Discover Combinatorial Structures using Deep Cross-Entropy Method

Ryan O' Connor[1], Aimen Taha[1], Ananta Manoranjan[2], Saurabh Ray[2], and Deepak Ajwani[1]

[1]University College Dublin, Dublin, Ireland
[2]New York University, Abu Dhabi

1 Assignment Description

This assignment introduces students to the cutting-edge intersection of machine learning and combinatorial optimization. Many practical problems in industry and scientific research hinge on challenging combinatorial optimization tasks. Machine-learning–based techniques offer a promising way to tackle these challenges, with reinforcement learning standing out as one of the most effective approaches, particularly in settings where only limited data is available. The core of this assignment is to discover new combinatorial structures using a reinforcement learning technique called Cross-Entropy Method (CEM). Towards this end, the assignment builds on Wagner's code[1] associated with the paper "Constructions in combinatorics via neural networks"[2]) to find a series of simple combinatorial structures. Unlike traditional methods that rely on exhaustive search or specialized heuristics, Wagner's algorithm leverages the deep cross-entropy method to find effective configurations, providing a novel and powerful approach to these problems. This assignment is designed to not only solidify your understanding of core ML concepts like neural networks and encodings but also to expose you to the cutting-edge ML techniques for discovering combinatorial configurations.

You will be tasked with implementing and applying Wagner's algorithm, specifically its permutation-based approach, to solve three distinct CO problems. The solutions will be implemented in Python, leveraging the provided framework. A key part of this assignment involves understanding how to represent combinatorial structures as bit vectors for a neural network and designing appropriate scoring functions to guide the optimization process.

2 General Instructions

For all tasks, you will use Wagner's implementation of the deep cross-entropy method to solve the combinatorial optimization problems. Your implementation should be in Python,

[1]https://github.com/zawagner22/cross-entropy-for-combinatorics
[2]https://arxiv.org/abs/2104.14516

following the conventions and structure of the provided sample solution code for task 0 in the file task0_solution.ipynb. The following can be modified to solve the various structure search problems:

- `N`: a variable that defines the number of elements in the arrangement/sequence.
- `DECISIONS`: a variable that defines the size of the bit string the neural network should produce
- `calcScore()`: the reward function that decodes the bit vector into a combinatorial configuration and applies the scoring function to return a score

3 Configuration Representation

The problems in this assignment are order-based, meaning each configuration can be represented using permutations. However, the neural network does not directly output a permutation. Instead, it generates a bit vector, which must then be converted into a permutation for evaluation. This two-step process (encoding a bit vector into a permutation and then evaluating that permutation with a scoring function) is central to the assignment. It allows the neural network to learn patterns in the bit vectors that correspond to high-quality permutations, effectively learning an optimal ordering for a given problem.

4 General Solution Requirements

Your submission should include a written explanation of your chosen encoding and scoring function for each task.

Permutation encoding. You must design and implement a bit vector to permutation encoding scheme. Given n = 16, a straightforward approach is to use 4 bits per number, resulting in a 64-bit vector. Since the problem requires a distinct ordering (a permutation), you must develop a strategy to handle duplicates and ensure all numbers from 0 to 15 appear exactly once. A common approach involves keeping a list of available numbers and using the decoded number to select an item from this list, removing it after selection. You should explain your chosen method clearly. For rectangle configurations, a permutation could represent the order in which rectangles are placed or their relative positions. Explain how your chosen permutation maps to the 2D coordinates and properties (e.g., width, height) of the rectangles.

Scoring function. Design a scoring function that returns a high value for a configuration that satisfies the constraints and a low value for one that does not.

5 Task 0: Maximizing Subsequence Lengths

Create a sequence of $n = 16$ distinct numbers from the set $\{0, 1, \cdots, 15\}$ such that the minimum of the length of the Longest Increasing Subsequence (LIS) and the Longest Decreasing Subsequence (LDS) is maximized. We seek to find a permutation π that maximizes

the objective function $\max_{\pi \in \Pi} \min\{\text{length}(\text{LIS}(\pi)), \text{length}(\text{LDS}(\pi))\}$ where Π is the set of all possible permutations of the sequence $\{0, 1, \cdots, 15\}$. Here $\text{LIS}(\pi)$ and $\text{LDS}(\pi)$ denote the Longest Increasing Subsequence and the Longest Decreasing Subsequence of π, respectively.

Example: Consider the permutation π = [12,2,4,7,14,1,5,13,6,3,0,10,9,11,15,8].

The LIS is [2,4,5,6,10,11,15], with a length of 7.

The LDS is [12,7,6,3,0] (or [14,13,10,9,8]), with a length of 5.

The score for this permutation is min{7,5} = 5.

6 Sample Solution for Task 0

Encoding. The goal is to turn a neural network's bit output into a valid sequence of numbers, called a *permutation*, where each number appears exactly once. For a set of 16 numbers (0-15), we use 4 bits to represent each number ($2^4 = 16$), resulting in a 64-bit vector from the network.

Simply converting each 4-bit chunk into a number could result in duplicates, which is not a valid permutation. To solve this, we use a more sophisticated method:

1. Start with a list of all available numbers (0-15).
2. Iterate through the 16 chunks of the 64-bit vector, converting each chunk into a decimal value.
3. Use this decimal value as an index to select and remove a number from the current list of available numbers. We use the modulo operator to ensure the index is always valid (e.g., if the list has 10 numbers left and our index is 12, we use index 3 = 12 mod 9 instead. Note that we start our indices from 0.).
4. Add the selected number to our final sequence.

This "select and remove" method guarantees that each number is chosen precisely once, producing a unique permutation ready for scoring.

Scoring Function. A sequence's quality is measured by its score, which is the minimum of the lengths of its Longest Increasing Subsequence (LIS) and its Longest Decreasing Subsequence (LDS). This can be understood using a Directed Acyclic Graph (DAG) (though one can also directly use a dynamic programming approach).

Longest Increasing Subsequence (LIS). Imagine each number in the sequence is a node in a graph. An arrow points from one number to another if the second number appears later in the sequence and is larger. The LIS length is the longest path in this graph.

Longest Decreasing Subsequence (LDS). Similarly, we create a new graph where an arrow points from one number to another if the second number appears later and is smaller. The LDS length is the longest path in this second graph.

Finally, the score is determined by the objective function, which seeks to maximize the minimum of the LIS and LDS lengths. The score for a given permutation π is

$$\min\{\text{length}(\text{LIS}(\pi)), \text{length}(\text{LDS}(\pi))\}.$$

Code Snippets.

```python

import math
import networkx as nx

#Decodes a bit vector into a unique permutation of numbers.
def bits_to_permutation(bits, N):
    k = math.ceil(math.log2(N)) #number of bits needed for each number

    #creating sub-vectors and converting binary to decimal
    sublists = [bits[i * k:(i + 1) * k] for i in range(N)]
    values = [int(''.join(str(b) for b in sub), 2) for sub in sublists]

    available = list(range(1, N + 1)) #available distinct numbers to pick from
    permutation = []

    for val in values:
        index = val
        if index >= len(available): #value out of range, wrap around with
        ↪  modulo
            index = index % len(available)
        chosen = available.pop(index) #take chosen number out of available
        ↪  number pool
        permutation.append(chosen)#add to permutation

    return permutation

#Creates a Directed Acyclic Graph (DAG) for LIS or LDS calculation.
#Directed Acyclic Graph where each number has an edge to all numbers ahead of
↪  it that are > or < than it
def create_dag(ordering, increasing=True):
    G = nx.DiGraph()

    for i in ordering:
        G.add_node(i, value=i)

    n = len(ordering)

    #calculate LIS
    if increasing:
        for i in range(n):
            for j in range(i + 1, n):
                if ordering[i] < ordering[j]:
                    G.add_edge(ordering[i], ordering[j])
    #calculate LDS
    else:
        for i in range(n):
            for j in range(i + 1, n):
                if ordering[i] > ordering[j]:
                    G.add_edge(ordering[i], ordering[j])

    return G

#Calculates the score based on the lengths of LIS and LDS.
```

```python
def objective_function(ordering):
    increasing_dag = create_dag(ordering,increasing=True)
    decreasing_dag = create_dag(ordering,increasing=False)

    lis = nx.dag_longest_path(increasing_dag)
    lds = nx.dag_longest_path(decreasing_dag)
    return min(len(lis), len(lds))

#Main function to calculate the score of a given state (bit vector).
#N defined earlier in the algorithm
def calcScore(state):
    ordering =  bits_to_permutation(state, N)
    return objective_function(ordering)
```

The full code is provided in the file `task0_solution.ipynb`.

7 Task 1: Minimizing Subsequence Lengths

This task is a variant of Task 0. Instead of maximizing the minimum length, you will minimize the maximum length of the LIS and LDS. The goal is to find a permutation where the longest chain (both increasing or decreasing) are as short as possible. The objective function is $\min_{\pi \in \Pi}\{\max(\text{length}(\text{LIS}(\pi)), \text{length}(\text{LDS}(\pi)))\}$.

8 Task 2: Rectangle Intersection

Draw $n = 5$ blue rectangles and n red axis parallel rectangles in the plane s.t. all pairs of rectangles of different color intersect and all pairs of rectangles with the same color do not intersect. This is, of course, an easy task for humans, but the point in this task is to have the RL algorithm discover the solution.

9 Task 3: Intersection-Constrained Rectangles

Create a configuration of $n = 8$ axis-parallel rectangles in the plane such that the following specific conditions are met: i) no three rectangles have a common intersection point, and ii) the largest subset of rectangles that are pairwise disjoint is exactly 3. A subset of rectangles is pairwise disjoint if no two rectangles in the subset intersect.