Complex networks exam

PCS master
Joseph Touzet

Problem 1 - Generation of instances of the random graph model

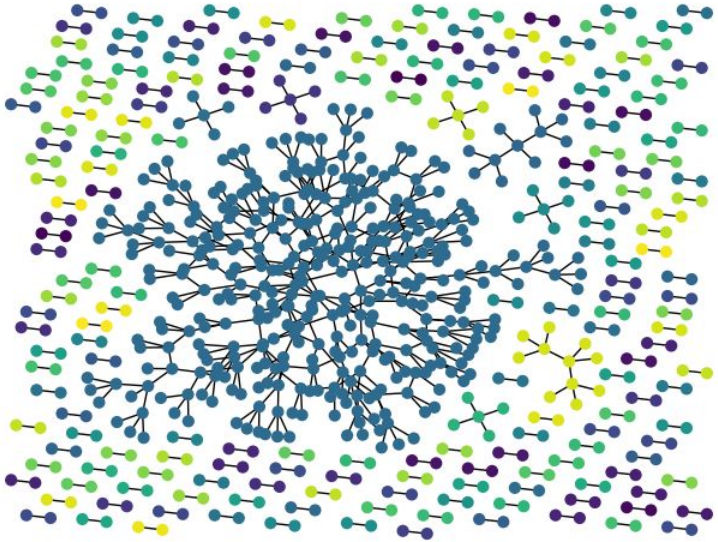
Code and algorithm

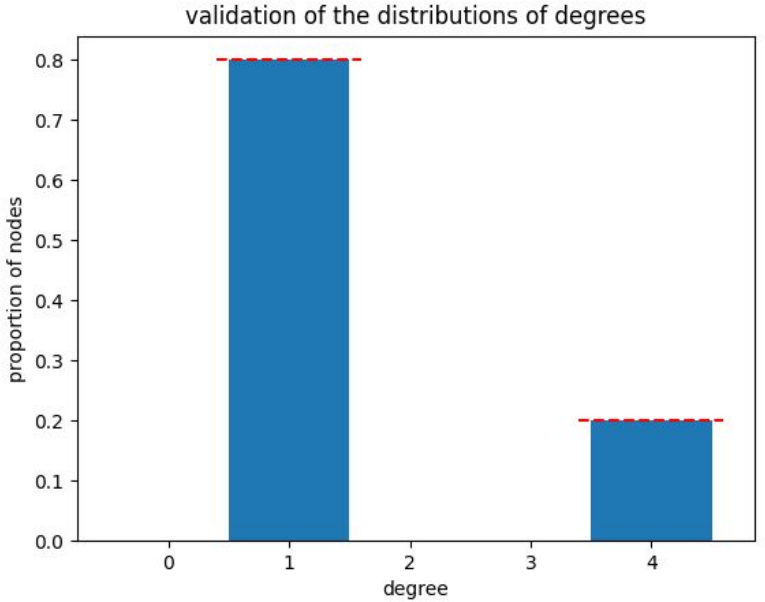
```python
def gen_rgm(N, pi):
    G = nx.Graph()
    for i in range(N):
        G.add_node(i)

    n_d1 = int(np.round(N*(1 - pi)/2)*2) # so that n_d1 + 4*n_d4 is pair
    n_d4 = N - n_d1

    stubs = []
    for i in range(n_d1):
        stubs.append(i)
    for i in range(n_d1, N):
        for j in range(4):
            stubs.append(i)

    while stubs:
        j = choose_and_pop_from_list(stubs)
        k = choose_and_pop_from_list(stubs)
        G.add_edge(j, k)
        G.add_edge(k, j)

    return G
```

```python
def gen_rgm_biased(N, pi):
    G = nx.Graph()
    for i in range(N):
        G.add_node(i)

    n_d1 = int(np.round(N*(1 - pi)/2)*2) # so that n_d1 + 4*n_d4 is pair

    # use a dictionary of degrees
    degree = {}
    for i in range(N):
        degree[i] = 1 if i < n_d1 else 4

    while len(degree) >= 2:
        unique_stubs = list(degree.keys())

        # select a node and remove it from the temporary list of nodes to select from
        j = choose_and_pop_from_list(unique_stubs)
        while True:
            # try to select another node and remove it from the temporary list
            k = choose_and_pop_from_list(unique_stubs)
            # only add edge if there isn't already an edge
            if not G.has_edge(j, k):
                G.add_edge(j, k)
                G.add_edge(k, j)

                # delete both index
                degree[j] -= 1
                # and if the degree is 0 remove the entry from the dict
                if degree[j] == 0:
                    del degree[j]

                degree[k] -= 1
                if degree[k] == 0:
                    del degree[k]

                break
            # if there is no more node to choose from:
            elif len(unique_stubs) == 0:
                # delete the entry for j because we know that we weren't able to create an edge from j
                del degree[j]
                break

    return G
```

Problem 1 - Generation of instances of the random graph model

Validation





Problem 2: The giant component

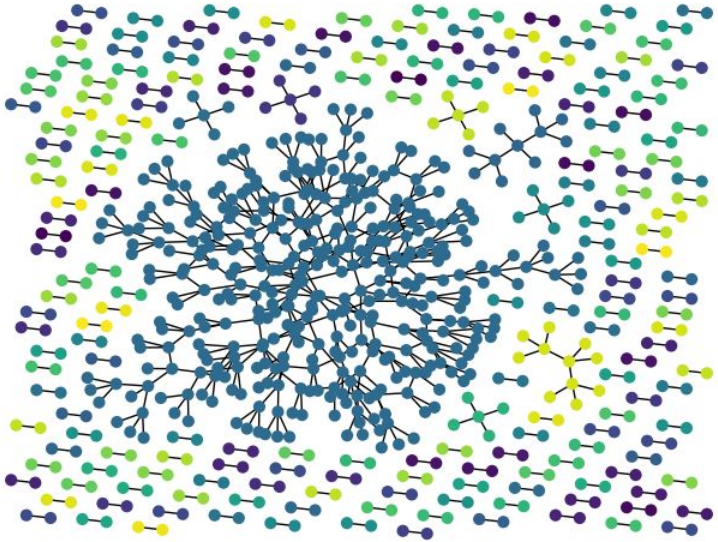
Code and algorithm

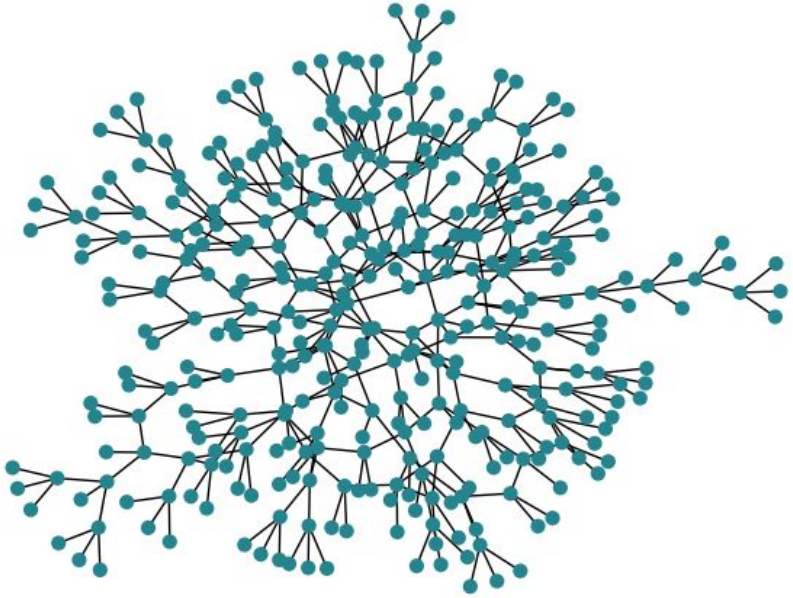
```python
def get_connected_component(graph, node, visited=None):
    if visited is None:
        visited = []
    visited.append(node)

    for neighbor in graph.neighbors(node):
        if neighbor not in visited:
            connected = get_connected_component(graph, neighbor, visited)

            for connected_node in connected:
                if connected_node not in visited:
                    visited.append(connected_node)
    return visited
```

```python
def find_largest_connected_component(graph):
    n_nodes     = graph.number_of_nodes()
    not_visited = list(graph.nodes)

    largest_connected_component = []
    while True:
        starting_node = not_visited[0]
        connected_component = get_connected_component(graph, starting_node)

        for node in connected_component:
            not_visited.remove(node)

        if len(connected_component) >= len(largest_connected_component):
            largest_connected_component = connected_component

        if len(not_visited) <= len(largest_connected_component):
            return largest_connected_component
```

Problem 2: The giant component

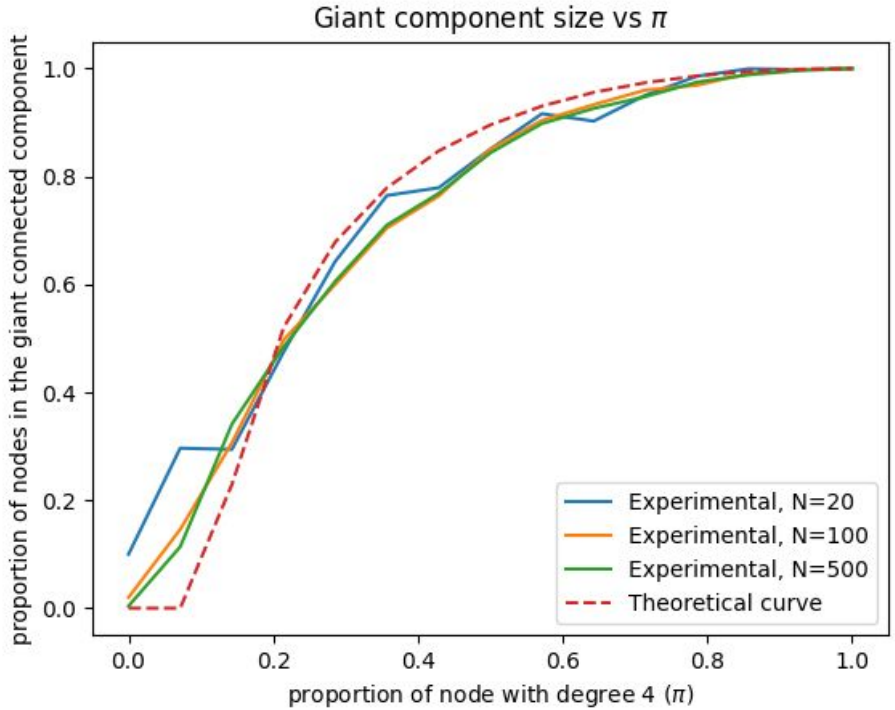
Validation

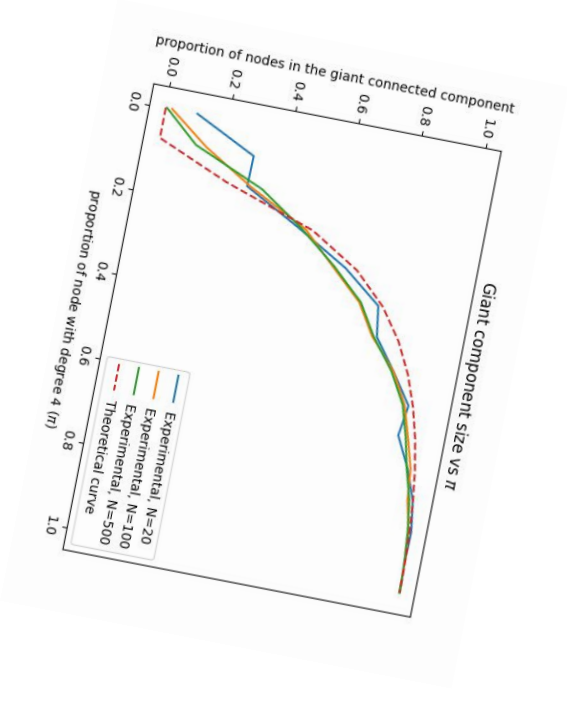




Problem 2: The giant component

Experiments and theory





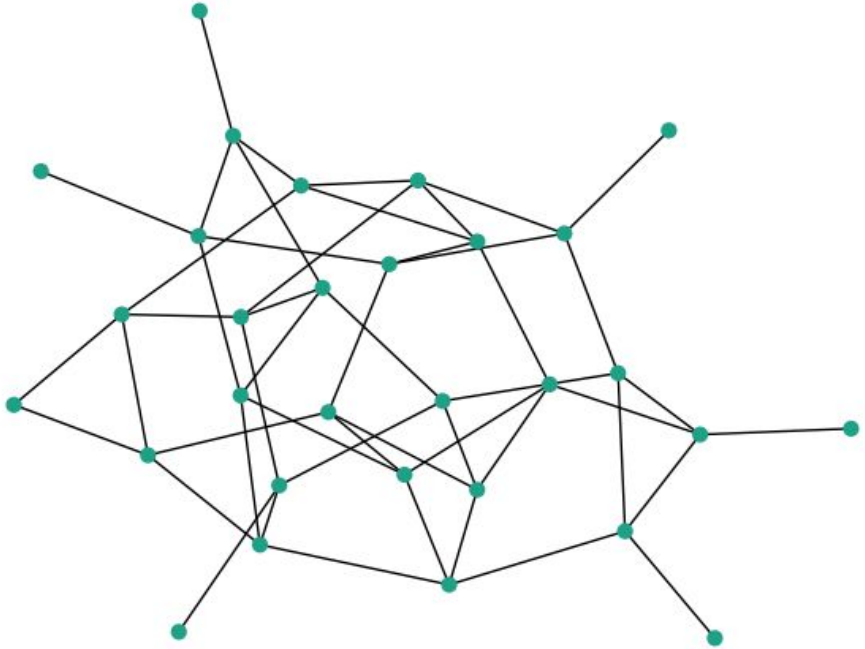
Problem 3: Emergence of the 3-core

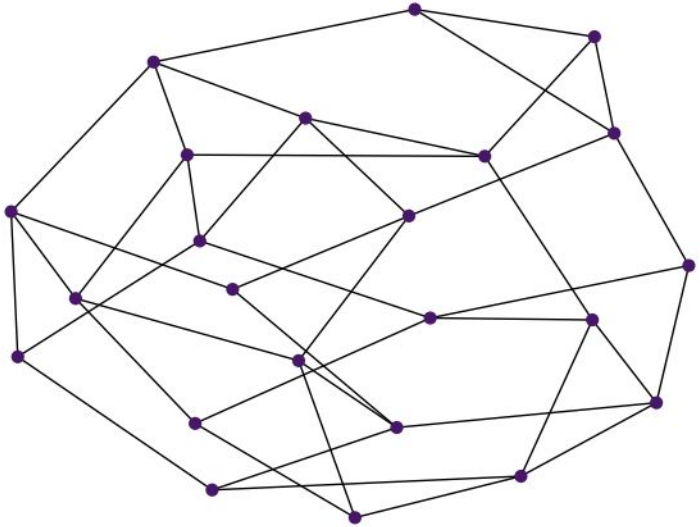
Code and algorithm

```python
# we use the prunning algorithm we saw in class:
def find_k_core(graph, k):
    G = copy.deepcopy(graph)
    while True:
        to_remove=[]

        for node in G.nodes:
            if G.degree[node] < k:
                to_remove.append(node)

        if len(to_remove) == 0:
            if len(G.nodes) == 0:
                return []

            # the k-core still has to be a single connected graph
            k_core = find_largest_connected_component(G)
            return k_core

        for node in to_remove:
            G.remove_node(node)
```

Problem 3: Emergence of the 3-core

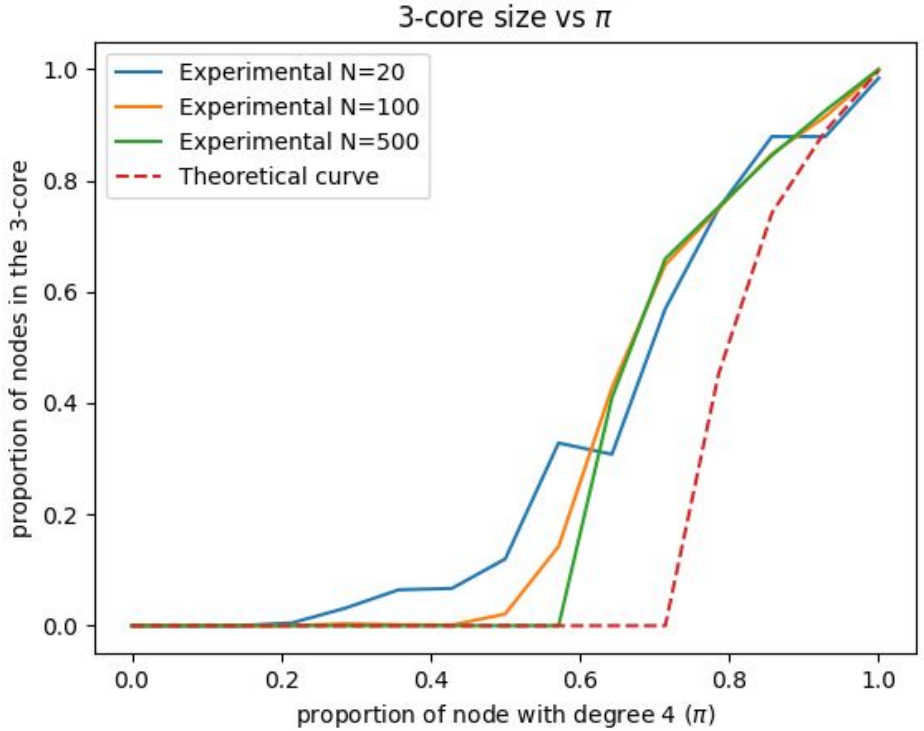
Validation

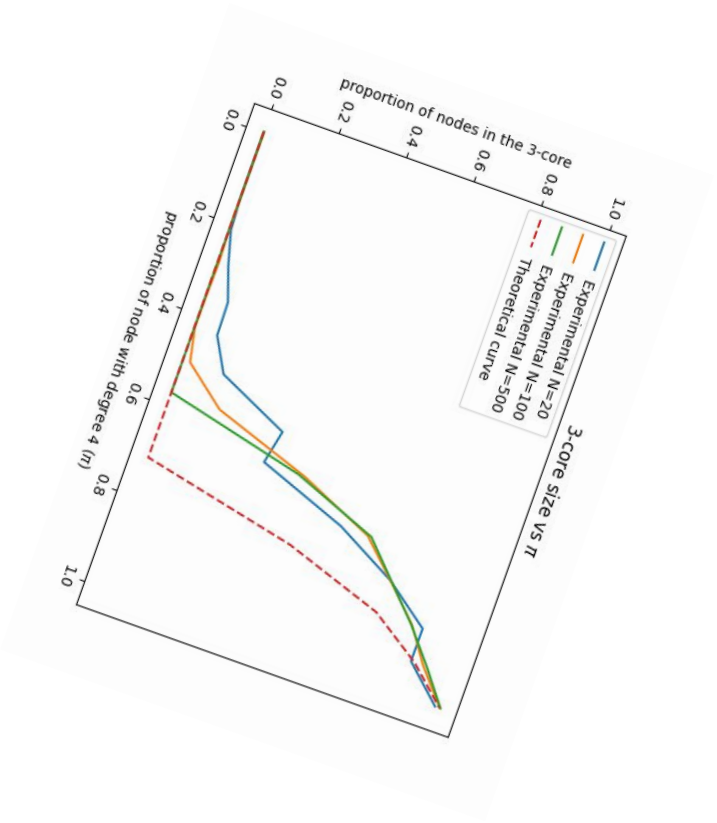




Problem 3: Emergence of the 3-core

Experiments and theory





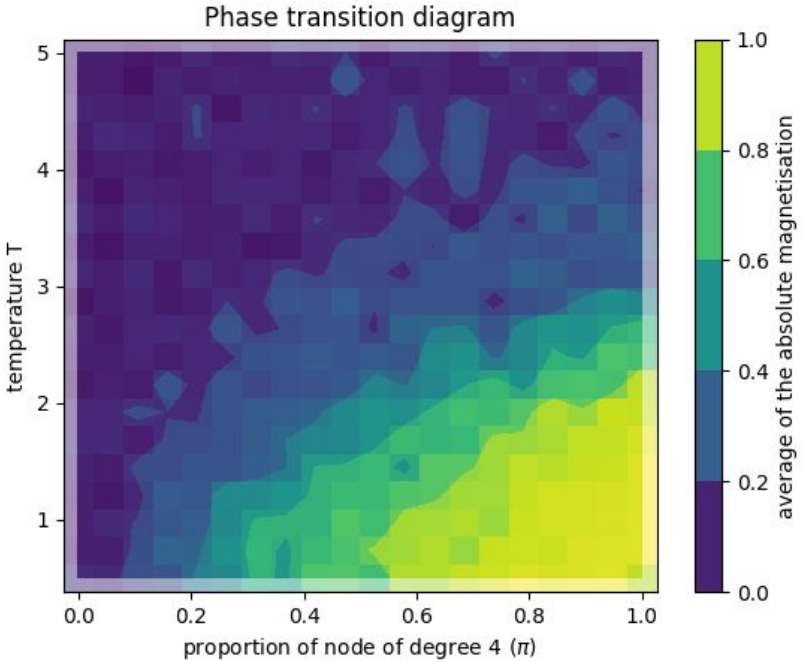
Problem 4: The ferromagnetic Ising model

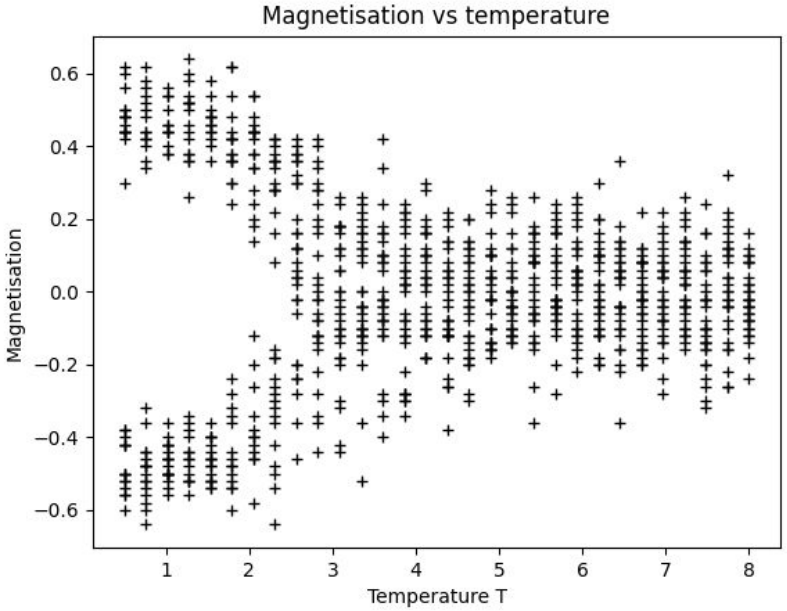
Code

```
def monte_carlo_simulate(graph, state, T, N_it_per_node):
    n_nodes = graph.number_of_nodes()

    for i in range(n_nodes*N_it_per_node):
        node = np.random.randint(0, n_nodes)

        total_surounding_spin = 0
        for neighbor in graph.neighbors(node):
            total_surounding_spin += state[neighbor]

        delta_E = 2*state[node]*total_surounding_spin
        if delta_E < 0:
            state[node] = -state[node]
        else:
            P_flip = np.exp(-delta_E/T)
            if np.random.rand() < P_flip:
                state[node] = -state[node]

    return state
```

Problem 4: The ferromagnetic Ising model

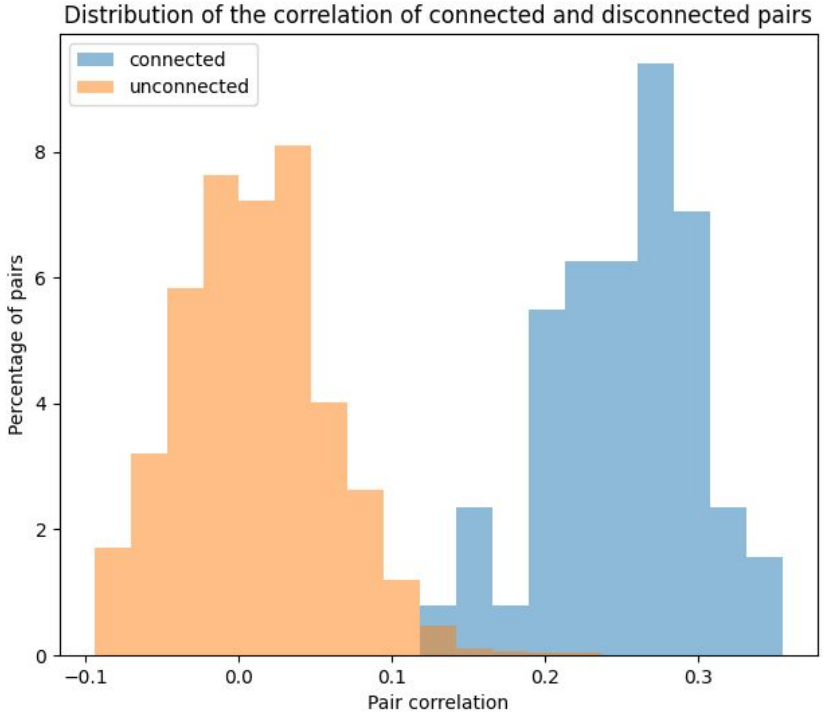
Experiments and theory

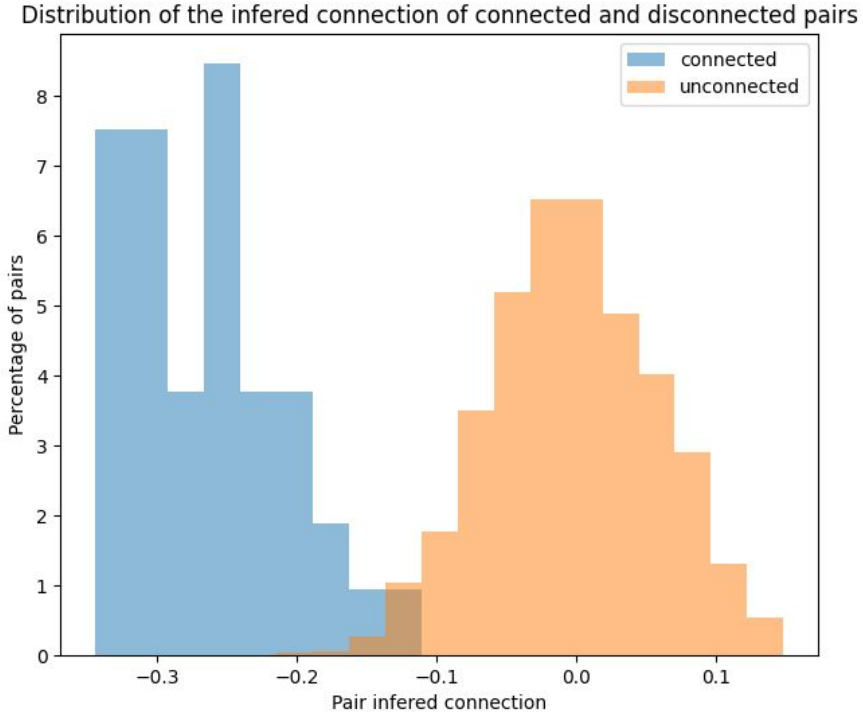




Problem 5: Inverse Ising model

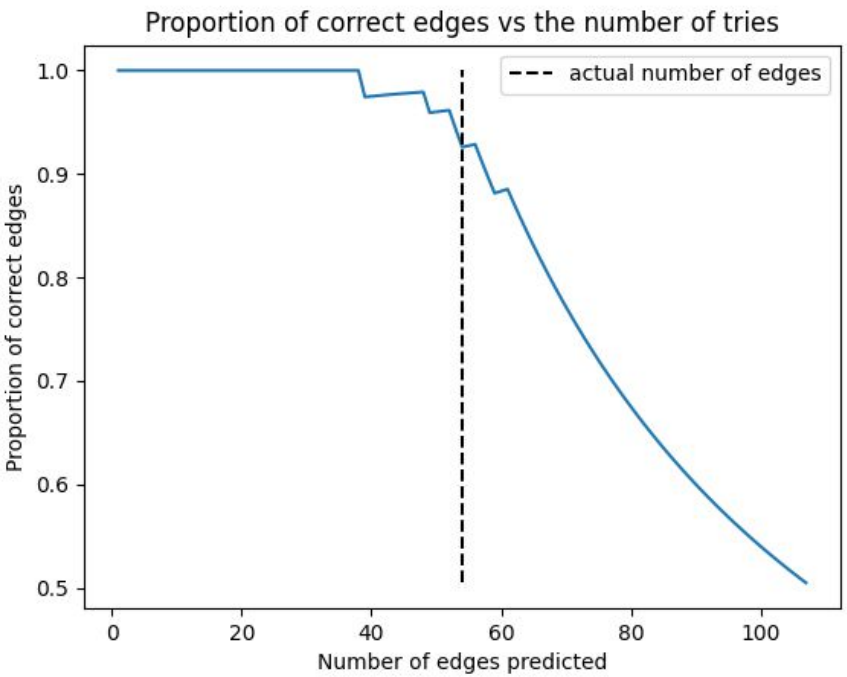
Experiments - connected vs unconnected edges separation

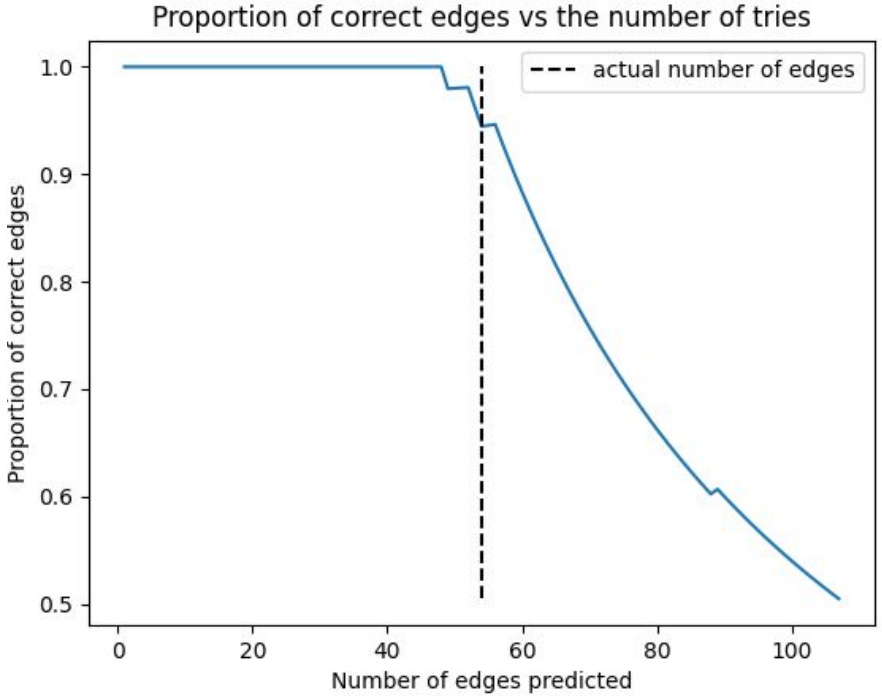




Problem 5: Inverse Ising model

Experiments - reconstruction efficiency





Conclusion and questions

Blabla…