39, 41, 39, 41, 58, 59, 39, 41, 39, 41, 58, 59, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 39, 41, 41, 39, 41, 39, 41, 39, 41, 58, 59, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 58, 59, 39, 41, 58, 59, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 39, 41, 41, 39, 39, 41, 41, 58, 59, 58, 59, 39, 41, 39, 41, 39, 58, 59, 41, 58, 59, 39, 39, 39, 41, 41, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 58, 59, 58, 59, 58, 59, 39, 41, 58, 59, 39, 41, 39, 41, 39, 41, 39, 58, 59, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 39, 41, 41, 39, 41, 39, 41, 39, 41, 39, 39, 41, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41, 39, 41]

Trying initial node: 39

Checking end node: 41

Found causal pair: (39, 41)

Remaining trace after removal: [58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59, 58, 59]

Trying initial node: 58

Checking end node: 59

Found causal pair: (58, 59)

Remaining trace after removal: []

Successfully resolved the trace.

Then, it prints the counter of all the pairs to a file

A screenshot of a computer program

Description automatically generated

## Counter Accuracy Verification

Example: trace small 5, common subsequences file. Is count accurate?

A screen shot of a computer

Description automatically generated

Looking at trace-small-5-cache0-cpu0:

[0 25 2 0 26 25 0 25 2 26 2 26 2 26 2 26 0 2 25 26 2 26 2 0 26 25 0 25 2 26 2 26 0 25 2 26 2 26 0 25 0 25 2 26 0 25 0 25 2 26 2 0 25 26 2 26 2 26 0 25 2 2 26 26 0 25 0 25 0 25 0 25 0 25 0 25 2 0 26 25 0 25 2 26 2 26 0 25 0 25 2 26 0 25 0 25 0 25 0 25 0 25 0 25 2 2 26 26 0 2 25 26 0 25 0 25 2 2 26 26 0 2 26 25 2 26 2 26 0 25 2 26 2 26 0 25 2 26 2 26]

Getting rid of 0 25 pairs  (right next to each other)

Getting rid of remaining 0 

Getting rid of remaining 25

29 + 6 = 35

Getting rid of 2 26 pairs  (right next to each other)

Getting rid of remaining 2A black and white text

Description automatically generated

Getting rid of remaining 26A black and white text

Description automatically generated

29+9=35

Also tested on the backtracking example, counter was accurate. Anything with 0 was a causal pair found but it didn’t satisfy the trace.

Will start on technical report. In the meantime here is a basic explanation of a python file that gets the traces for the synthetic small files.

"""

Message File processing

-Read the message file. ignore # or whitespace.

-example line-> 0 : cpu0:cache0:wt:req

-so read as index: src, dest, cmd, type.

-Keys: src, dest.

-Store in a dictionary. the key is the src,dest. For example, 'cache 0' 'cpu0'

('cache0', 'cpu0', (0, 2, 25, 26))

0 : cpu0:cache0:wt:req

2 : cpu0:cache0:rd:req

25 : cache0:cpu0:wt:resp

26 : cache0:cpu0:rd:resp

Groups are tuples with the src, dest, and group of indices corresponding to that

"""

#Read in the msg file and extract the pairings in the data flow.

#1 and 2 are a pair if: src1 = dest2, dest1=src2. cmd1=cmd2. if type1 is resp, type2 must be req and vice versa.

def extract\_groups\_from\_msg\_file(file\_path):

    group\_indices = {}

    with open(file\_path, 'r') as file:

        for line in file:

            line = line.strip()

            # Ignore lines that start with #

            if line.startswith('#'):

                continue

            elif line:

                parts = [part.strip() for part in line.split(':')]

                if len(parts) == 5:

                    index, src, dest, cmd, type\_ = parts

                    key = tuple(sorted((src, dest)))  # Sorting to consider src, dest and dest, src as the same

                    if key not in group\_indices:

                        group\_indices[key] = []

                    group\_indices[key].append(int(index))

    groups = set()  # Avoid duplicates

    for key, indices in group\_indices.items():

        src, dest = key

        group = (src, dest, tuple(sorted(indices)))  # Include the indices

        groups.add(group)

    return sorted(groups)  # Return sorted groups

"""

Read a trace txt file into a list. Ignore the delimiters, and stop at -2

"""

#Read in the trace file into a list

def read\_trace\_file(file\_path):

    numbers = []

    with open(file\_path, 'r') as file:

        for line in file:

            parts = line.strip().split()

            for part in parts:

                if part == '-1':

                    continue

                elif part == '-2':

                    return numbers

                else:

                    numbers.append(int(part))

    return numbers

"""

For each number in a trace, see if the number is in the group of indices.

for example, 0 is in ('cache0', 'cpu0', (0, 2, 25, 26))

so I will append that number into the group sequence for trace-small-5-cache0-cpu0.

so the extracted sequence is:

0 25 2 0 26 25 0 25 2 26 2 26 2 26 2 26 0 2 25 26 2 26 2 0 26 25 0 25 2 26 2 26 0 25 2 26 2 26 0 25 0 25 2 26 0 25 0 25 2 26 2 0 25 26 2 26 2 26 0

25 2 2 26 26 0 25 0 25 0 25 0 25 0 25 0 25 2 0 26 25 0 25 2 26 2 26 0 25 0 25 2 26 0 25 0 25 0 25 0 25 0 25 0 25 2 2 26 26 0 2 25 26 0 25 0 25 2 2

26 26 0 2 26 25 2 26 2 26 0 25 2 26 2 26 0 25 2 26 2 26

Write it to a file

"""

#extract the sequences and output into file names based on the src/dest

def extract\_sequences(trace, groups, name):

    sequences = []

    # Initialize sequences for each group

    group\_sequences = {group: [] for group in groups}

    # Iterate through the trace list

    for num in trace:

        # Check if the number belongs to any group

        for group in groups:

            if num in group[2]:  # Check if num is in group indices

                group\_sequences[group].append(num)

    # Write sequences to files

    for group, sequence in group\_sequences.items():

        src, dest = group[0], group[1]

        filename = f"{name}-{src}-{dest}.txt"

        with open(filename, "w") as file:

            file.write(" ".join(map(str, sequence)))

        sequences.append(sequence)

    return sequences

#functions here

####################################################################

file\_path = "synthetic\_traces/newLarge.msg"

groups = extract\_groups\_from\_msg\_file(file\_path)

for g in groups:

    print(g)

####################################################################

# file\_path = "trace-small-5.txt"

# trace\_list = read\_trace\_file(file\_path)

# print("TRACE LIST", file\_path, trace\_list)

# sequences = extract\_sequences(trace\_list, groups, 'trace-small-5')

# print(f"Extracted sequences for {file\_path}")

# for seq in sequences:

#     print(seq)

# # ####################################################################

# file\_path = "trace-small-10.txt"

# trace\_list = read\_trace\_file(file\_path)

# print("TRACE LIST", file\_path, trace\_list)

# sequences = extract\_sequences(trace\_list, groups, 'trace-small-10')

# print(f"Extracted sequences for {file\_path}")

# for seq in sequences:

#     print(seq)

# # ####################################################################

# file\_path = "trace-small-20.txt"

# trace\_list = read\_trace\_file(file\_path)

# print("TRACE LIST", file\_path, trace\_list)

# sequences = extract\_sequences(trace\_list, groups, 'trace-small-20')

# print(f"Extracted sequences for {file\_path}")

# for seq in sequences:

#     print(seq)

# # ####################################################################

This was adapted for the large traces.

For multiple traces

And for Gem5, it was the same exact method, just reading a different message file and making adjustments on how its read